FEATURE CONSTRUCTION TECHNIQUES IN TIME-SERIES ANALYSIS AND SINGLE-OBJECTIVE OPTIMIZATION

Gašper Petelin

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia

Supervisor: Assoc. Prof. Gregor Papa, Jožef Stefan Institute, Ljubljana, Slovenia

Evaluation Board:
Asst. Prof. Bernard Ženko, Chair, Jožef Stefan Institute, Ljubljana, Slovenia
Assoc. Prof. Niko Lukač, Member, The Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia
Prof. Dr. Boris Naujoks, Member, Cologne University of Applied Sciences, Köln, Germany

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Gašper Petelin

FEATURE CONSTRUCTION TECHNIQUES IN TIME-SERIES ANALYSIS AND SINGLE-OBJECTIVE OPTIMIZATION

Doctoral Dissertation

TEHNIKE GRADNJE ZNAČILK PRI ANALIZI ČASOVNIH VRST IN ENOKRITERIJSKI OPTIMIZACIJI

Doktorska disertacija

Supervisor: Assoc. Prof. Gregor Papa

Ljubljana, Slovenia, April 2025

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my girlfriend, whose support, patience, and encouragement have been invaluable throughout this journey. Your belief in me has been a constant source of motivation, and I cannot thank you enough for always being by my side.

I am also profoundly grateful to my family and friends, whose support, understanding, and encouragement have made this endeavor possible. Your kindness and belief in my abilities have given me strength during challenging times, and I truly appreciate everything you have done for me.

A special thank you goes to my supervisor, Assoc. Prof. Gregor Papa, for his invaluable guidance, patience, and insightful advice throughout my research. His expertise and mentorship have significantly shaped my work and academic growth. I would also like to extend my gratitude to my colleagues, past and present, for their support, collaboration, and inspiring discussions that have enriched my research experience.

Abstract

Feature construction, encompassing both feature engineering, which involves the manual design of features by domain experts, and representation learning, which refers to the automated discovery of useful data representations during model construction, is a fundamental aspect of machine learning. Its goal is to transform raw data into a more suitable form that can be exploited by machine learning approaches. While feature construction is well-explored in domains like natural language processing and computer vision, where it has been used in many applications, it has not been explored to the same extent in time-series analysis and remains largely unexplored in single-objective optimization.

In the domain of single-objective continuous optimization, only a handful of approaches exist. The primary goal of feature construction in this context is to generate features that capture properties of the objective function, such as the number of optima or the difficulty of finding them. Another important application of feature construction in optimization is algorithm selection, where the properties of the objective function are used to recommend the most suitable optimization algorithm for a given optimization function. While existing approaches in this area are capable of capturing some underlying properties of objective functions, they fall short, particularly in the task of recommending the most suitable optimization algorithm for a given objective function. This thesis proposes two new feature construction approaches. The first approach utilizes topological data analysis to extract features from samples drawn from a given objective function. This method captures various topological rotation and translation invariant properties that serve as descriptors of the objective function. The second proposed feature construction technique also uses samples drawn from the objective function, but applies random transformations to obtain a high-dimensional representation of the objective function. Both approaches are evaluated across multiple tasks, including differentiating between objective functions, predicting high-level properties, and algorithm selection. While these approaches demonstrate effectiveness for differentiating between problems and high-level property prediction, some limitations remain, particularly in algorithm selection.

The second part of the thesis focuses on feature construction techniques for time series data. We explore two tasks. The first is investigating the importance of time series features when used for selecting the most suitable forecasting algorithm. In this context, there are a handful of research approaches that have explored algorithm selection, but unfortunately, most of the proposed techniques only show that such techniques can be used for algorithm selection but do not provide explainability on how and why the most suitable algorithms are selected. We show that an explainable algorithm selection pipeline can be constructed, which provides consistent explainable decisions across different pipeline configurations. In the second task, we address the robustness of constructed representations for the task of classification. Time series classification has been largely explored, but there are still some challenges, such as how robust the constructed features are to various distortions and how this affects the accuracy of downstream classifiers. We explore the topic of robustness and provide some explanations as to why certain representations are more robust than others.

Povzetek

Gradnja značilk, ki vključuje tako ročno zasnovo s strani domenskih strokovnjakov kot tudi avtomatizirano učenje uporabnih predstavitev podatkov, je ključen vidik strojnega učenja. Glavni cilj je pretvoriti surove podatke v obliko, ki jo lahko učinkoviteje izkoristimo s pristopi strojnega učenja. Čeprav je gradnja značilk dobro raziskana na področjih kot sta obdelava naravnega jezika in računalniški vid, je na področju analize časovnih vrst raziskana v nekoliko manjši meri, medtem ko na področju enokriterijske optimizacije ostaja večinoma neraziskana.

Področje enokriterijske optimizacije ponuja le peščico pristopov za gradnjo značilk. Glavni cilj raziskovanja gradnje značilk v tem kontekstu je ustvariti značilke, ki opisujejo osnovne lastnosti optimizacijske funkcije, kot sta število ekstremov ali zahtevnost njihovega iskanja. Druga pomembna aplikacija gradnje značilk v optimizaciji je takoimenovana izbira algoritmov, kjer se lastnosti funkcije uporabljajo za priporočanje najustreznejšega optimizacijskega algoritma. Medtem ko obstoječi pristopi na tem področju sicer zajamejo nekatere osnovne lastnosti funkcij, imajo v veliki meri še vedno številne pomanjkljivosti, zlasti pri izbiri algoritmov. V okviru disertacije sta razviti dve novi metodi gradnje značilk. Prva metoda uporablja pristope topološke analize podatkov, kjer z vzorci, pridobljenimi iz optimizacijske funkcije, pokažemo, da je možno zgraditi značilke, ki so invariantne na rotacije in translacije v prostoru. Drugi pristop gradnje značilk temelji na uporabi naključnih transformacij, pri čemer so pridobljeni vzorci pretvorjeni v večje število značilk. Oba pristopa sta ovrednotena na različne načine, kot so razlikovanje med funkcijami, napovedovanje lastnosti funkcij in izbira algoritmov.

Drugi del te disertacije se osredotoča na tehnike gradnje značilk za podatke časovnih vrst. Prvi sklop se nanaša na pomen značilk časovnih vrst pri izbiri najustreznejšega napovednega algoritma. Na tem področju obstaja nekaj raziskovalnih pristopov, ki naslavljajo izbiro algoritmov, vendar večina teh pristopov zgolj pokaže, da lahko značilke uporabimo za izbiro algoritmov, vendar brez poglobljene analize, zakaj je bila izbrana posamezna metoda. V okviru te disertacije pokažemo, da je mogoče zasnovati razložljiv način izbire algoritmov, ki omogoča transparentne odločitve. Drugi sklop pa se nanaša na vprašanje robustnosti grajenih predstavitev pri klasifikaciji časovnih vrst. Čeprav je področje klasifikacije časovnih vrst dobro raziskano, ostajajo odprta vprašanja, kot so vpliv različnih deformacij na značilke ter kako te vplivajo na delovanje končnih klasifikatorjev. V disertaciji zato raziskujemo robustnost značilk in ponudimo razlago, zakaj so nekatere predstavitve bolj odporne na deformacije kot druge.

Contents

List of Figures

Abbreviations

AS ... Algorithm Selection
COCO ... Comparing Continuous Optimizers
ELA ... Exploratory Landscape Analysis
FLA ... Fitness Landscape Analysis
FS ... Feature Selection
LOIO ... Leave-One-Instance-Out
LOPO ... Leave-One-Problem-Out
ML ... Machine Learning
RFM ... Random Filter Mappings
TDA ... Topological Data Analysis
TLA ... Topological Landscape Analysis
TSC ... Time Series Classification

Chapter 1

Introduction

1.1 Motivation

In the field of machine learning, the quality of the input data and its transformation into meaningful features play a critical role in determining the success of predictive models (Zhong et al., 2016). For that reason, feature construction, which includes both feature engineering (where features are manually crafted by domain experts) and representation learning (where useful representations are typically learned during model construction, most often through gradient descent), is a crucial aspect to consider when modeling a problem. Essentially, it involves transforming raw data into a form that is better suited for machine learning algorithms. Feature construction thus plays a key role by helping to connect unprocessed data with the advanced algorithms built to analyze it. This transformation is vital because it allows algorithms to identify patterns and insights more efficiently, thereby improving their predictive power and ability to make sense of the data.

When constructing features, there are several key requirements that must be met to ensure they are effective. First, they must be useful for the task at hand, providing relevant information that aids in solving the problem and improving model performance. They should also be meaningful, with a clear relationship to the target variable and the problem domain, ensuring they add value. It is beneficial if features are explainable or interpretable (Linardatos et al., 2020), as this helps humans in understanding them, increasing transparency and trust in the model's decisions. Features should also be scalable, computationally feasible even with large datasets, and non-redundant, avoiding high correlation between features that could undermine model performance and interpretability. Additionally, they should be aligned with domain knowledge to reflect the true underlying structure of the data, ensuring they are relevant and contribute meaningfully to the task. Finally, other properties are also sometimes desirable, such as invariance (H. Zhao et al., 2019), where features remain stable across different domain-specific transformations, as this can help improve generalization while respecting the problem's characteristics.

Feature construction has been extensively explored in domains such as computer vision (Bengio et al., 2013; Jiang, 2009) and natural language processing (Z. Liu et al., 2023), where there is a large number of established techniques for extracting and engineering informative features from data. These fields have developed sophisticated methods to capture key patterns and structures that can improve the performance of machine-learning models. However, representation learning and, to some extent, feature engineering remain comparatively underdeveloped and less extensively studied in the fields of time series analysis and continuous single-objective optimization. In time series analysis, where the data often exhibits complex temporal dependencies, the challenge of constructing relevant and useful features becomes more intricate. Similarly, in single-objective optimization, feature

construction can play a crucial role in improving the search process and solution quality of optimization algorithms, yet it remains underexplored. The main topic of this thesis, therefore, is the development of new feature construction approaches tailored to these areas, as well as a detailed analysis of the behavior and effectiveness of different feature construction techniques. Through this work, the thesis aims to fill the gap in the literature and provide novel insights into how feature construction can enhance models in time series analysis and optimization tasks.

1.2 Problem Statement

This section provides a brief overview of some open questions in the field of feature construction for time series analysis and single-objective continuous optimization, and how these are addressed in our thesis. Much more thorough definitions are subsequently provided in the respective sections in later chapters.

1.2.1 Single-Objective Continuous Optimization

In single-objective continuous optimization, there are two key reasons for developing feature construction approaches.

The first reason is that feature construction can reveal additional insights into optimization functions (Mersmann et al., 2011). By constructing meaningful features that characterize the underlying properties of the objective function, researchers and practitioners can gain a deeper understanding of its behavior. For example, such features may uncover patterns like regions of local minima, the function's smoothness, or the presence of high-level characteristics such as multimodality, global structure, or funnel-shaped landscapes. Additionally, one might be interested in how far apart objective functions are embedded in the feature space and whether it is possible to differentiate between dissimilar functions. Understanding the features of objective functions can also aid in the development of more informative benchmarks for evaluating optimization algorithms. The ability to describe an objective function in terms of its features improves the comparability of different optimization tasks, enabling more systematic analysis.

The second, and perhaps even more important, reason for the need for feature construction in this context is the concept of optimization algorithm selection (AS) (Muñoz et al., 2015), a concept known as meta-learning (Vanschoren, 2019). Given a specific objective function, the goal is to select the most appropriate optimization algorithm that will perform best based on the characteristics of the objective function. Feature construction plays a pivotal role here by providing a structured way to capture the properties of the objective function. The most common approach involves generating a set of samples from objective functions, extracting features from them, and evaluating the performance of various optimization algorithms. A meta-model is then trained to map the extracted features to the best-performing algorithm. The main goal is for the meta-model to learn how properties such as search space complexity, noise levels, and landscape smoothness can be used to select the most suitable optimization algorithm for a given problem. By selecting optimization algorithms based on these features, it is possible to improve the efficiency and effectiveness of the optimization process, ultimately leading to better performance in terms of speed, accuracy, and robustness.

However, both of the listed reasons for feature construction are still largely unexplored. In terms of characterizing objective functions, existing approaches (with the best-known example being Exploratory Landscape Analysis (Mersmann et al., 2011) features) are not always sufficient. Existing features are often not invariant to transformations such as

translations, scaling, and rotation, meaning that simple transformations, which generally do not influence the difficulty of the objective function, can still alter some features (Škvorc et al., 2020). Additionally, some feature sets contain highly correlated features, which can lead to reduced interpretability if not mitigated properly with techniques such as feature selection or ranking. For the task of algorithm selection, the need for quality features and representations is even greater. Features should be informative enough to enable the recommendation of the best-performing optimization algorithm based on the computed features. However, it has been demonstrated that optimization algorithm selection only works if the new functions are extremely similar to those already observed during the training phase. If the functions deviate too much from the training set, machine learning models for algorithm selection fail to accurately choose the best optimization algorithm (Petelin & Cenikj, 2024a). It is, therefore, crucial to address the generalizability of newly proposed features.

1.2.2 Time Series Analysis

For time series analysis, feature engineering and representation learning have been explored more extensively compared to the optimization domain, though there are still significant opportunities for further research and improvement (Meng et al., 2023). There are quite a few feature construction approaches that have been developed to extract meaningful features from time series data. These techniques focus on transforming raw temporal data into more structured representations that can reveal patterns, trends, and seasonality, which are crucial for understanding the underlying processes. Various statistical methods, domain-specific heuristics, and even machine learning models have been employed to design features that can effectively represent the temporal dynamics of the data. These approaches have shown considerable success in recent years for a multitude of tasks, such as classification (Middlehurst, Schäfer, & Bagnall, 2024), forecasting (N. K. Ahmed et al., 2010), extrinsic regression (Tan et al., 2021), and even algorithm selection. As part of the thesis we mainly focus on two challenges of feature construction for time series analysis.

The first challenge we address is related to explainability in algorithm selection for forecasting. Over the years, numerous forecasting methods have been proposed, each with its own set of advantages and disadvantages. As a result, selecting the most suitable algorithm for a given forecasting task is not always straightforward. In practice, one way to address this is by training a machine learning meta-model to perform forecasting algorithm selection based on features extracted from a given time series. The goal of the meta-model is to learn the relationship between time series properties and the performance of forecasting algorithms. Once an appropriate algorithm is selected, it is then used to generate forecasts. While such approaches have been proposed, they often lack a focus on the explainability of the ML models responsible for making decisions about which forecasting algorithm to choose. This lack of transparency raises concerns about how and why particular algorithms are selected. Therefore, there is a need to better understand how ML models select the base forecasting algorithms based on time series features and, more importantly, to identify which features are most significant in guiding these decisions. Gaining this insight can improve the training of models for algorithm selection and help uncover areas where forecasting models might be performing poorly, thus highlighting potential weaknesses in existing approaches. Such knowledge is essential for refining forecasting models and algorithms, ultimately contributing to more robust and effective algorithm development in the field of time series forecasting.

The second challenge related to time series that we address in this thesis is the robustness of feature construction techniques and the models that utilize these features for classification. In this context, robustness refers to how features and the subsequent clas-

sification models respond when exposed to various naturally occurring distortions in the data, such as noise, missing values, or outliers. This is an important aspect for two key reasons. First, it enables us to gain a deeper understanding of the behavior of feature construction techniques, providing insights into how different types of distortions affect the quality and reliability of the features. By examining the impact of these distortions, we can better assess how well the features generalize across different conditions and how they influence the performance of downstream classifiers. Second, robustness is critical because it directly influences the performance of classification models in real-world scenarios. In many practical applications, data can be noisy or incomplete, and ensuring that features and classifiers can maintain their accuracy despite these challenges is essential for developing reliable, real-world models. Addressing the robustness of feature construction techniques allows us to create more robust models that perform consistently, even under less-than-ideal conditions.

1.3 Aims and Hypotheses

This thesis addresses open questions raised in the previous section and outlines the hypotheses used to guide the research. The first part explores novel approaches for feature construction in the continuous single-objective optimization domain. We propose two new methods for constructing features from samples of an objective function. These features can then be used for further downstream tasks. The second part of the thesis focuses on feature construction techniques in time series analysis. Specifically, we evaluate the explainability of meta-models for forecasting algorithm selection in terms of feature importance and the robustness of features to distortions in time series classification.

The first single-objective feature construction technique we propose draws on concepts from topological data analysis (TDA), a subfield of applied mathematics that uses techniques from topology to analyze datasets. TDA offers a variety of tools that extract topological properties, such as the appearance of holes, noise, and other structures, from samples of points. The proposed feature extraction techniques are evaluated on the tasks of problem differentiation, high-level property prediction, and algorithm selection. To evaluate the method, the following two hypotheses are proposed:

H1a: *Using insights from topological data analysis, it is possible to construct rotation and translation invariant features that differentiate between single-objective optimization problems and capture their high-level properties.*

H1b: *Performance of predictive models for optimization algorithm selection based on such topological features is comparable to the performance of models based on existing exploratory landscape analysis features.*

The second feature construction approach involves applying a set of random transformations or functions. Each random function maps samples obtained from the objective function into a corresponding set of features. The proposed random mapping technique works based on the distances between samples, mapping them into a high-dimensional space that can be used to represent objective functions. Similarly to the first approach, we evaluate this feature construction method on the tasks of problem differentiation, high-level property prediction, and algorithm selection. To evaluate the method, the following two hypotheses are proposed:

H2a: *Features based on randomly initialized filters calculated using distances between optimization problem samples are able to differentiate between distinct single-objective optimization problems and capture their high-level properties.*

H2b: *Performance of predictive models for optimization algorithm selection based on such features is comparable to the performance of models based on existing exploratory landscape analysis features.*

The next topics we explore are related to algorithm selection in the time series forecasting domain. While the use of feature representations of time series data to aid in selecting the most suitable forecasting model has been studied before, most approaches treat the algorithm selection model as a black box. In contrast, we explore how the algorithm selection model makes decisions and which features it considers important. We investigate the constructed features and their significance in successfully performing algorithm selection. Additionally, we evaluate the feature agreement between different algorithm selection models, as well as the agreement between individual forecasting algorithms. The hypothesis is defined as:

H3: *Meta-models for selecting time series forecasting algorithms are able to be explained using feature scoring techniques.*

The last topic that falls under the scope of this thesis is the evaluation of the robustness of representations to naturally occurring distortions in the context of time series classification. Similar to the previous hypothesis, where it is beneficial to have interpretable features, it is also important to understand how such features react to various distortions. For this reason, we explore the sensitivity of time series distortions on the constructed features and the subsequent accuracy of time series classification models. The hypothesis we propose is as follows:

H4: *By simulating the most frequent types of distortions, it is possible to empirically evaluate the robustness of different time series classification algorithms in terms of their classification accuracy in the presence of various data distortions.*

1.4 Scientific Contributions

Scientific contributions are grouped based on the hypotheses we proposed in the previous section. These correspond to new feature representations for single-objective optimization functions, as well as the explainability and robustness of features in the time series domain.

SC1: A new feature extraction approach based on topological data analysis is proposed for constructing representations of single-objective continuous optimization functions. These representations can be used for various tasks, such as problem differentiation, high-level property prediction, and optimization algorithm selection.

SC2: A new feature extraction approach based on randomly initialized functions is proposed for constructing representations of single-objective continuous optimization functions. These representations can be applied to various tasks, including problem differentiation, high-level property prediction, and optimization algorithm selection.

SC3: Explainability techniques are applied to the forecasting algorithm selection pipeline to make it more transparent regarding how decisions are made, and to identify which time series properties have the most substantial effect on the performance of time series forecasting algorithm.

SC4: Different naturally occurring time series distortions of increasing magnitudes are simulated and evaluated on how they affect constructed feature representation and how this influences the behaviour of downstream classifiers in terms of classification accuracy.

1.5 Thesis Structure

The introductory chapter of this thesis provides general information on the topic of feature construction and discusses its impact on the field of machine learning. This is followed by a description of the open problems and the problem definition of feature construction in the domains of continuous single-objective optimization and time series analysis. Next, the hypotheses explored in this thesis are presented. Finally, a separate section outlines the scientific contributions of the thesis.

This is followed by Chapter 2, where we discuss two newly proposed approaches for feature construction in representing continuous single-objective optimization functions. The first approach employs techniques from topological data analysis to describe optimization functions in terms of features. This is followed by another approach that uses random functions to map samples obtained from objective functions to their representations.

Chapter 3 focuses on the explainability and robustness of feature construction approaches in the domain of time series analysis. The first part of the chapter discusses forecasting algorithm selection and explores ways to enhance the explainability of this approach. It highlights how making forecasting algorithm selection more interpretable can reveal more information about features used to perform selection. The chapter then explores how the constructed features respond to naturally occurring distortions in time series data, such as noise and missing data. It further explores the impact of these distortions on the downstream task of classification, analyzing how the robustness of features influences the overall performance and reliability of classification models.

Lastly, Chapter 4 concludes our work by reviewing the scientific contributions of the thesis and discussing the key findings. This chapter also outlines potential directions for future research, highlighting areas where further investigation could build upon the current work and address existing challenges in the field.

Chapter 2

Feature Construction in Continuous Single-objective Optimization

Representing optimization problems in terms of features that can be used by machine learning models is crucial for tasks such as algorithm selection and configuration, performance prediction, high-level property prediction, and, in general, for understanding similarities between optimization problems. This section presents two different approaches for feature construction for single-objective optimization problems, primarily through the use of topological data analysis and the construction of randomly initialized filters/functions that transform samples into numerical features.

The first approach introduces TinyTLA features detailed in the paper *TinyTLA: Topological landscape analysis for optimization problem classification in a limited sample setting* published in the *Swarm and Evolutionary Computation* (Petelin et al., 2024). Based on current knowledge, this is one of the first papers to introduce topological concepts into the optimization domain. Such concepts are highly relevant in the field of optimization as they capture various properties of samples from an objective function, such as holes, toruses, valleys, and more. The paper further introduces concepts from topological data analysis, exploring how they can be used to construct features and whether those features are sufficiently discriminative to differentiate between problems with different properties.

The second approach for feature extraction, known as Random Filter Mappings, was published in a paper *Random Filter Mappings as Optimization Problem Feature Extractors* in *IEEE Access* (Petelin & Cenikj, 2024b). This paper introduces a novel method for extracting features by mapping objective function samples through random transformation functions. Similar techniques have been explored in other fields, such as computer vision and time series classification. Methodologically, this approach occupies a middle ground between manual feature engineering and deep-learning based feature construction, balancing aspects of both.

Note that the proposed approaches differ slightly in computational efficiency, memory usage, and the maximum number of samples from the objective function they can effectively process. For this reason, one must be cautious when comparing results, as the methodologies vary slightly. This is primarily due to the fact that computing the topological properties of a point cloud grows exponentially with sample size and problem dimension.

2.1 TinyTLA: Topological Landscape Analysis for Optimization Problem Classification in a Limited Sample Setting

Topological Data Analysis is a subfield of applied mathematics that utilizes tools from topology to extract meaningful information from high-dimensional data. Due to its ability to handle complex, multi-dimensional structures, TDA can be well-suited for applications in various fields, including continuous single-objective optimization. This section explores the use of TDA as a tool for constructing features from samples obtained through optimization functions. In the paper *TinyTLA: Topological landscape analysis for optimization problem classification in a limited sample setting*, we introduce the TinyTLA approach, a novel feature extraction method that leverages TDA to derive topological features from optimization problems. TinyTLA focuses on distances between sample points obtained from the optimization function. These distances are used to capture persistent topological structures representing geometric formations in the optimization landscape. Proposed features are invariant to rotation and translation, making them unique compared to existing feature extraction methods.

The paper explores hypotheses **H1a** and **H1b** and highlights two key contributions: firstly, demonstrating that topological features provide complementary insights to traditional landscape analysis; and secondly, offering a sensitivity analysis that shows the robustness of these features in scenarios with limited sample sizes. The results confirm that TinyTLA's topological features are effective in classifying optimization problems. The subsequent section also presents an additional analysis of the features' success in high-level property prediction and algorithm selection that was not included in the original paper.

Swarm and Evolutionary Computation 84 (2024) 101448

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

TinyTLA: Topological landscape analysis for optimization problem classification in a limited sample setting

Gašper Petelin [a,b,*], Gjorgjina Cenikj [a,b], Tome Eftimov [a]

[a] Computer Systems Department, Jožef Stefan Institute, Ljubljana, 1000, Slovenia
[b] Jožef Stefan International Postgraduate School, Ljubljana, 1000, Slovenia

ARTICLE INFO

Keywords:
Single objective numerical optimization
Topological data analysis
Topological landscape features

ABSTRACT

In numerical optimization, the characterization of optimization problems and their properties has been a long-standing issue. Overcoming it is a crucial prerequisite for many optimization-related tasks such as building quality benchmarks, algorithm selection, and algorithm configuration. Several approaches to extracting features from single-objective optimization problems have been proposed but they all have some inherent limitations and thus offer an incomplete look at the problems and their properties. In this work, we extend and improve our previous work on existing topological features that offer a new look at optimization problems where their similarity is quantified in terms of the appearance of topological structures. We show that topologically inspired features are not correlated with existing state-of-the-art landscape feature groups, meaning that they capture different and thus complementary information. Topological features are subsequently used to classify problem instances from the COmparing Continuous Optimizers (COCO) benchmark showing that similar problems most often have similar topological features. Further, we perform a sensitivity analysis of the proposed methodology to its hyperparameters and provide some additional insight into the behavior of the topological features. Lastly, we also investigate topological features and their generalization across different sample sizes.

1. Introduction

The representation of single-objective numerical continuous optimization problems in terms of numerical features, also known as problem landscape features, is a prerequisite for several tasks in numerical optimization. In automated algorithm selection (i.e., selecting the best algorithm for each problem) [1,2] and configuration (i.e., finding the best hyper-parameters for each algorithm) [3], the existence of numerical features characterizing problem instances enables the training of models based on Machine Learning (ML), where the calculated features are used to predict algorithms' performance. Such features are also indispensable in the field of benchmarking, where they can be used to conduct analyses of the similarity of problem instances [4], as well as the representativeness and redundancy of the problem space in existing benchmark suites [5], and the identification of problem landscape areas that are over or under-represented by the available benchmark suites [6]. In the past two decades, a significant amount of research effort has been dedicated to the construction of features that capture properties of single-objective continuous optimization problems. A large number of features have been proposed, which can be broadly categorized as features based on Fitness Landscape Analysis (FLA) [7] and features based on Exploratory Landscape Analysis (ELA) [8]. Despite being useful for a number of tasks, the existing features still have some properties that are often undesirable for the tasks of benchmarking and algorithm selection, such as large computational complexity in the case of high dimensional problems, requiring human effort to be generated, being sensitive to the number of samples and sampling technique from which they are calculated [9,10], and not being invariant to transformations such as scaling and shifting [4,10,11]. The issues mentioned above point out that additional research is required to generate features that will address some of the limitations of existing approaches. In [12], we have proposed the Topological Landscape Analysis (TLA) methodology, which applies approaches from Topological Data Analysis (TDA) to generate features of optimization problems based on the topological structures that can be identified in point clouds obtained by sampling the optimization problem. In the original paper, we show that features describing the topology of a point cloud extracted from samples of the optimization problem are informative enough to differentiate between problem classes in the COmparing Continuous Optimizers (COCO) benchmark suite [13]. However, the proposed approach is extremely limited in a number of ways: (i) when it comes to the number of required samples, a few thousand samples are needed to detect any kind of useful topological structures, which can

* Corresponding author at: Computer Systems Department, Jožef Stefan Institute, Ljubljana, 1000, Slovenia.
E-mail address: gasper.petelin@ijs.si (G. Petelin).

https://doi.org/10.1016/j.swevo.2023.101448
Received 26 June 2023; Received in revised form 8 November 2023; Accepted 6 December 2023
Available online 8 December 2023

be prohibitively expensive in practice; (ii) a few hundred samples are needed to be even able to compute the features; (iii) the features that are obtained are high dimensional and they vary with the number of samples meaning that using more samples would most often generate more features; All of these reasons limit the practicality of the originally proposed topology based feature extraction technique.

Our contribution: In this paper, we conduct a comprehensive analysis of the TLA methodology and propose additional improvements on how to use the samples more efficiently to construct a point cloud that allows the extraction of features in scenarios where a lower number of samples from the optimization problem are available. Such reformulation solves the problems with a minimum number of samples and a varying number of features and substantially improves the quality of the extracted topological information. Further, we investigate the impact of the hyperparameters of the proposed method on the behavior of topological features. The newly proposed features are also compared to the existing ELA feature groups with a conclusion that topological features are not strongly correlated with any of the existing feature groups and thus capture different information than existing features. Lastly, we show that topological features are sensitive to sample size meaning that different topological features appear with changing sample sizes.

The rest of the paper is organized as follows. In Section 2 a literature review of the topic is presented. Section 3 introduces the basic concepts of TDA for feature construction. Section 4 gives a detailed description of how TDA can be applied to extract topological features for optimization problem instances, followed by Section 5 describing results. Finally, we describe some of the limitations of the study in Section 6, and in Section 7 we conclude our work.

Reproducibility: The extracted TinyTLA features are available at https://doi.org/10.5281/zenodo.7642287. The code used to extract the TinyTLA features can be found at the Gitlab repository https://repo.ijs.si/gpetelin/tinytla.

2. Related work

This section offers an overview of existing research and methodologies related to understanding the properties of optimization problems. Characterizing the properties of single-objective continuous optimization problems is a well-studied and important part of the field of automated algorithm design for black-box optimization problems. Capturing such properties can aid in getting new insights into the characteristics of the optimization problems, improving algorithm selection [1] and constructing better and more representative benchmarks [5] for fairer algorithm evaluation [14]. Based on the characteristics the features capture, the existing approaches can be broadly categorized as high-level features such as Fitness Landscape Analysis (FLA) [7] and low-level features such as Exploratory Landscape Analysis (ELA) [8].

High-level features capture optimization problem characteristics like global structure, separability, multimodality, variable scaling, and search space homogeneity, which is linked to optimization algorithm performance [15] and aids in algorithm selection. However, obtaining these features is challenging. Their construction requires knowledge from experts who are able to characterize the properties of the optimization problems. In practice, such features are often known only for different artificially created optimization functions included in benchmarks such as COCO [13] and CEC [16].

Describing the optimization problem can also be approached by first sampling candidate solutions from it and using those samples to characterize the function by its lower-level features. Numerous such features exist, with the ELA feature set being the most widely used. In these situations, optimization problems are depicted using a collection of numerical features that can be utilized in subsequent tasks. Note that when we refer to ELA features, we are not talking just about features introduced in [8] but all the feature groups [17,18] that are included in *flacco* [19]. Such features have proved to be extremely effective at differentiating between COCO [13,20] optimization problem classes and algorithm selection [21,22]. Although the ELA features are the most commonly utilized low-level characteristics for describing optimization functions, it is important to acknowledge the existence of other types of features. Examples include Local optima networks [23], Length scale features [24], Adaptive local landscape feature vector [25] and others [26]. Recently, newly emerging approaches where features are learned through gradient descent [27,28] have also become popular. With advances in representation learning in fields such as computer vision [29,30] and natural language processing [31] it comes naturally that such methods are also adapted for the optimization domain. Representation learning methods for feature creation are most often not crafted manually but are automatically learned throughout the process of solving predefined tasks with the goal of such representation being useful for other tasks. One such approach that has been successfully used in optimization is the so-called feature-free approach for representation learning. In [27,32] authors rely on using small initial samples together with deep learning techniques such as convolutional neural networks and neural networks designed for point cloud classification [33] to map the samples to high-level, expert-defined landscape properties. Such approaches offer a completely new way of feature construction that does not require them to be manually constructed. However, such neural-network-based techniques are not without limitations. They come with the cost of decreasing the explainability and require a sufficient amount of data to train. Additionally, some recent approaches such as de Nobel et al. [34] Kostovska et al. [35] and Cenikj et al. [36], extract features from samples visited during the optimization process. However, the methodology discussed in this paper only uses static samples and does not include samples obtained during algorithm execution. The reason behind this is that we initially prioritize exploring the usefulness of the topological landscape features within the problem space, before utilizing them to examine the relations between the problems and algorithms.

In terms of exploring topological features of optimization functions, this is not the first paper on the topic. In topological data analysis, all the samples are treated as a point cloud where different topological shapes may be present. Compared to other feature extraction approaches, the goal is to capture such topological shapes. The original introduction of the layered TLA methodology [12] opened the possibility of using topology to construct features. In the layered TLA feature extraction methodology, samples are first obtained from the optimization function, and are then partitioned into distinct bins depending on the objective function's values. Subsequently, topological data analysis is applied independently to each bin to derive feature, which are then concatenated to create a comprehensive representation of the function. However, the previously proposed layered TLA left a series of open questions that were yet to be explored. The most critical question is if the methodology can be adapted to work with a low number of samples. The original methodology was limited to a minimum of 800 samples which presents a large limitation in cases where that criterion is not fulfilled. Another drawback of the previously proposed methodology is that the number of extracted features was dependent on the number of samples. When increasing the number of samples, the number of topological features would also increase. This property can be highly undesirable in cases where the goal is to compare topological features for different sample sizes. With the number of features being tied to the number of samples, a large sample size produces a lot of features and makes training ML models (for tasks such as algorithm selection or algorithm configuration), much more difficult and time-consuming. Some other open questions involve understanding the impact of different hyperparameters (e.g., sample size, distance metric between samples) on the features' predictive performance, investigating the novelty and the complementarity of the information captured by the layered TLA

features and the existing landscape features, as well as extensibility of the proposed methodology to combinatorial optimization problems.

Despite numerous approaches being proposed, the advancement of the representation models in the last few years opens up new possibilities for describing optimization problems in terms of numerical features, which could provide fresh perspectives on problem characteristics. Furthermore, to the best of our knowledge, there have been scarce efforts to depict the problem landscape through the extraction of topological features that correspond to the presence of different shapes in samples.

3. Background on topological data analysis

The field of Topological Data Analysis (TDA) employs topological techniques for extracting information from often high-dimensional data sets. The main benefits of TDA techniques are that they can be especially robust to noise in the data where other techniques may struggle [37]. Given a point cloud, TDA tries to find the presence of specific topological structures such as spheres, toruses, and connected components that can be used to differentiate between point clouds and are invariant to rotations, translations, and scaling. By identifying such structures, a point cloud can be described in terms of various features, which capture the existence of different structures across different scales, where of special interest are structures that persist across scales, independently of certain transformations (e.g. rotations, scaling). Approaches from TDA have been successfully applied in different areas such as finance [38,39], neuroscience [40], feature extraction from time-series [41,42], image processing [43,44].

Due to TDA being a field that is not generally well-known in optimization, in order to make this work self-contained, the remainder of this section introduces a few basic concepts necessary for a later understanding of the process of obtaining features from samples of optimization problem instances. We purposefully avoid the complex theoretical background and only provide a simplified high-level overview of the TDA concepts related to our proposed methodology of feature extraction. For more detailed rigorous mathematical definitions of TDA, we refer the readers to Chazal and Michel [45] and Skaf and Laubenbacher [46].

3.1. Simplices and simplicial complex

In TDA, Simplices and Simplicial Complexes represent the most fundamental building blocks from which more complex structures are constructed. Simplices (also known as simple pieces) represent a generalization of the notions of lines, triangles, or tetrahedrons to arbitrarily high dimensions. To get a better understanding of simplices, Fig. 1 visualizes the first four simplices: 0-simplex (vertex), 1-simplex (edge), 2-simplex (triangle), and 3-simplex (tetrahedron). When constructing such simplices, it is important to understand that for a higher-dimensional k-simplex to exist, all the (k-1)-simplices also have to be present. This means that for a tetrahedron ABCD to exist, the triangles ABC, ABD, ACD, and BCD have to exist as its sub-components, as well as the six edges AB, AC, BC, AD, BD, CD, and the vertices A, B, C, D. For example, triangle ABC could not exist without all three points A, B, and C being connected together. However, it is important to note that even if edges AB, BC, and AC do exist they do not necessarily have to form a triangle and they can just exist as tree edges forming a simplicial complex.

A simplicial complex is a collection of points, edges, triangles, and higher-dimensional counterparts known as simplices. Fig. 2 shows one realization of a simplicial complex constructed from multiple simplices. The shown simplicial 3-complex also contains pairs of points that are pairwise connected with an edge and form a triangle, as well as pairwise connected points that do not form a triangle. Note that the figure contains an example with an additional triangle that does not form a 2-simplex. The ultimate goal of TDA is therefore to build such simplicial complexes and discover interesting topological properties they might possess.

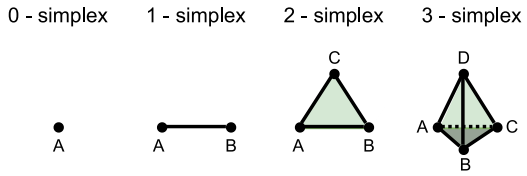


Fig. 1. A set of k-simplices (with k ranging from 0 to 3) that serve as the fundamental components of a simplicial complex.

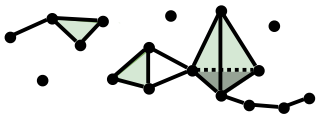


Fig. 2. Visualization of an arbitrary simplicial complex with 17 points (0-simplices), 18 edges (1-simplex), 6 triangles (2-simplex), and one tetrahedron (3-simplex).

3.2. Building simplicial complex

In most practical applications, a simplicial complex is not inherent to the problem but has to be built from a set of points in a point cloud. There are multiple methods for transforming a set of N points into a simplicial complex, two of them being the Vietoris–Rips (also known as Rips complex) complex, and the Čech complex. In terms of computational complexity, the Vietoris–Rips complex is more efficient than the Čech complex [47] and is, therefore, a more commonly used technique. Considering the computational cost, our proposed methodology uses the Vietoris–Rips complex.

When tasked with constructing a Vietoris–Rips complex from a point cloud consisting of N points, the first step is to calculate pairwise distances between all the points and obtain a $N \times N$ symmetric matrix. Although the method used for computing pairwise distances between points has no effect on the methodology used to construct the simplex, the obtained simplex is inherently tied to the distance matrix and selecting a wrong distance measure between points can fail to reveal useful topological structures as explored later. Once the distance matrix is obtained, a Vietoris–Rips complex VR_ϵ consists of all simplices whose vertices have a pairwise distance of less than ϵ. This means that a simplex of dimension k can be formed only if the pairwise distances of all of the $k+1$ points that are part of the simplex are lower than the predefined ϵ threshold. For instance, an edge (1-simplex) can be formed if two points (0-simplices) are closer than ϵ, while a triangle (2-simplex) can be formed if the pairwise distances between three points are less than ϵ. Fig. 3 features an example of a VR_ϵ complex, where each dotted circle is a circle with a radius $\epsilon/2$, and two points are connected if their $\epsilon/2$ circles intersect. The obtained Vietoris–Rips complex shown in the figure also has an interesting topological structure since it encloses one hole, a crucial concept studied with homology.

3.3. Homology

During the construction of a simplicial complex, the appearance of different structures (sphere, torus, connected components, topological circles, trapped volumes) is tracked, with the goal of using that information for constructing features that will describe the point cloud. A homology is an approach from algebraic topology that is able to quantify such structures and formalize the notion of the topological features of a simplicial complex. Essentially, the goal is to count the number of holes that appear in the data. For any dimension k, the k-dimensional holes are represented by a vector space H_k. For example, the 0-dimensional homology group H_0 describes the connected components of a simplicial complex while the 1-dimensional homology group H_1 and 2-dimensional homology group H_2 represent 1-dimensional loops (structured like a doughnut) and 2-dimensional cavities, respectively.

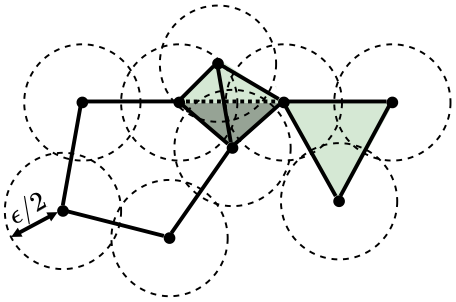


Fig. 3. Example of a Vietoris–Rips complex constructed from 9 points. Points are connected if their $\epsilon/2$-neighborhoods (dotted line) intersect. Shown points form 13 edges (1-simplex), 5 triangles (2-simplex) and one tetrahedron (3-simplex).

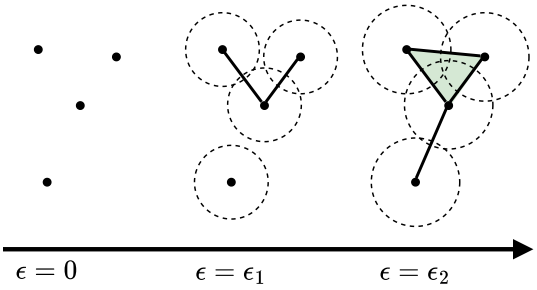


Fig. 4. Set of four points and Vietoris–Rips simplicial complex with different values of ϵ capturing topological features at different scales.

3.4. Persistence homology

The concept of homology addresses only the appearance of the previously described structures for a static simplicial complex. While this can often be extremely useful, we would like to also describe how structures of the simplicial complex persist across scales, since structures that persist over a wide range of spatial scales are deemed more likely to represent more important underlying structures that are not just sampling artifacts. Persistence Homology is an extension of the concept of homology that is designed for computing topological features of space at different spatial resolutions. Fig. 4 shows the process of building a Vietoris–Rips simplicial complex at different values of the ϵ threshold. At small values of ϵ, the points are disconnected and only vertices (0-simplex) are present. When ϵ is gradually increased, new higher dimensional simplices are formed and lower dimensional ones disappear. At sufficiently large ϵ, all the vertices are connected and form a single connected component. Our interest lies in the underlying formation of topological structures i.e. when they are born (all pairwise distances of the topological structure are less than ϵ), and when they die (consumed by higher dimensional simplices).

3.5. Persistence diagram

Persistence diagrams visualize the birth of topological structures and their subsequent death. The diagram consists of the birth axis and death axis which show at what value of the ϵ threshold a certain topological structure is born and when it dies, respectively. Fig. 5 shows the construction of a persistence diagram from a set of points demonstrated by birth and consequent death of a hole. The solid gray line in the figure represents the $x = y$ diagonal line and symbolizes an important property of the diagram. All the points are above this line since topological structures must first be born before they die. Points closer to the diagonal signify short-lived features, often considered noise, while

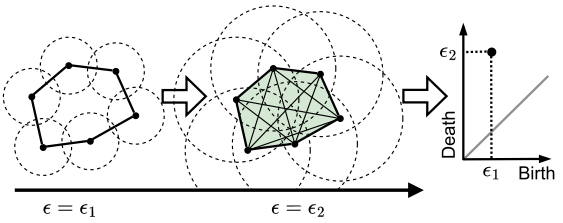


Fig. 5. A process of building a persistence diagram with the appearance of a hole at ϵ_1 and its death at ϵ_2 that is described by a point in a diagram.

those further away represent more robust topological features. Fig. 5 shows the persistence diagram in birth–death coordinates but this is not the only way to visualize the appearance of topological structures. Persistence diagrams in birth–death coordinates can easily be transformed into birth-persistence coordinates by applying a linear transformation $T(x, y) = (x, y-x)$ that showcases when a certain structure was born and how long it was alive. Such transformation is also required and acts as a prepossessing step of the last step of obtaining a persistence image which forms the basis for feature construction. It is important to point out that there are other techniques for comparing persistence diagrams without the need for obtaining a persistence image such as Bottleneck and Wasserstein distances [48,49]. A more comprehensive explanation of the methods for computing distances between persistence diagrams and their properties is available in [50,51].

3.6. Persistence image

Given a persistence diagram in birth-persistence coordinates, the goal is to use it to extract meaningful topological features that can be later used to numerically represent individual optimization problems. One simple approach to extracting features is the binning approach shown in Fig. 6. With this approach, both axes are first discretized to some arbitrary precision with each of the buckets containing a certain number of points. A way to visualize this is by placing a grid over the persistence image and simply counting how many points are present in each grid cell. The level of discretization or the number of bins that are generated is also known as the resolution of the persistence image. Unfortunately, such an approach of generating features is not stable [52] and small perturbations of the point cloud can move the points between the bins of the grid. To mitigate this problem, persistence diagrams are often transformed into persistence images by selecting a probability distribution and positioning it at each point in the persistence diagram to achieve a smoothing effect. While the selection of the distribution used for smoothing is arbitrary, a common practice is to use the Gaussian distribution, centered at each point of the persistence diagram with some predefined variance. This approach is shown in Fig. 6. Such a technique spreads the mass of a single point over a larger area, ensuring that the generated persistence images are less susceptible to noisy observations in the point cloud. The persistence image can then be transformed into features describing the original point cloud by concatenating all of the pixels from the image into a single vector.

3.7. Practical example

To get a better understanding of the feature extraction and how the previously described concepts are related, Fig. 7 shows an example of four point clouds with different topological properties all containing exactly 500 points. The first two images show a sphere with and without noise. We can observe that the obtained persistence diagrams are relatively similar. In both diagrams, we get one birth–death point that is far from the diagonal and represents the hole in the middle of

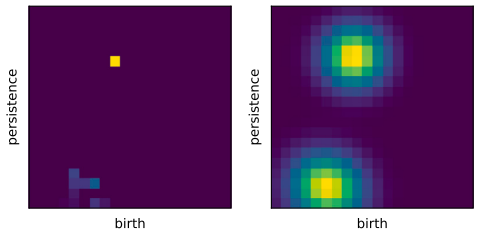


Fig. 6. A persistence image obtained from persistence diagram with an approach of binning with no smoothing (left) with each point from the persistence diagram belonging to exactly one bin and a persistence image obtained with an approach with smoothing (right) where mass is distributed over multiple bins.

the sphere. For point clouds on the right in the shape of an infinity sign, a similar observation can be made. The point cloud essentially contains two holes, therefore, two corresponding (overlapping) points that are far away from the diagonal can be observed on the persistence diagram of the infinity sign. Similarly, when the infinity sign point cloud is scaled down and rotated, the persistence diagram gets scaled by the same factor. This shows the advantageous properties of TDA concepts, including rotation/translation invariance, insensitivity to noise, and scaling. In the last step of the visualization (bottom row), persistence images generated from the persistence diagram homology H_1 are also shown.

4. Methodology

Here, we give a detailed overview of the proposed methodology and the application of the landscape feature extraction method with a low number of samples. The approach of extracting features can be divided into the following steps and is shown in Fig. 8: (i) generating a set of N candidate solutions $\vec{x_1}, \vec{x_2}, \dots \vec{x_N}$ and evaluating them with a given objective function $f(\vec{x_i})$ to obtain their objective function values $f(\vec{x_1}), f(\vec{x_2}), \dots f(\vec{x_N})$; (ii) rescaling of the landscape by applying sample specific transformations on $\vec{x_1}, \vec{x_2}, \dots \vec{x_N}$; (iii) calculation of sample-specific pairwise distances between all $\vec{x_1}, \vec{x_2}, \dots \vec{x_N}$ and objective value specific pairwise distances between $f(\vec{x_1}), f(\vec{x_2}), \dots f(\vec{x_N})$ and subsequently combining them into one distance measure; (iv) transforming the pairwise distances into a persistence diagram and subsequently transforming the persistence diagram into a persistence image used for numerical features. The algorithm presented in Pseudocode 1 outlines each step, and further information about each step can be found in the remaining section.

The rest of this section is organized into individual subsections that give additional details about each step of the procedure. Note that this methodology is an improvement of a previously proposed TLA approach [12]. The novel approach introduced here is how to apply TLA in scenarios where a low number of samples are available.

4.1. Obtaining samples from the objective function

When given the optimization problem, the first step is to generate a set of candidate solutions $\vec{x_1}, \vec{x_2}, \dots \vec{x_N}$ and evaluate them using the objective function $f(\vec{x_i})$. Note that the dimension of vector $\vec{x_1}$ is equal to the problem dimension. The obtained candidate solutions serve as a basis for further feature creation of the TLA methodology. There exists extensive literature [9] on how to perform sampling and what strategy to use when computing features for ELA. In general, our proposed methodology does not dictate what sampling strategy to use. Therefore, any of the existing sampling strategies can be used. Moreover, topological features are generally not extremely sensitive to noise [53] (i.e. perturbations of cloud points have small effects on persistence image). We leave the sensitivity of topological features to sampling techniques and perturbations of samples as future work.

Algorithm 1: Algorithm pseudocode for topological feature extraction.

$\alpha \leftarrow 0.2$;
$X \leftarrow createInitialSample()$;
$y \leftarrow evaluate(X)$;
$X \leftarrow applyTransform(X, \text{“}volume\text{”})$;
$X_d \leftarrow samplePairwiseDistances(X, \text{“}euclidean\text{”})$;
$Y_d \leftarrow samplePairwiseDistances(y, \text{“}euclidean\text{”})$;
$X_d^n \leftarrow X_d^n / max(X_d)$;
$Y_d^n \leftarrow Y_d^n / max(Y_d)$;
$M_d \leftarrow \alpha X_d^n + (1-\alpha) Y_d^n$;
$Pd \leftarrow persistenceDiagram(M_d)$;
$Pi \leftarrow persistanceImage(Pd, H_1, 0.002)$;
$features \leftarrow flatten(Pi)$;

4.2. Applying transformations

This subsection explains transformation techniques for obtained samples. These transformations were previously introduced in [12] to simplify feature comparison, although they are not strictly necessary. Normalizations are applied to aid in constructing better features and simplify the pipeline. This step makes persistence diagrams in later steps easier to compare. Note that transformations here are only applied to the candidate solution $\{\vec{x_1}, \vec{x_2}, \dots, \vec{x_n}\}$ and not objective function values. We use the following two transformations:

(i) The ***Axis*** transformation is applied to a set of candidate solutions in two steps. The first step is to apply Principal Component Analysis (PCA) on the set of all candidate solutions $\vec{x_1}, \vec{x_2}, \dots \vec{x_N}$ which centers the set of points at the origin while preserving the distances between points. Note that in this case PCA is not used as a dimensionality reduction technique but just as a step to align/center the point cloud near the origin. Once points are shifted toward the origin with the help of PCA, the next step is then to transform the points so they are always between -1 and 1. This is performed by simply linearly scaling each axis separately so the maximum value is equal to 1 while the minimum value equals -1. In the newly transformed point cloud, all the points are thus scaled to be between -1 and 1 as shown in the second picture of Fig. 9.

(ii) Similarly to the previous transformation, the ***Volume*** transformation also first applies PCA on the candidate solutions thus moving the point cloud toward the coordinate system origin. The difference between this method and the *Axis* transformation is the scaling of the transformed candidate solutions. With the *Volume* transformation, after applying PCA, an axis-aligned bounding box [54] is fitted around the centered candidate solutions. Points are then scaled in such a way that the bounding box surface (2D problems), volume (3D problems), or hypervolume (>3D problems) is equal to 1. In other words, all dimensions of candidate solutions are scaled by some constant factor in such a way that the enclosing axis-aligned bounding box has a certain surface, volume, or hypervolume depending on the dimension.

To give a better overview of the transformations and the reasoning behind them, Fig. 9 shows an example of an ellipse point cloud with the previously described transformations. Due to the normalization of the axes using the *Axis* transformation, the ellipse point cloud is treated as a stretched circle and thus transformed into a point cloud that is equivalent to a circle. Such a transformation will try to counteract the scaling and will thus map stretched objects in such a way that their topological properties are similar. The *Volume* transformation, on the other hand, will still try to scale the point cloud with the important property of not stretching the point cloud.

Which of the later two transformations should be used comes down to what properties need to be preserved. If we look at the ellipse as being equivalent to the circle, then the *Axis* transformation should be used. On the other hand, if differentiating between stretched point clouds is important, the *Volume* transformation should be used.

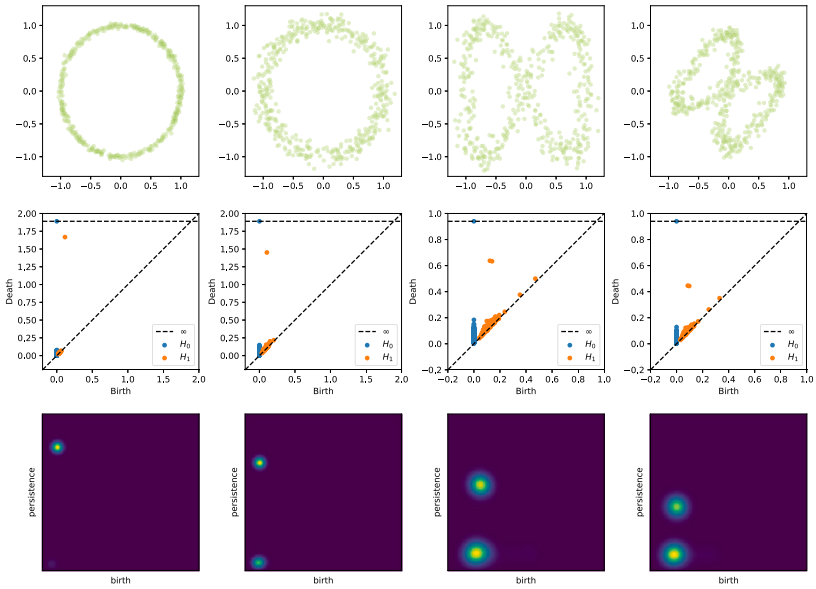


Fig. 7. Examples of different point clouds (top) with 500 points and their persistence diagrams (middle) and persistence images (bottom). The diagram in the picture shows homologies for dimension 0 (blue) as well as dimension 1 (orange). Persistence images only show dimension 1 homology (H_1).

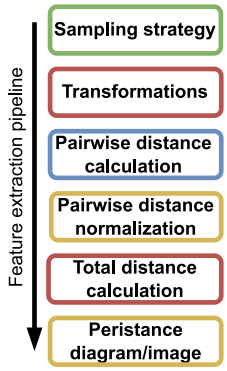


Fig. 8. A step-by-step approach for deriving topological characteristics, starting with generating samples, followed by computing distances between them to identify topological structures.

4.3. Pairwise distance calculation

The construction of a persistence diagram is conditioned on having a metric that can calculate distances between all candidate solutions and their objective function values. Therefore, selecting a meaningful distance metric is crucial for constructing a persistence diagram that can be later transformed into meaningful numerical features. The original TLA paper proposes a binning approach where samples are grouped into bins based on the objective function value of individual samples. Distances between samples in individual bins are subsequently calculated using Euclidean distance. This implicitly introduces some information about the fitness values into the persistence diagram generation. Unfortunately, this also introduces a need to select a trade-off between finer discretization of fitness values (by selecting more bins) and stability of the generated persistence diagram due to a low number of samples in each layer. In our case, we define the distance between samples in the following way. Let $p_i = (\vec{x}_i, f(\vec{x}_i))$ and $p_j = (\vec{x}_j, f(\vec{x}_j))$ be the two samples sampled at $\vec{x}_i$ and $\vec{x}_j$ with objective function values of $f(\vec{x}_i)$ and $f(\vec{x}_j)$, respectively. We define two symmetric matrices representing the distances between candidate solutions and the distance between their objective function values. The matrix X_d represents pairwise distances between all candidate solutions and is defined as $X_d(i,j) = dist(\vec{x}_i, \vec{x}_j)$, where $dist(\vec{x}_i, \vec{x}_j)$ represents some arbitrary metric (for example Euclidian distance) for calculating the distance between candidate solutions. Note that $\vec{x}_i$ and $\vec{x}_j$ here represent a numerical vector. Furthermore one has to also calculate the distance between objective values of given candidate solutions. The distance matrix Y_d between objective function values is thus defined as $Y_d(i,j) = dist(f(\vec{x}_i), f(\vec{x}_j))$. For the later case of calculating distances between objective function values, $f(\vec{x}_i)$ and $f(\vec{x}_j)$ are no longer numerical vectors but are just scalar values since we are working with single-objective optimization. Both matrices X_d and Y_d thus capture a piece of really important information about the optimization problem. The matrix X_d represents the distance between candidate solutions while Y_d contains information about differences in their objective values.

4.4. Pairwise distance scaling

Distances obtained in the previous section contain unnormalized values meaning that values inside matrices have no upper bound. For that purpose, comparing optimization functions that have been transformed cannot be achieved. To solve this, both matrices are further scaled in such a way that minimum values are equal to zero and the maximum value is equal to one. This is achieved by simply dividing all the values in both matrices by the maximum value described by the following equations and obtaining new scaled matrices $X_d^n(i,j)$ and $Y_d^n(i,j)$:

$$X_d^n(i,j) = \frac{X_d(i,j)}{\max\limits_{1<i,j<N}(X_d(i,j))} \tag{1}$$

$$Y_d^n(i,j) = \frac{Y_d(i,j)}{\max\limits_{1<i,j<N}(Y_d(i,j))} \tag{2}$$

While this step of scaling distances is not strictly necessary and features can be obtained without it, it helps construct features that are invariant to transformations such as scaling.

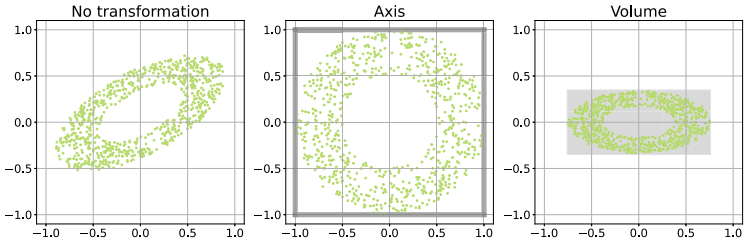


Fig. 9. Transformations that can be applied to the set of all candidate solutions. With *Axis* transform the new point cloud is inside $[-1,1]^D$ while for *Volume* transform, the volume (hypervolume) of the box is equal to 1.

4.5. Quantifying the distances between samples

When extracting topological features, one has to define a single-distance matrix used to obtain a persistence diagram. Therefore the two matrixes X_d^n and Y_d^n have to be combined together into a single one representing the total distance between candidate solutions and their function values. This is achieved by taking a linear combination between both distance matrices as defined by the equation:

$$M_d(i,j) = \alpha X_d^n(i,j) + (1-\alpha) Y_d^n(i,j) \tag{3}$$

Total distances between samples thus comprise the distance between candidate solutions as well as differences in their objective function values. The parameter α is used to control the fusion of information about the distance between the samples and the distance between the function values. Note that distance calculation is performed after normalization. Such an approach of fusing the distances together controlled with the hyperparameter α allows finer control over combining information about the location of where the sample was obtained and its function value. Without combining information from both matrices it is impossible to build one persistence diagram that captures the distance between samples and the differences between the objective function values. This technique of combining both distance matrices together into one matrix enables the use of much smaller sample sizes and captures more informative topological features. This step of combining both matrices together is also the main advantage over the initially proposed [12] method. In the original paper, samples were split into multiple bins depending on their objective value. For example, if two bins were used, half of the samples with the highest objective value would be put into one bin, while the rest (samples with the lowest values) would be assigned to another bin. The matrix $X_d^n(i,j)$ would be calculated only between pairs of samples that are in the same bin while $Y_d^n(i,j)$ would not be explicitly computed. This means that topological features would be obtained for each bin separately without taking into account objective function values with other samples in the same bin. The main limitation of binding was therefore throwing away a lot of information about objective values of samples.

4.6. Persistence homology and diagrams

With the obtained distance matrix which captures distances between sample points and their function values, persistence homologies and diagrams can be computed as described in Sections 3.4 and 3.5. The persistence diagram in this case holds the information about different topological structures that appear in the point cloud described with a given pair-wise distance matrix. A persistence diagram can be generated for homologies of different dimensions. In our case, we limit ourselves to homologies H_0, H_1, and H_2 due to the associated computational cost.

4.7. Numerical feature construction

With a given persistence diagram that represents the births and deaths of topological structures, the next step is to construct finite-dimensional numerical features from it. The process of transforming the persistence diagram into a persistence image is described in Section 3.6. The obtained features are most often described by a high-dimensional vector that represents the creation and lifetime of topological properties of a problem. One important thing to note here is that feature vectors obtained using different transformations and homology dimensions can be concatenated together. For example, with two transformations and three persistence diagrams obtained with homologies H_0, H_1, and H_2 one can construct six feature representations for a certain optimization problem. Those feature representations can be further concatenated together to obtain different topological views of the problem before being used in machine learning applications.

4.8. Example of extracted topological features

To better understand of what the extracted features look like, Fig. 10 shows two different optimization problems together with their persistence diagrams and persistence images (only for homology H_1), constructed with a sample size of 600. We see that the computed persistence diagrams and persistence images of the function samples are slightly different, meaning that the functions differ in their topological features. Note however that for many real-world point clouds, differences in persistence diagrams are not as easily identifiable [55] as in the toy example shown in Fig. 7.

5. Results

In this section, we first describe the experimental setup and the hyperparameters used to conduct our experiments. In the second part of this section, we explore the similarity of the proposed features for different problem classes of the COCO benchmark and analyze the similarity of the proposed TLA features to existing ELA features together with the influence of hyperparameters.

5.1. Experimental setup

Topological feature extraction was performed with the *Scikit-TDA* [56] library while ELA features were calculated with the *flacco* [19] library. For classification, we used the *AutoGluon* [57] framework. Lastly, we use the *scikit-learn* [58] library for different utilities for performing classification.

To test the proposed feature extraction technique, we use the COCO benchmark suite [13] which is one of the most commonly studied benchmark suites for optimization algorithms. The COCO suite contains problem instances from 24 different problem classes, where each problem instance represents an instance of the base problem class with some applied transformation such as scaling, shifting, or rotation. We use 100 problem instances from 24 problem classes which in total means 2400 problem instances. In our study, we rely on the most

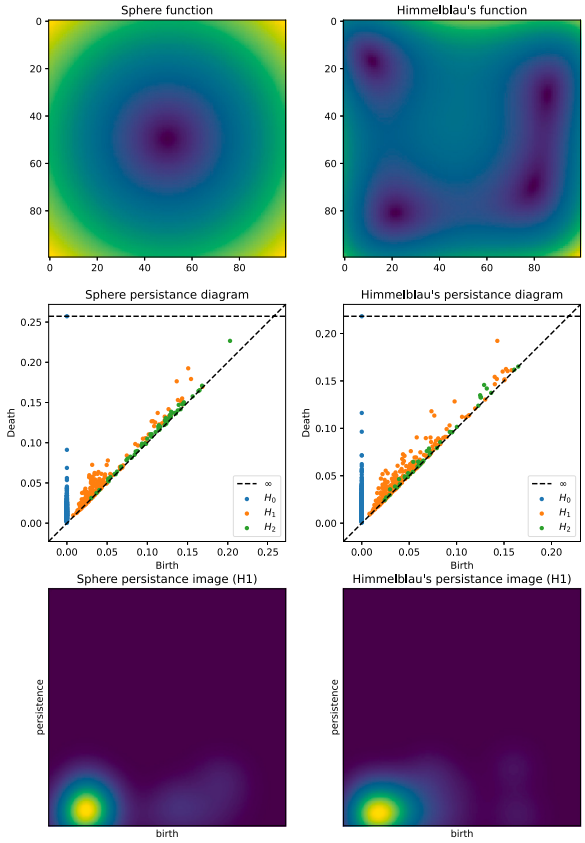


Fig. 10. Sphere function and Himmelblau's function together with their persistence diagrams and persistence images for homology H_1 (orange points).

fundamental sampling strategy where samples are generated uniformly. Motivated by the definition of the COCO problem instances, for a D dimensional problem, the sampling range is limited to $[-5, 5]^D$. The proposed methodology uses the following hyperparameters unless otherwise specified. Parameter α used to merge distance between samples and their objective function values is set to 0.2, the kernel size used for smoothing is set to 0.0002, and Euclidean distance is used to compute distances between candidate solutions as well as their objective function values. However one has to note that for higher-dimensional problems, Euclidean distance often becomes less informative [59].

Due to the computational complexity of obtaining persistence diagrams for higher-dimensional homologies, we limited ourselves to homologies H_0, H_1 and H_2. When transforming a persistence diagram to a persistence image for a selected homology, one has to select the appropriate number of bins for discretization. In our case, for H_0 where the birth occurs only at $\epsilon = 0$, the birth range was set to be between 0 and 0.01, while persistence was set to be between 0 and 1.0 with a discretization of 0.01. With such parameter values, the persistence diagram for H_0 generates 100 numerical features. For homologies H_1 and H_2, the discretization step was 0.01 with the birth and death axes both being between 0 and 1.0. This means that numerical representations for H_1 and H_2 are vectors of length 10,000. All three homologies' birth and death ranges are limited to 1.0 since the largest distance between points is at most 1.0 due to the applied normalization step, as described in Section 4.4. Since we are working also with different transformations (*Axis* and *Volume*), the final objective function representation is obtained by concatenating all the six representations together (all combinations of homology and transformation) to get a final numerical vector of length 40,200. In some cases PCA is used to reduce the number of features (from 40,200 initial features) in such a way that 99.5% of the variance is preserved, making it easier for different machine-learning tasks. Note also that features are not normalized before PCA is performed since it was experimentally observed that pre-PCA normalization produces a much worse representation. Unless otherwise specified, all of the features are extracted from 200 samples regardless of the dimensionality of the problem. While such sampling is uncommon, 200 samples present a trade-off between the computational cost of obtaining topological features and accuracy. A more common strategy is to extract features where samples are not constant for every dimension but are proportional to the size of the problem dimensionality such as in [60,61]. It is therefore important to note that when figures show feature performance, they show that in terms of the number of samples and not the number of samples multiplied by the problem dimension.

5.2. Problem instance similarity and comparison with ELA

In this subsection, we first analyze the extracted features and use them to visualize similarities between instances. We further compare the correlations between proposed features and different ELA feature groups to examine if features are complementary to each other. Note that in this section we use 2D problems for all the visualizations. Figs. 11–13 visualize the cosine similarity between instances from 24 COCO benchmark problem classes. These examples use the first 100 instances of dimension 2 with 200 samples (i.e., 100D sampling). Be aware that the figures utilize varying ranges.

Fig. 11 shows the cosine similarity heatmap for the homology H_0 where samples are previously transformed with the *Axis* transformation. We can observe that for some problem instances it is easy to identify the problem class they belong to, while for others, differentiation to which problem class they belong is much more challenging. For example, focusing on instances from problem 24, we can observe that instances from this class can be identified since correlations between them are strong. On the other hand, instances from problem classes 22 and 23 are highly correlated not only with instances within the same class but also between the two problem classes themselves. We would therefore expect that based on the extracted features, differentiating between classes 22 and 23 would be more challenging when using only homology H_0.

If the focus is shifted toward Fig. 12 where the same transformation is used but now with homology H_1, we can observe that some classes are easy to identify since instances belonging to them are highly correlated between themselves. An example of this would be the fifth problem where all its instances are highly correlated between themselves. Lastly, Fig. 13 also shows the homology H_2. From this figure, we can observe that problem instances belonging to problem 19 can be easily identified which was not the case for homologies H_0 and H_1. This indicates that different homologies reveal different topological information and that the homology of specific dimensions can capture information that is unique to a dimension. Moreover, this demonstrates that problem instances belonging to the same problem class usually have similar topological features. An almost identical pattern of correlations can be observed when applying the *Volume* transformation but we do not show it due to being extremely similar to the *Axis* transformation.

To further explore the behavior of the proposed TLA features, we compare them to different ELA feature groups. We use Pearson linear correlation to quantify the similarity between all feature pairs. Fig. 14 shows the correlation between all features calculated using 2400 instances from COCO. Note that for the purposes of easier visualization, topological features were reduced with PCA to only 10 features. In general, the process of generating topological features will produce a large number of highly correlated features. From the figure, we can observe that some feature groups contain highly correlated features

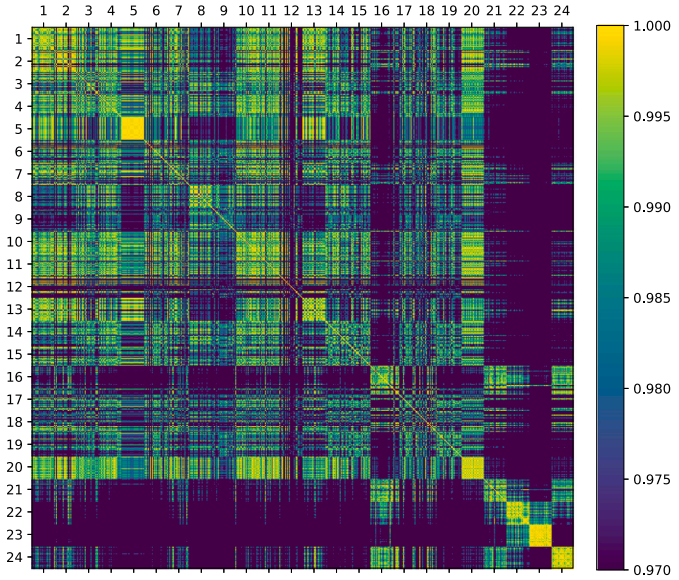


Fig. 11. Cosine similarities between TLA features of COCO instances with *Axis* transformation and homology H_0 for all 24 problem classes and 100 problem instances per class. Note the different ranges compared to Figs. 12 and 13.

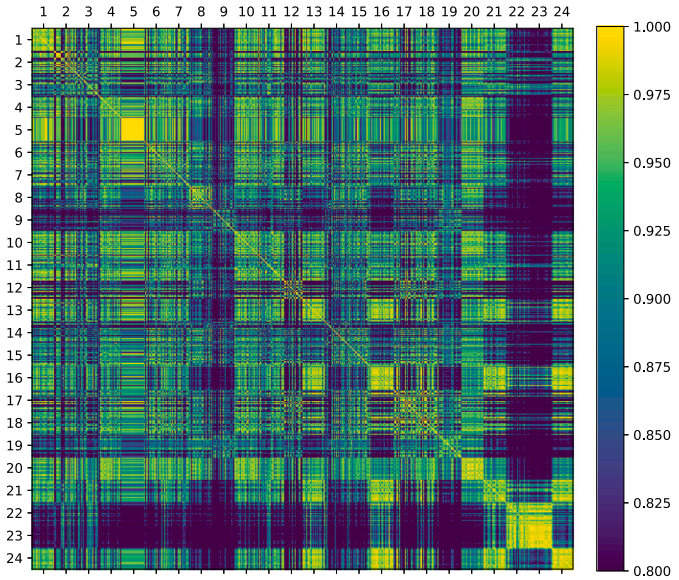


Fig. 12. Cosine similarities between TLA features of COCO instances with *Axis* transformation and homology H_1 for all 24 problem classes and 100 problem instances per class. Note the different ranges compared to Figs. 12 and 13.

while others do not. For example, the feature group *DISP* which captures dispersion properties contains a lot of highly correlated features while features in the group *Basic* are not that strongly correlated between themselves. More interesting is the observation of the correlation between groups. While some feature groups are linearly correlated between themselves (for example *DISP* and *ELA Level*), none of the feature groups is strongly correlated with the topological features. This indicates that the extracted topological features cover a different part of the feature space compared to the ELA feature groups. This is an important finding because diversifying the feature space can lead to a more comprehensive understanding of functions. A failure to account for relevant features can lead to misleading interpretations, making it critical to explore the feature space as fully as possible.

5.3. Problem classification

When using the extracted features to perform classification we use the following methodology. The full data set consists of 24 COCO problem classes with each problem class having 100 problem instances. In total, we work with 2400 problem instances from which features are extracted. The data is split in such a way that 90% of the data is used for training while the remaining 10% is used for testing. When performing random splits, a stratified split strategy is used where the

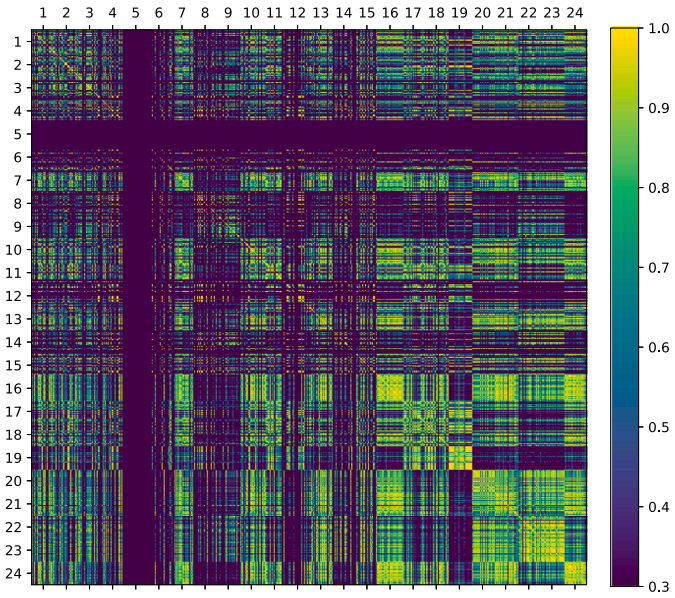


Fig. 13. Cosine similarities between TLA features of COCO instances with *Axis* transformation and homology H_2 for all 24 problem classes and 100 problem instances per class. Note the different ranges compared to Figs. 12 and 13.

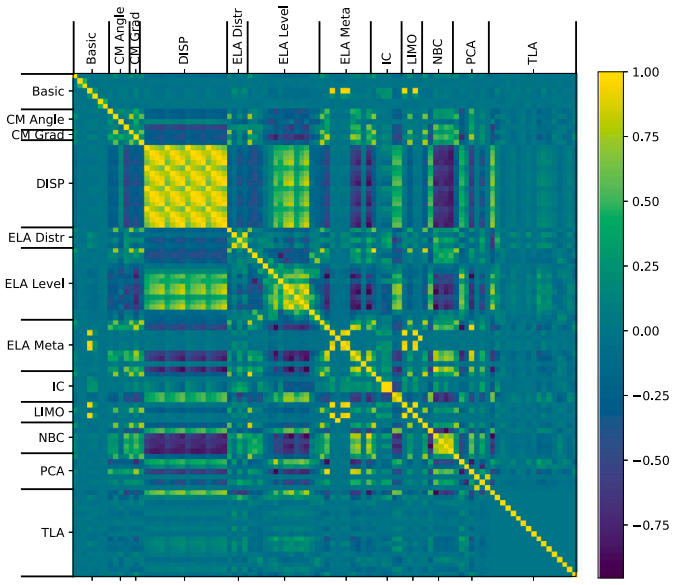


Fig. 14. Correlation between individual ELA and TLA features computed on 2400 COCO problem instances from 24 different problem classes.

balance between classes is preserved after the split meaning that in each split the probability of observing an instance belonging to a certain problem class is preserved. The procedure is repeated 30 times to get a better overview of the performance and robustness of the models. When performing classification it is always challenging to select the most appropriate classification algorithm [62]. In our study, we used the AutoML [63] library AutoGluon. AutoGluon is an efficient library that automatically performs algorithm selection, hyperparameter tuning, and automated stack ensembling and can most often outperform other individual models such as neural networks and random forest when using tabular data [64]. In most of the AutoML frameworks, the only requirement is to specify the time budget for training models. In our case, we used a relatively small time budget of one minute.

To get a better overview of how informative the newly proposed TLA-based features are, we compare them with ELA feature groups for classification. In this case, the classifier is trained separately for each feature group. Fig. 15 shows the classification accuracy for different ELA feature groups and the TLA features on 2D problems. Additionally, ELA ALL denotes the accuracy when all ELA feature groups are combined. Since we have 24 problem classes, classifying instances randomly would produce a classification accuracy of 1/24. Additionally, the number in parenthesis shows how many individual features are in each feature group. We see that all of the feature groups outperform

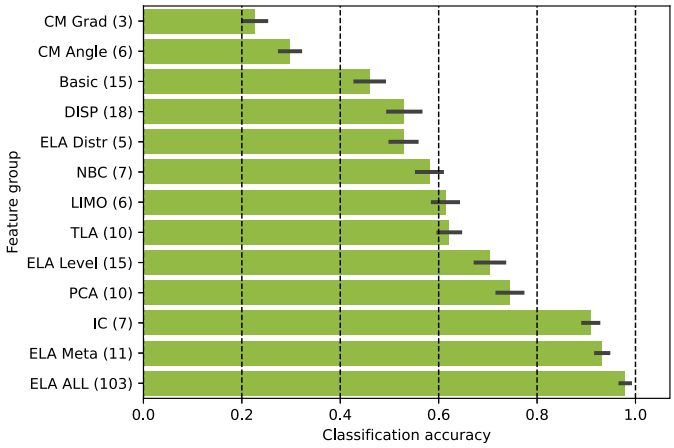


Fig. 15. Classification accuracy for individual ELA feature types together with features extracted with topological data analysis for 2D problems. Gray bars represent the standard deviation in error for 30 independent runs. The number in the parenthesis show how many individual features are in each feature group.

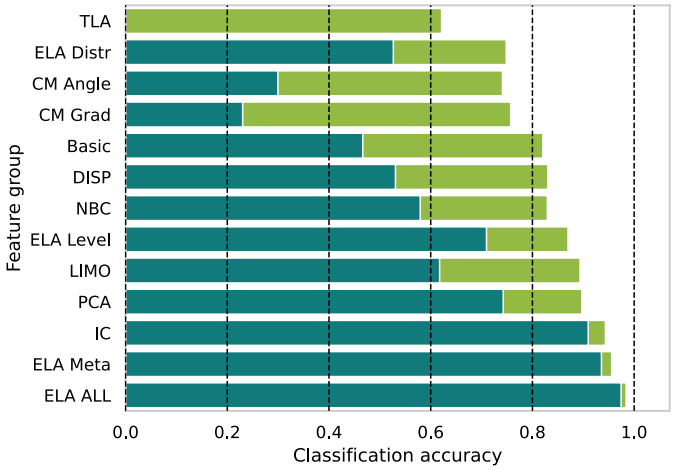


Fig. 16. Classification accuracy for individual ELA feature groups concatenated with TLA features obtained on 2D problems.

the baseline naive random class classifier. Additionally, we see that some ELA feature groups are much more informative than others for the task of problem classification. The least informative feature groups, in this case, are feature groups *CM Grad*, *CM Angle* and *Basic*, while the most informative feature groups are *IC* and *ELA Meta*, the last of which achieve almost 90% classification accuracy. TLA features are the fifth most informative feature group when it comes to the task of classifying samples into one from the 24 COCO classes with an accuracy of around 63%, making it comparable to *LIMO* feature group in terms of classification accuracy. This demonstrates that, even though it might be visually challenging to distinguish problem instances from one another when represented with topological features as discussed in the previous section, machine learning models are quite adept at identifying the specific topological patterns for each of the problem classes.

To further explore the information captured by the TLA features in comparison to existing features, we also show how complementary they are with other ELA feature groups. Fig. 16 shows the classification accuracy of individual ELA feature groups and classification accuracy when the same group is concatenated together with the TLA features. This shows how much additional information is contained in TLA features when compared to other ELA feature groups. TLA with no additional ELA features can correctly classify approximately 63% of the instances. Concatenating topological representation with other features can further improve that number. Concatenating TLA with less informative feature groups such as *CM Grad*, *CM Angle* or *Basic* can increase the classification accuracy to around 75%, meaning that their representation complements each other. Similarly, combining TLA features with feature groups such as *IC* and *ELA Meta* can further increase their accuracy to above 90%. If TLA features are concatenated with all the ELA feature groups, the further improvement in classification accuracy is only around 1%. This shows that concatenating TLA with other individual groups of features is beneficial since they carry complementary information improving classification accuracy.

It is also interesting to investigate which problem classes are most commonly misclassified. Figs. 17 and 18 show confusion matrices for ELA and TLA feature groups, respectively. Note that figures are aggregated over all 30 runs to get more robust results. Focusing first on the confusion matrix for ELA features, we observe that most of the problem instances are classified correctly. The exception is instances belonging to problems 10 — Ellipsoidal Function, 11 — Discus Function and 12 — Bent Cigar Function. With TLA confusion matrix shows that more problem instances are misclassified. Problem classes whose instances are most often misclassified are classes 6 — Attractive Sector Function, 10 — Ellipsoidal Function, 11 — Discus Function, 15 — Rastrigin Function, 17 — Schaffers F7 Function, and 18 — Schaffers F7 Function, moderately ill-conditioned. This means that TLA features alone have

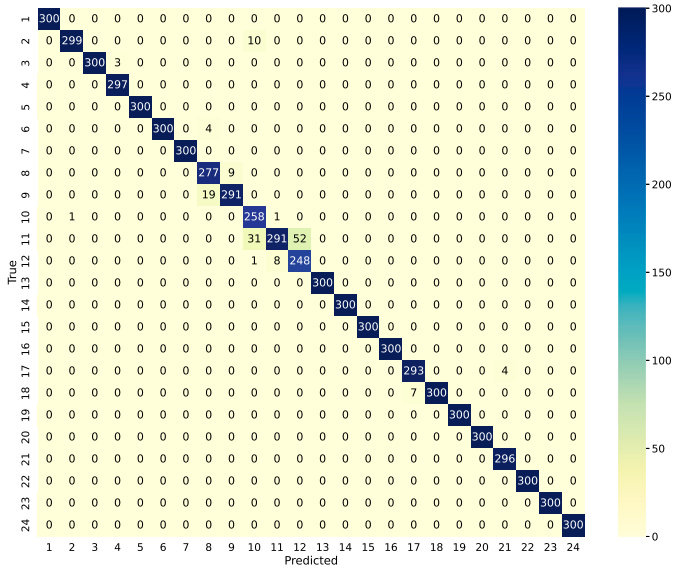


Fig. 17. Confusion matrix when classifying 2D problem instances into one of the 24 problem classes using ELA features. Values are aggregated over all 30 train/test splits.

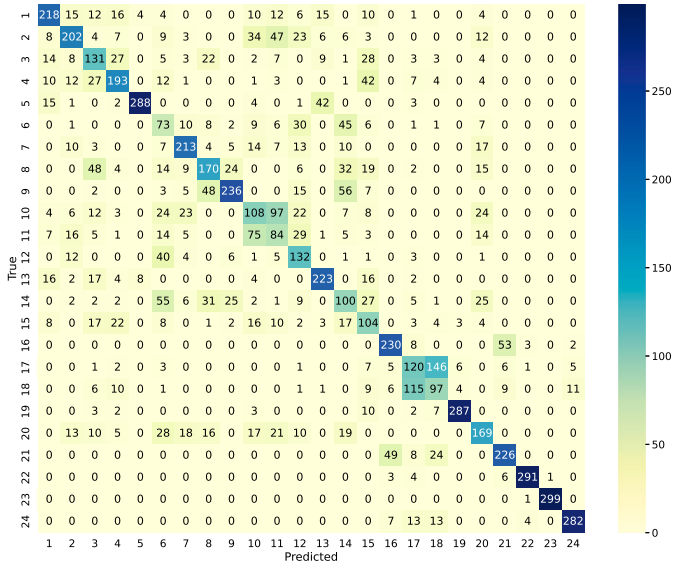


Fig. 18. Confusion matrix when classifying 2D problem instances into one of the 24 problem classes using TLA features. Values are aggregated over all 30 train/test splits.

less power to distinguish between problem instances compared to the ELA features that consist of many different feature groups. One possible explanation of why some problems cannot be differentiated using topological features is that the extracted features are invariant to some transformations meaning that a function that is rotated, translated, or changed in some other way may produce similar topological features. In this regard, they can never achieve accuracy that is comparable to ELA features on the given benchmark since they are unable to differentiate between some transformations. For example, problems 17 and 18 cannot be differentiated with topological features. Both of those problems are Schaffers F7 Function with one being moderately ill-conditioned.

5.4. Hyperparameter sensitivity analysis

In this section, we delve deeper into the impact of certain hyperparameters related to feature extraction. We investigate their sensitivity and examine how they affect the differentiation of problem instances.

5.4.1. Sensitivity to the parameter α

We further investigate the influence of the α parameter on the classification performance. Note that the features used in this example are obtained by first concatenating features from homologies H_0, H_1, and H_2, and both transformations. Since feature representations obtained in this way have very high dimensionality, PCA is applied as a dimensionality reduction technique, preserving 0.995 of the variance.

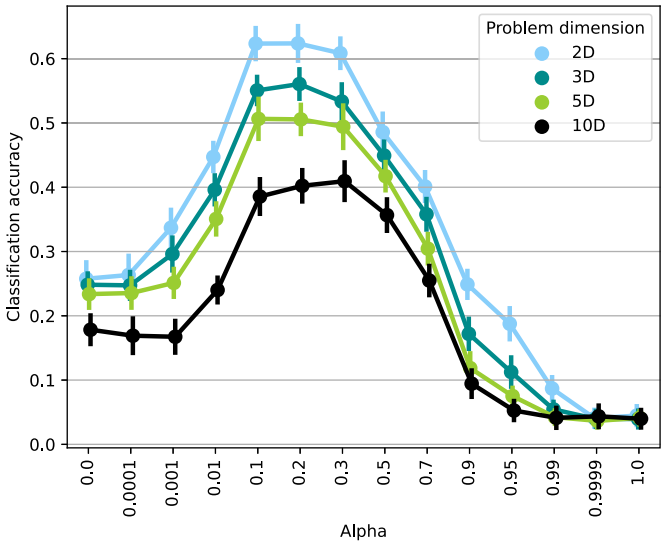


Fig. 19. Influence of α parameter on classification accuracy.

Fig. 19 shows the influence of the α parameter on the classification accuracy for problems with different dimensions. Note that the axis showing α parameter values are not equally spaced. When α is set to a number 0, meaning that only objective function values $f(\vec{x}_i)$ are used to obtain features we see that classification accuracy of around 25%–35% can be obtained. Increasing the α parameter introduces into the features also information about the location $\vec{x}_i$ of where the samples were obtained, not only the objective value of the samples $f(\vec{x}_i)$, and we can see that this increases the classification accuracy for all four dimensions. With setting α to around 0.1–0.3 the maximum accuracy of around 64% is achieved for 2D problems, with accuracy being around 40% for higher-dimensional 10D problems. With further increases of the α parameter, the accuracy begins to decline again. With α set to 1.0 or close to it, classification accuracy drops to the majority classifier accuracy for all dimensional problems. This demonstrates that selecting the α parameter which balances trade-offs between preserving the positional information $\vec{x}_i$ between samples and information between their objective function values $f(\vec{x}_i)$, is crucial for obtaining meaningful topological features. An intuitive explanation for this is that knowing only objective function values $f(\vec{x}_i)$ (when α equals 0) does carry some information about the function while including just sample locations $\vec{x}_i$ (when α equals 1) carries no information about the problem. The most important topological structures are revealed when both distance matrices $X_d^n(i,j)$ and $Y_d^n(i,j)$ are combined.

5.4.2. Sensitivity to the image smoothing

Here, we investigate the influence of persistence image smoothing on classification performance. Fig. 20 shows the influence of selecting the different smoothing parameters on the classification accuracy. Note that the axis showing smoothing parameter values is not equally spaced. When binning is performed without smoothing (i.e. kernel size is 0.0), we observe a classification accuracy of between 45% for 2D problems and 30% for 10D problems. Introducing smoothing into the persistence image improves the classification accuracy significantly. With the smoothing level set to between 0.0001 and 0.001 the highest accuracy can be achieved for all the problem dimensions. If the smoothing parameter is set to too large of a value, we enter a regime where persistence images are over-smoothed and information about topological features is getting lost, which can be observed in degraded classification performance. At around 0.05, the classification accuracy drops to the same level as when no smoothing is applied. This implies that some smoothing is beneficial since it can overcome some problems related to binning, but smoothing persistent images too much can hide important topological information.

5.4.3. Sensitivity to the homology dimension

This subsection explores the homology dimension and its contributions toward classification accuracy. Note that due to the computational cost, we only include homologies up to the dimension of two. Fig. 21 shows the classification accuracy for different problem dimensions with homologies H_0, H_1, H_2 and a case where the representations obtained with all three homologies are concatenated. An observation can be made that homology H_0, which describes how connected components are formed, usually carries the most information for all problem dimensions with the exception of 10D problems where H_1 homology is the most important one. The least informative homology in all of the cases is the homology H_2 that never exceeds the accuracy of 35% for any of the problem dimensions. The best classification accuracy is achieved when concatenating features of all three homologies together. With this, we capture topological holes of different dimensions thus achieving the highest classification accuracy.

5.4.4. Sensitivity to the distance metric

Here we investigate distance metrics for quantifying the distances between samples (see Section 4.5) in a point cloud and the impact of these metrics on classification accuracy. Fig. 22 shows the classification accuracy for different distance metrics. Major differences between metrics can be observed. For example, cosine similarity, the worst-performing metric, achieves results that are no better than the majority classifier. While we do not investigate why this happens with cosine similarity, a reason for this could be that points are far apart but have a similar angle relative to the origin which does not reveal useful topological features. All other metrics achieve much higher classification accuracy, with Canberra, Chebyshev, Euclidian, and Cityblock metrics having similar performance with accuracy well above the majority classifier, with an accuracy of 60% and 40% for 2D and 10D problems respectively. Lastly, we see that the metrics that perform well for low-dimensional problems also have good performance for higher-dimensional problems. Similarly, metrics that perform badly on lower-dimensional problems have also poor performance on higher-dimensional problems.

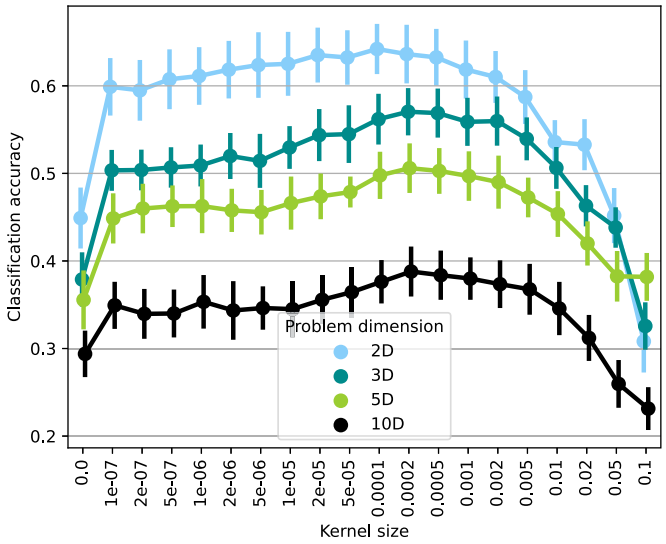


Fig. 20. The influence of the smoothing parameter that controls how individual points from the persistence diagram are smoothed when constructing a persistence image.

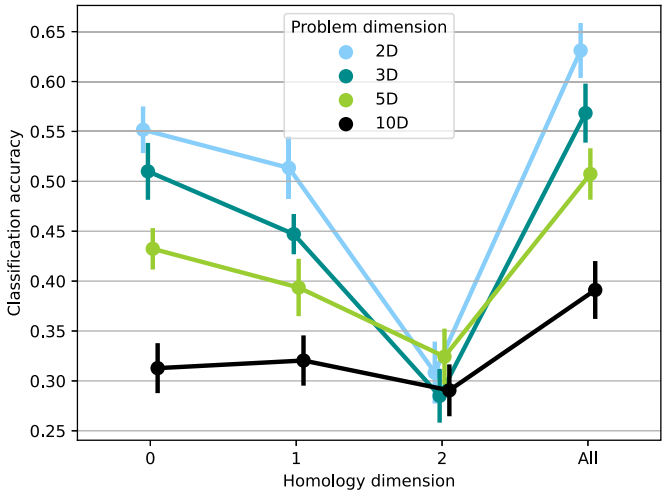


Fig. 21. Influence of homology dimension parameter on classification accuracy.

5.4.5. Sensitivity to the sample size

The quality of the constructed feature representations is proportional to the number of samples extracted from the optimization problem. Therefore, in order to capture as much information as possible, one would prefer to use a large sample size. Unfortunately, computing objective values can be expensive when dealing with a large number of candidate solutions. Therefore it is important to find a good balance between the number of samples and the quality of the representations. Fig. 23 demonstrates the problem classification accuracy obtained using TLA features constructed using different sample sizes. Note that the axis listing the number of samples shows the actual number of samples and not the number of samples adjusted per dimension. That means that we performed between 10 and 600 function evaluations. We see that for 2D problems, a small set of samples can already create informative features. With only 10 samples, the accuracy for 2D problem classification is already around 10% which represents an improvement over the accuracy of 4.2% (majority classifier). For 2D problems with around 400 samples, we reach a state where providing more samples does not build better topological features. For other problem dimensions a similar conclusion can be drawn. With only 10 samples classification based on topological features is only slightly better than the majority class classifier. For 10D problems, having 10 samples is only enough to obtain 6% accuracy while 53% accuracy can be obtained with 400 samples. We see the pattern that the classification accuracy for higher dimensional problems is usually lower than for simpler 2D problems with the same number of samples. We can also observe that after some number of samples, performing additional sampling has diminishing returns on classification accuracy.

A similar comparison of classification accuracy with different numbers of samples can also be performed with ELA features. Fig. 24 shows such a comparison. We can observe that the correlation between the number of samples and model performance holds also for ELA, with larger sample sizes generating more informative ELA features. We can also conclude that ELA features perform better than topological features with all sample sizes when performing classification. But it is important to note that ELA features consist of multiple feature groups.

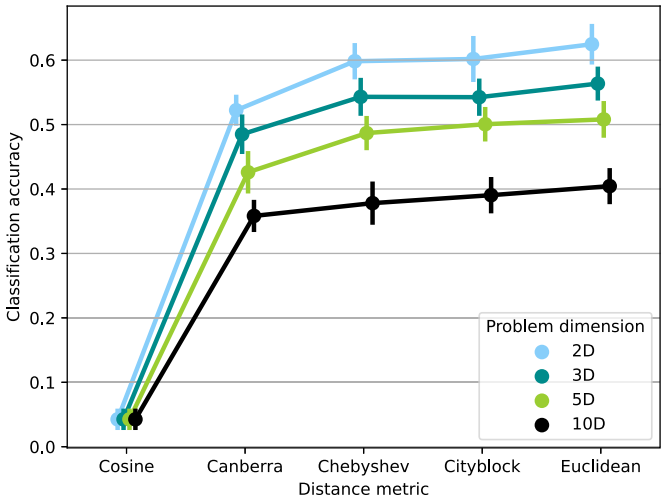


Fig. 22. Influence of different metrics on classification accuracy aggregated over 30 runs. Standard deviations in classification accuracy are shown with vertical lines.

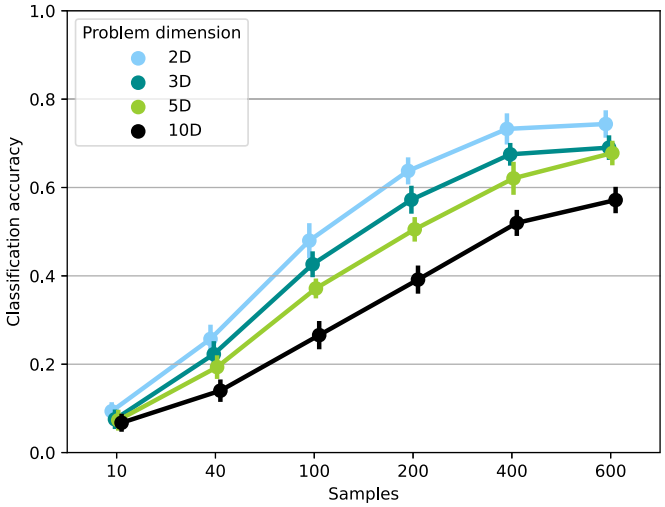


Fig. 23. Influence of the number of samples on the problem classification accuracy using TLA feature group.

5.5. Generalizability of topological features across sample sizes

This subsection investigates if there is a generalizability between feature representations computed with different numbers of samples. With this, we answer the question if topological features are tied to the sample size or do we expect the topological features to have the same value independent of the sample size. In other words, do we expect topological features to be the same given a different number of samples? To answer this question we use the following setup. We first train a model that is able to perform classifications based on topological features computed with 200 samples. Next, we use this model to perform classification on features obtained with a different number of samples. Fig. 25 show the classification accuracy of such a setup for problems of different dimensions. For all problem dimensions, a model trained on topological features from one sample size will have reduced performance when predicting features with different sample sizes. Even when the sample size used to obtain features is changed by 5% between the train and test set, we can immediately see that performance drops by a small percentage. When the difference between the train sample size and the test sample size is even larger, the accuracy quickly drops to what would be expected from a random classifier. This demonstrates that topological features are inherently tied to the sample size used to compute them and that they do not generalize between sample sizes. Implications for this are that machine learning models that are trained with topological features (independent of the task) are inherently linked to the sample size and if changed, the performance of a model is likely to decrease.

6. Limitations

This section lists some limitations of the study. In this study, we test our features for the task of COCO problem instance classification and report/visualize the impact of hyperparameters (size of the smoothing kernel, distance metric, etc.) used during the construction of topological feature values on the classification accuracy.

Joint hyperparameter optimization [65] is recommended for classification, but it has not been done here to avoid overfitting to a relatively small COCO benchmark with only 24 problem classes.

ELA features may have missing, identical, or infinite values for some problem instances and are removed before analysis. In contrast, TLA features are always represented by finite values with no missing values.

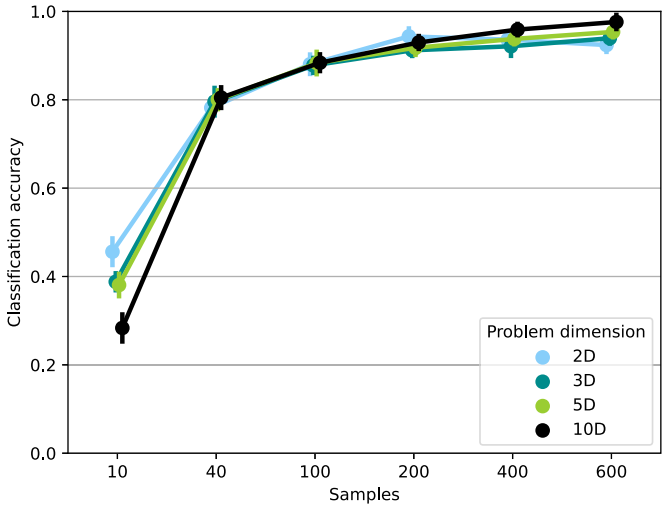


Fig. 24. Influence of the number of samples on classification accuracy using ELA feature group. Note that the axis differs from Fig. 23.

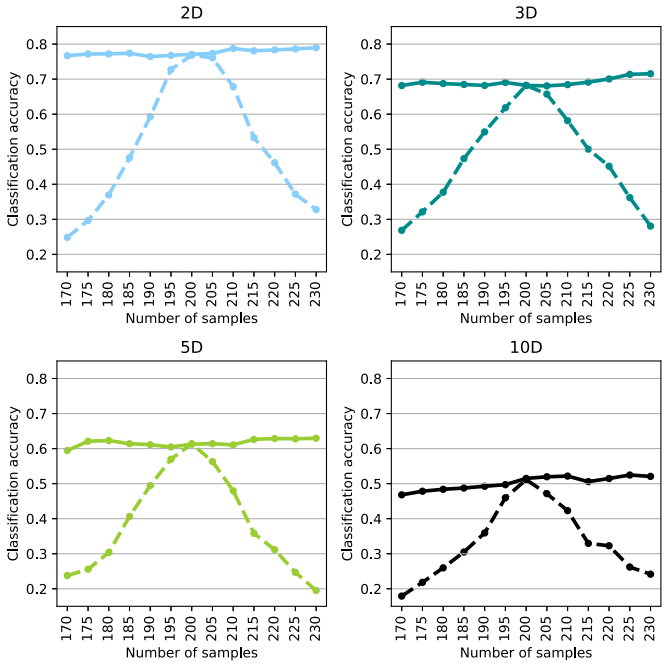


Fig. 25. Classification accuracy of models trained and evaluated on topological features. The solid line shows the performance of models where features use the same sample size while the dotted line shows the model train on sample size 200 and applied to different sample sizes.

Figures in analysis mostly visualize 2D problems because it is easier to link objective functions to topological features and explain them. Hyperparameter sections should only act as a guide for important hyperparameters and their impact on classification accuracy, not as optimal values for higher dimensions or other benchmarks.

When visualizing similarities between features (visualized with heatmaps), we switch between cosine similarity and correlations between features. We use cosine similarity when comparing individual problem instances represented with ELA and TLA feature groups. Some ELA feature groups differ in magnitudes, or are represented by categorical values, or even missing. In such cases, subtracting the mean or standard deviation cannot be performed without losing the meaning of the information in the vectors. On the other hand, when comparing features over all problem instances, the features usually have the same magnitude over all problem instances which allows the use of Pearson correlation.

7. Conclusion and future work

In this work, we expand on the existing concept of Topological Landscape Analysis (TLA) for numerical feature extraction in scenarios where a low number of samples are available. Treating samples as a point cloud, describing them in term of pairwise distances, and applying approaches from topology have numerous advantages and offers a new look at the optimization problems. The extracted features are used to visualize the similarity between problem instances where it can be

seen that they can differentiate between problems. Next, we compare the proposed TLA features with the existing ELA features to determine if such features are similar and how they differ from ELA features. We show that topological features are complementary to existing ELA feature groups and that they cover parts of the feature landscape that is not covered by any other feature group. In terms of classification power, the TLA features achieve accuracy that is comparable to some of the best-performing ELA feature groups. We perform a thorough investigation of how hyperparameters influence the obtained features and consequently, the classification accuracy. With this, we show that hyperparameters carry a large importance for successful feature extraction. We also show that topological features are inherently tied to the sample size and they do not generalize between sample sizes. Such topology-inspired features are potentially useful since they are geared toward detecting and describing the presence of various shapes in high-dimensional spaces. As such, they have the potential to be useful in detecting different aspects of optimization functions as they are designed to act as descriptions of topological and geometric shapes meaning that they might detect funnels and different modalities which have been linked to how hard problem is to solve.

Immediate future steps are investigating how the newly proposed features relate to high-level features and if they can bridge the gap between high and low-level feature groups. Further exploration is needed to reduce the number of highly correlated topological features while maintaining performance through the use of concepts such as Betti numbers and extraction of summary statistics from persistence diagram such as persistence entropy. Additionally, we aim to further explore the theoretical background of the proposed features and the information they capture in comparison to other feature groups. While we only show the task of classifying problem instances, it is crucial to also evaluate topological features on other tasks such as predicting the performance of different optimization algorithms. In this regard, one could explore how TLA features complement the ELA features in predicting algorithm performance and whether they improve the generalizability of algorithm selection models on unseen instances. Furthermore, the current work explores TLA features as a static representation of the problem instance, in that the samples it uses are generated using some artificial sampling technique which aims to cover the entire landscape of the problem instance. Following some recent approaches that calculate features based on samples explored by the algorithm during its execution [34–36], TLA features could also be used to represent the interaction between a problem and an algorithm instance.

CRediT authorship contribution statement

Gašper Petelin: Conceptualization, Methodology, Software, Validation, Writing – original draft, Writing – review & editing, Visualization. **Gjorgjina Cenikj:** Methodology, Software, Validation, Writing – original draft, Writing – review & editing, Supervision. **Tome Eftimov:** Validation, Writing – original draft, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

Funding in direct support of this work: Slovenian Research Agency: research core funding No. P2-0098, project No. N2-0239 to TE, young researcher grant No. PR-11263 to GP, and young researcher grant PR-12393 to GC. We would also like to thank Urban Škvorc and Peter Korošec who were involved in numerous discussions.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.swevo.2023.101448.

References

[1] B. Bischl, O. Mersmann, H. Trautmann, M. Preuß, Algorithm selection based on exploratory landscape analysis and cost-sensitive learning, in: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, 2012, pp. 313–320.
[2] M.A. Muñoz, Y. Sun, M. Kirley, S.K. Halgamuge, Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges, Inform. Sci. 317 (2015) 224–245.
[3] M.A. Muñoz, M. Kirley, S.K. Halgamuge, A meta-learning prediction model of algorithm performance for continuous optimization problems, in: Parallel Problem Solving from Nature-PPSN XII: 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I 12, Springer, 2012, pp. 226–235.
[4] U. Škvorc, T. Eftimov, P. Korošec, Understanding the problem space in single-objective numerical optimization using exploratory landscape analysis, Appl. Soft Comput. (ISSN: 1568-4946) 90 (2020) 106138, http://dx.doi.org/10.1016/j.asoc.2020.106138, URL: https://www.sciencedirect.com/science/article/pii/S1568494620300788.
[5] G. Cenikj, R. Dieter Lang, A. Petrus Engelbrecht, C. Doerr, P. Korošec, T. Eftimov, SELECTOR: Selecting a Representative Benchmark Suite for Reproducible Statistical Comparison, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2022.
[6] M.A. Muñoz, K. Smith-Miles, Generating new space-filling test instances for continuous Black-Box optimization, Evol. Comput. 28 (2020) 379–404, http://dx.doi.org/10.1162/evco_a_00262.
[7] K.M. Malan, A.P. Engelbrecht, A survey of techniques for characterising fitness landscapes and some possible ways forward, Inform. Sci. (ISSN: 0020-0255) 241 (2013) 148–163, http://dx.doi.org/10.1016/j.ins.2013.04.015, URL: https://www.sciencedirect.com/science/article/pii/S0020025513003125.
[8] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, G. Rudolph, Exploratory landscape analysis, in: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, 2011, pp. 829–836.
[9] Q. Renau, C. Doerr, J. Dreo, B. Doerr, Exploratory Landscape Analysis is Strongly Sensitive to the Sampling Strategy, ISBN: 978-3-030-58114-5, 2020, pp. 139–153, http://dx.doi.org/10.1007/978-3-030-58115-2_10, Chapter 0.
[10] U. Škvorc, T. Eftimov, P. Korošec, The effect of sampling methods on the invariance to function transformations when using exploratory landscape analysis, in: 2021 IEEE Congress on Evolutionary Computation (CEC), 2021, pp. 1139–1146, http://dx.doi.org/10.1109/CEC45853.2021.9504739.
[11] R.P. Prager, H. Trautmann, Nullifying the inherent bias of non-invariant exploratory landscape analysis features, in: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), Springer, 2023, pp. 411–425.
[12] G. Petelin, G. Cenikj, T. Eftimov, TLA: Topological landscape analysis for single-objective continuous optimization problem instances, in: Proceedings of IEEE Symposium Series on Computational Intelligence, 2022.
[13] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, D. Brockhoff, COCO: A platform for comparing continuous optimizers in a black-box setting, Optim. Methods Softw. 36 (1) (2021) 114–144.
[14] K.V. Price, A. Kumar, P. Suganthan, Trial-based dominance for comparing both the speed and accuracy of stochastic optimizers with standard non-parametric tests, Swarm Evol. Comput. 78 (2023) 101287.
[15] F. Zou, D. Chen, H. Liu, S. Cao, X. Ji, Y. Zhang, A survey of fitness landscape analysis for optimization, Neurocomputing 503 (2022) 129–139.
[16] G. Wu, R. Mallipeddi, P. Suganthan, Problem Definitions and Evaluation Criteria for the CEC 2017 Competition and Special Session on Constrained Single Objective Real-Parameter Optimization, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore, 2016.
[17] P. Kerschke, H. Trautmann, Comprehensive feature-based landscape analysis of continuous and constrained optimization problems using the R-package flacco, in: Applications in Statistical Computing, Springer, 2019, pp. 93–123.
[18] M.A. Muñoz, M. Kirley, S.K. Halgamuge, Exploratory landscape analysis of continuous space optimization problems using information content, IEEE Trans. Evol. Comput. 19 (1) (2014) 74–87.

[19] P. Kerschke, H. Trautmann, Comprehensive feature-based landscape analysis of continuous and constrained optimization problems using the R-Package Flacco, in: Applications in Statistical Computing: From Music Data Analysis to Industrial Quality Improvement, Springer International Publishing, Cham, ISBN: 978-3-030-25147-5, 2019, pp. 93–123, http://dx.doi.org/10.1007/978-3-030-25147-5_7.

[20] N. Hansen, S. Finck, R. Ros, A. Auger, Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions (Ph.D. thesis), INRIA, 2009.

[21] N. Belkhir, J. Dréo, P. Savéant, M. Schoenauer, Per instance algorithm configuration of CMA-ES with limited budget, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2017, pp. 681–688.

[22] R. Tanabe, Benchmarking feature-based algorithm selection systems for Black-Box numerical optimization, IEEE Trans. Evol. Comput. 26 (6) (2022) 1321–1335.

[23] G. Ochoa, S. Verel, F. Daolio, M. Tomassini, Local optima networks: A new model of combinatorial fitness landscapes, in: Recent Advances in the Theory and Application of Fitness Landscapes, Springer, 2014, pp. 233–262.

[24] R. Morgan, M. Gallagher, Length scale for characterising continuous optimization problems, in: Parallel Problem Solving from Nature-PPSN XII: 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I 12, Springer, 2012, pp. 407–416.

[25] Y. Li, J. Liang, K. Yu, K. Chen, Y. Guo, C. Yue, L. Zhang, Adaptive local landscape feature vector for problem classification and algorithm selection, Appl. Soft Comput. 131 (2022) 109751.

[26] K.M. Malan, A.P. Engelbrecht, Characterising the searchability of continuous optimisation problems for PSO, Swarm Intell. 8 (2014) 275–302.

[27] M.V. Seiler, R.P. Prager, P. Kerschke, H. Trautmann, A collection of deep learning-based feature-free approaches for characterizing single-objective continuous fitness landscapes, 2022, arXiv preprint arXiv:2204.05752.

[28] B. van Stein, F.X. Long, M. Frenzel, P. Krause, M. Gitterle, T. Bäck, DoE2Vec: Deep-learning based features for exploratory landscape analysis, 2023, arXiv preprint arXiv:2304.01219.

[29] A. Kolesnikov, X. Zhai, L. Beyer, Revisiting self-supervised visual representation learning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 1920–1929.

[30] P.H. Le-Khac, G. Healy, A.F. Smeaton, Contrastive representation learning: A framework and review, IEEE Access 8 (2020) 193907–193934.

[31] Z. Liu, Y. Lin, M. Sun, Representation Learning for Natural Language Processing, Springer Nature, 2020.

[32] R.P. Prager, M.V. Seiler, H. Trautmann, P. Kerschke, Automated algorithm selection in single-objective continuous optimization: A comparative study of deep learning and landscape analysis methods, in: International Conference on Parallel Problem Solving from Nature, Springer, 2022, pp. 3–17.

[33] M.-H. Guo, J.-X. Cai, Z.-N. Liu, T.-J. Mu, R.R. Martin, S.-M. Hu, Pct: Point cloud transformer, Comput. Vis. Media 7 (2) (2021) 187–199.

[34] J. de Nobel, H. Wang, T. Baeck, Explorative data analysis of time series based algorithm features of CMA-ES variants, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2021, pp. 510–518.

[35] A. Kostovska, A. Jankovic, D. Vermetten, J. de Nobel, H. Wang, T. Eftimov, C. Doerr, Per-run algorithm selection with warm-starting using trajectory-based features, 2022, arXiv preprint arXiv:2204.09483.

[36] G. Cenikj, G. Petelin, C. Doerr, P. Korošec, T. Eftimov, DynamoRep: Trajectory-based population dynamics for classification of black-box optimization problems, 2023, arXiv preprint arXiv:2306.05438.

[37] A. Atla, R. Tada, V. Sheng, N. Singireddy, Sensitivity of different machine learning algorithms to noise, J. Comput. Sci. Coll. 26 (5) (2011) 96–103.

[38] M. Gidea, Topology data analysis of critical transitions in financial networks, 2017, http://dx.doi.org/10.48550/ARXIV.1701.06081, URL: https://arxiv.org/abs/1701.06081.

[39] A. Nobi, S. Lee, D.H. Kim, J.W. Lee, Correlation and network topologies in global and local stock indices, Phys. Lett. A 378 (34) (2014) 2482–2489.

[40] A.E. Sizemore, J.E. Phillips-Cremins, R. Ghrist, D.S. Bassett, The importance of the whole: topological data analysis for the network neuroscientist, Netw. Neurosci. 3 (3) (2019) 656–673.

[41] M. Gidea, Y. Katz, Topological data analysis of financial time series: Landscapes of crashes, Physica A 491 (2018) 820–834.

[42] S. Zeng, F. Graf, C. Hofer, R. Kwitt, Topological attention for time series forecasting, Adv. Neural Inf. Process. Syst. 34 (2021) 24871–24882.

[43] K. Garside, R. Henderson, I. Makarenko, C. Masoller, Topological data analysis of high resolution diabetic retinopathy images, PLoS One 14 (5) (2019) e0217413.

[44] D. Freedman, C. Chen, Algebraic topology for computer vision, Comput. Vis. (2009) 239–268.

[45] F. Chazal, B. Michel, An introduction to topological data analysis: fundamental and practical aspects for data scientists, 2017, arXiv preprint arXiv:1710.04019.

[46] Y. Skaf, R. Laubenbacher, Topological data analysis in biomedicine: A review, J. Biomed. Inform. (2022) 104082.

[47] S. Dantchev, I. Ivrissimtzis, Efficient construction of the Čech complex, Comput. Graph. 36 (6) (2012) 708–713.

[48] V. Divol, T. Lacombe, Understanding the topology and the geometry of the space of persistence diagrams via optimal partial transport, J. Appl. Comput. Topol. 5 (1) (2021) 1–53.

[49] B. Piccoli, F. Rossi, On properties of the generalized Wasserstein distance, Arch. Ration. Mech. Anal. 222 (3) (2016) 1339–1365.

[50] H. Edelsbrunner, J.L. Harer, Computational Topology: An Introduction, American Mathematical Society, 2022.

[51] S. Agami, Comparison of persistence diagrams, Comm. Statist. Simulation Comput. (2021) 1–14.

[52] H. Adams, T. Emerson, M. Kirby, R. Neville, C. Peterson, P. Shipman, S. Chepushtanova, E. Hanson, F. Motta, L. Ziegelmeier, Persistence images: A stable vector representation of persistent homology, J. Mach. Learn. Res. 18 (2017).

[53] F. Chazal, D. Cohen-Steiner, Q. Mérigot, Geometric inference for probability measures, Found. Comput. Math. 11 (6) (2011) 733–751.

[54] P. Cai, C. Indhumathi, Y. Cai, J. Zheng, Y. Gong, T.S. Lim, P. Wong, Collision detection using axis aligned bounding boxes, Simul. Serious Games Appl. (2014) 1–14.

[55] N. Ravishanker, R. Chen, Topological data analysis (TDA) for time series, 2019, arXiv arXiv preprint arXiv:1909.10604.

[56] C.T. Nathaniel Saul, Scikit-TDA: Topological data analysis for Python, 2019, http://dx.doi.org/10.5281/zenodo.2533369.

[57] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, A. Smola, AutoGluon-tabular: Robust and accurate AutoML for structured data, 2020, arXiv preprint arXiv:2003.06505.

[58] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[59] R. Morgan, M. Gallagher, Sampling techniques and distance metrics in high dimensional continuous landscape analysis: Limitations and improvements, IEEE Trans. Evol. Comput. 18 (3) (2013) 456–461.

[60] P. Kerschke, M. Preuss, S. Wessing, H. Trautmann, Low-budget exploratory landscape analysis on multiple peaks models, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, 2016, pp. 229–236.

[61] P. Kerschke, H. Trautmann, Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning, Evol. Comput. 27 (1) (2019) 99–127.

[62] N. Pise, P. Kulkarni, Algorithm selection for classification problems, in: 2016 SAI Computing Conference (SAI), IEEE, 2016, pp. 203–211.

[63] X. He, K. Zhao, X. Chu, AutoML: A survey of the state-of-the-art, Knowl.-Based Syst. 212 (2021) 106622.

[64] P. Gijsbers, M.L. Bueno, S. Coors, E. LeDell, S. Poirier, J. Thomas, B. Bischl, J. Vanschoren, AMLB: an AutoML benchmark, 2022, arXiv preprint arXiv:2207.12560.

[65] M. Feurer, F. Hutter, Hyperparameter optimization, in: Automated Machine Learning, Springer, Cham, 2019, pp. 3–33.

2.1.1 High-level Properties

Expanding on the topic that was not originally included in the paper, we explore the high-level property prediction of COCO problem instances. The importance of high-level characteristics, such as multimodality, global structure, and funnel structure, should not be underestimated when it comes to automated algorithm selection. As a result, it is a standard practice to identify the high-level characteristics of a specific objective function. In this section, we assess the effectiveness of the proposed features in capturing the overarching attributes of the problem instances found within the COCO benchmark suite. Table 2.1 provides an overview of the high-level characteristics of every COCO problem instance used in this thesis.

Table 2.1: Attributes of the high-level properties of the 24 COCO functions, as defined in (Seiler et al., 2022), are presented. In predictive modeling for these properties, functions labeled with * are excluded since they represent unique cases with distinct properties when utilizing the leave-one-problem-out approach.

BBOB problem class	Multimodal	Global structure	Funnel
1: Sphere	none	none	yes
2: Ellipsoidal separable	none	none	yes
3: Rastrigin separable	high	strong	yes
4: Büche-Rastrigin	high	strong	yes
5: Linear Slope	none	none	yes
6: Attractive Sector	none	none	yes
7: Step Ellipsoidal	none	none	yes
8: Rosenbrock	low	none	yes
9: Rosenbrock rotated	low	none	yes
10: Ellipsoidal high cond.	none	none	yes
11: Discus	none	none	yes
12: Bent Cigar	none	none	yes
13: Sharp Ridge	none	none	yes
14: Different Powers	none	none	yes
15: Rastrigin multimodal	high	strong	yes
16: Weierstrass	high	medium	none
17: Schaffer F7	high	medium	yes
18: Schaffer F7 mod. ill-cond	high	medium	yes
19: Griewank-Rosenbrock	high	strong	yes
20: Schwefel	medium	deceptive*	yes
21: Gallagher 101 Peaks	medium	none	none
22: Gallagher 21 Peaks	low	none	none
23: Katsuura	high	none	none
24: Lunacek bi-Rastrigin	high	weak*	yes

Our validation methodology employs the leave-one-problem-out (LOPO) cross-validation strategy, wherein one problem class is designated for testing while the remaining classes are utilized for training. This procedure is iteratively applied to all 24 problems within the dataset, resulting in the development and assessment of 24 distinct models. We execute 30 iterations of training the AutoGluon classifier, mirroring the methodology employed in

the prior section on classification. We create distinct machine-learning models for each high-level property assessment. A model is trained to assess the degree of multimodality, tasked with predicting four levels, namely, `none`, `low`, `medium`, and `high`. Similarly, another model is dedicated to predicting the global structure, aiming to classify it into one of three categories: `none`, `medium`, or `strong`. It's important to note that we exclude `weak` and `deceptive` global structures due to the impracticality of including them in the training and test datasets, as there is only one problem class associated with these values. Therefore, a separate model is developed to predict the funnel structure, where the targets are limited to two values: `none` and `yes`. We evaluate accuracy using the macro-averaged F1 score as opposed to relying on basic classification accuracy. This choice is motivated by the presence of class imbalance within the dataset, which makes classification accuracy less suitable for accurate assessment.

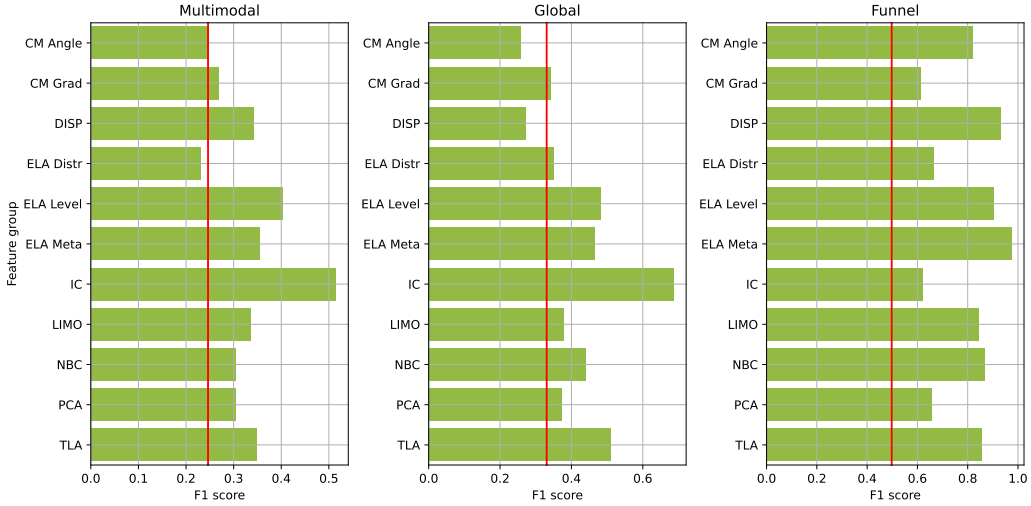


Figure 2.1: The macro-averaged F1 score acquired during the prediction of high-level properties for 2D optimization functions through leave-one-problem-out validation. The red line represents the macro-averaged F1 score that would result from random predictions of these high-level properties.

In Figure 2.1, the depicted graph displays the macro-averaged F1 score achieved when predicting multimodality levels, global structure, or funnel structure using the topological features for 2D problem instances. The figure also provides a comparative analysis of the ELA feature groups and shows a baseline accuracy (red line) that would be achieved if high-level properties were predicted randomly. We observe that, when it comes to identifying multimodality, the *IC* features exhibit the highest level of accuracy, with topological features ranking fourth in terms of accuracy. Regarding the prediction of global structure, topological features are the second most reliable, though they face some difficulty in detecting funnel structures, where their performance is on par with *CM Angle*, *LIMO*, and *NBC* features. This means to some extent topological features can capture function properties that are usually correlated with how hard it is to solve a particular problem.

Note that due to computational complexity, we only evaluate high-level property prediction with 2D features. With higher dimensions, the number of samples has to be artificially limited due to the exponential time complexity of obtaining the persistent image. Additionally, with AutoGluon, a model must be trained for every train/test fold, meaning that AutoGluon has to be trained repeatedly with a large number of features, which again substantially increases computational cost. All of this means that it is difficult to perform the same analysis for higher dimensions within a reasonable timeframe.

2.1.2 Algorithm Selection

Lastly, we focus our efforts on what is arguably the most important task in describing optimization functions using a set of features: the ability to perform optimization algorithm selection based on the proposed features. For this purpose, we use the previously described COCO benchmark, which consists of 24 problem classes, each comprising 15 problem instances. In this context, problems refer to distinct objective functions, and instances are variations of those functions obtained by shifting or scaling. Within the framework of the COCO benchmark, two dominant methods for algorithm selection and performance prediction during validation stand out: “leave-one-instance-out" (LOIO) as referenced in (Nikolikj et al., 2022) and “leave-one-problem-out" (LOPO), highlighted in (Tanabe, 2022). Using the LOPO validation technique, automated algorithm selection is usually performed by removing one COCO problem class from the training set and using it as a test set. A meta-model is then trained on 23 problem classes (each class with multiple instances) to perform algorithm selection. However, leveraging LOPO validation for AS models can be challenging due to the varied characteristics inherent to COCO problems, as discussed in (Tanabe, 2022). During the prediction phase, one might encounter problem instances with properties that have not been previously observed making the LOPO validation strategy extremely challenging (Tanabe, 2022). On the other hand, the LOIO validation strategy tends to be more lenient since the AS model is built using all problem classes. With this validation method, certain problem instances from a specific problem class are designated for training, and the rest are allocated for testing. As a result, when evaluating the meta-model, functions of similar characteristics have been previously encountered, making it easier to choose the most suitable algorithm or predict performance.

For fairness, we show both validation strategies as one is much more changing than the other. In our case, we treat AS as a regression problem where the meta-model is trained to predict relative performances of meta-models. The meta-model we use as well as the preprocessing steps are identical to the ones used in the rest of the paper. In our study, we use the following algorithm portfolio A: GA (Katoch et al., 2021), DE (Price et al., 2006). PSO (Kennedy & Eberhart, 1995), ES (Bäck, 2005) and CMA-ES (Hansen & Ostermeier, 2001). We evaluate AS on 2D problem instances where the algorithms are run for 1000D function evaluations on each problem instance. For the LOPO strategy, we perform cross-validation where instances belonging to one problem are removed from the training set while the LOIO strategy is performed by removing one instance belonging to each problem class. To score the predictions we use the following metric.

First, we calculate the relative performance of algorithms for each run by scaling the obtained solution values. Solutions are scaled so that the algorithm with the best solution receives a score of 0, while the one with the worst solution gets a score of 1. This score is then averaged over 30 independent runs to obtain more robust final scores. If an algorithm is the best in all the runs, its final score will be 0; conversely, if it is the worst in all runs, a final score of 1 is assigned. The equation below describes the calculation in more detail:

$$ns_{a,p,r} = \frac{y_{a,p,r} - \min_{a \in A} y_{a,p,r}}{\max_{a \in A} y_{a,p,r} - \min_{a \in A} y_{a,p,r}} \tag{2.1}$$

$$s_{a,p} = \frac{1}{R} \sum_{r \in |R|} ns_{a,p,r} \tag{2.2}$$

In this case $y_{a,p,r}$ is the best solution discovered by the algorithm a on problem p in run r. The size of the algorithm portfolio and the number of runs are represented by the $|A|$ and $|R|$ respectively. The first equation $ns_{a,p,r}$ represents the normalized score of an algorithm for a particular run while $s_{a,p}$ is the average normalized score over all runs.

When the meta-model selects an algorithm, the error of its prediction for a particular problem is calculated by taking the normalized $s_{a,p}$ of the algorithm that is predicted as the best algorithm and subtracting the actual best score from it. More details are available in (Cenikj et al., 2024).

To provide a concrete example. Let's assume we have three algorithms [DE, GA, CMA-ES] that achieve the best objective values [13.5, 14.2, 13.4] in the first run and [13.4, 14.0, 13.1] in the second run. These two runs would be scaled into [0.125, 1.000, 0.000] and [0.333, 1.000, 0.000]. After this, the two runs are averaged to get the mean relative performance of algorithms for the problem to be [0.229, 1.000, 0.000]. In this case, if algorithm CMA-ES was selected as the best one we would get an error of 0 while if we select DE as the best algorithm, we get an error of 0.229 as this represent in relative term how much worse the selected algorithm is to the best one.

We opt for this error metric over alternatives like classification accuracy because of the way it addresses misclassification. When an incorrect algorithm is selected, the added penalty is relative to the performance of other algorithms. For instance, if two algorithms exhibit nearly identical results but one is marginally superior, selecting the incorrect one as the best incurs only a minor penalty. This implies that the strategy is more forgiving when the chosen algorithm is close in performance to the optimal one.

The suggested metric is designed to measure how far off our performance is expected to be, compared to the best and worst-performing algorithms. If we always chose the optimal algorithms (the virtual best solver), the error metric would be zero. Utilizing topological features for algorithm selection via the LOIO strategy results in an error of 0.0536, whereas the error for the single best solver is 0.0992. This indicates that the error with the SBS is nearly cut in half when AS is conducted using topological attributes. When the same methodology is used with ELA features instead of topological ones, the error marginally decreases to approximately 0.04096, suggesting that ELA features have a slight edge over the proposed features in algorithm selection. However, the LOIO strategy is a relatively easy one compared to the LOPO evaluation strategy

Adopting the LOPO approach, topological features marginally exceed the performance of SBS. Here, topological features result in an error rate of 0.09641 compared to SBS's error rate of 0.09732. This suggests that when facing new problems that differ significantly from the training distribution, the advantage of topological features over SBS is almost nonexistent. Similarly, ELA features attain a slightly lower error rate of 0.09461, which, although better than both topological features and SBS, is not sufficient to be deemed statistically significant. A conclusion can be drawn that topological features, as well as ELA features, can perform well if similar problems have been encountered before. However, they are not able to select algorithms for problems deemed dissimilar, such as the problem classes in the COCO benchmark.

2.1.3 Statistical Analysis

Here, we provide a statistical analysis of models for all three tasks: classification, high-level property prediction, and algorithm selection specifically for the 2D problems. It is important to note that, since we use random repeated train/test splits, performing proper statistical analysis is extremely challenging as the runs are not independent. In other words, the independence condition required by many statistical tests is violated, and using such tests would yield results that are not statistically valid. To clarify, we performed 30 train/test splits, training the model with each feature portfolio and evaluating their accuracy on the test set. Because we use 30 random splits with different seeds, some instances overlap between the train splits, introducing dependency in the results obtained across different splits (Demšar, 2006). Nevertheless, we provide a statistical comparison

under the assumption that the classification accuracies from 30 splits are independent.

For problem instance classification and high-level property prediction, we first conducted an evaluation involving the assessment of normality and variance. Using the Shapiro-Wilk test with a significance level of 0.05, we found that for classification, none of the feature portfolios except the `ela_meta` group follow a normal distribution, whereas for high-level property prediction, none of the feature portfolios are normally distributed, which is important for selecting subsequent tests. Additionally, Levene's test revealed that the performances obtained from different feature portfolios have unequal variances. As at least one set of performances is not normally distributed and the variances are unequal, we applied the Friedman test, which resulted in a p-value close to 0.00, indicating statistically significant differences between the results of different feature portfolios for both tasks. To identify specific pairs of feature portfolios with significant differences, the Nemenyi test was performed. For classification, the feature sets `ic`, `ela_meta`, and `pca` statistically outperformed our proposed topological features. The `tla` feature set achieved performance statistically similar to `ela_level`, `limo`, and `nbc`, while all other feature sets, including the baseline model, performed worse as shown in Figure 2.2. These findings highlight the variability in performance among feature portfolios and emphasize the importance of selecting appropriate feature sets for classification and high-level property prediction.

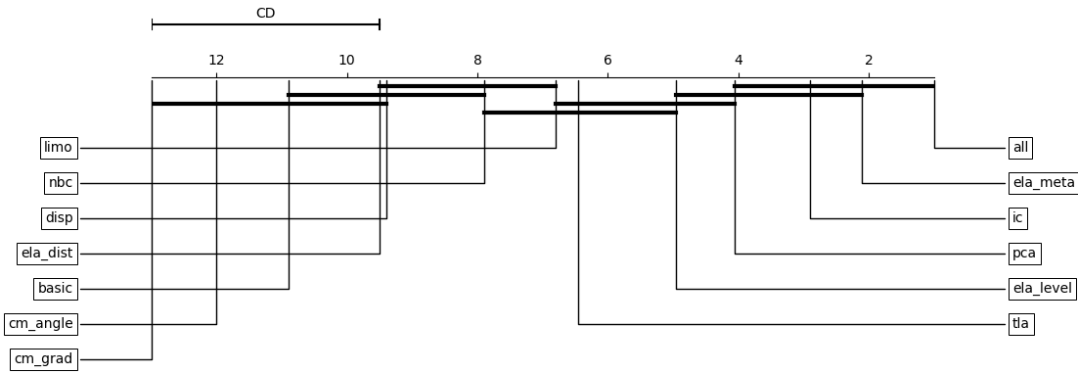


Figure 2.2: The critical diagram illustrates the results of the Neymani test. Feature portfolios connected by lines indicate that there is no statistically significant difference between their classification accuracies, while those not connected show a significant difference.

For high-level property prediction, particularly for multimodality, the `tla` feature set performs statistically worse than `ic` but exhibits statistically similar performance to `limo`, `ela_meta`, `ela_level`, and `disp`, with all other feature sets performing worse. For global structure prediction, only the `ic` features outperform the `tla` features, while `ela_meta` and `ela_level` achieve performance equivalent to the `tla` feature set. Lastly, for funnel structure prediction, the feature sets can be categorized into two groups: the first group includes feature sets with an F1 score above 0.8, all of which demonstrate statistically similar performance, and the second group comprises the remaining feature sets with an F1 score below 0.65, forming another statistically similar group of approaches.

Lastly, we perform a similar analysis for the task of AS for both LOIO and LOPO. The only difference from the previous example is that, in this case, all ELA feature groups are concatenated together and subsequently compared with the TLA. Using the same approach as before, we conclude that for LOIO, the ELA feature groups significantly outperform the proposed topological features at a significance level of 0.05. Additionally, topological features also significantly outperform the proposed baseline. However, for LOPO, the results are substantially different. As discussed in the previous section, for LOPO, none

of the AS models, including those based on both TLA and ELA features, statistically outperform the proposed baseline.

2.1.4 Discussion

This chapter introduces a new set of topological features for use in continuous single-objective optimization, specifically for problem differentiation, high-level property prediction, and algorithm selection. We investigate two hypotheses: **H1a** - *Using insights from topological data analysis, it is possible to construct rotation and translation invariant features that differentiate between single-objective optimization problems and capture their high-level properties*, and **H1b** - *Performance of predictive models for optimization algorithm selection based on such topological features is comparable to the performance of models based on existing exploratory landscape analysis features*, as defined in Section 1.3.

In the presented paper, we show that the proposed topological features can differentiate between optimization problems and successfully predict high-level properties. In all cases, the proposed features perform statistically significantly better than the baseline, confirming the validity of hypothesis **H1a**.

The second approach in our study focuses on the use of topological features for algorithm selection. Here, we demonstrate that while these features significantly outperform the baseline in the LOIO evaluation, they do not perform better in the LOPO evaluation. Additionally, and most importantly, the topological features underperform compared to the ELA features in LOIO, with no significant difference in LOPO. Based on these findings, we conclude that hypothesis **H1b** is rejected, as ELA features exhibit better performance with certain evaluation methodologies.

The main contribution of the proposed approach is that the features capture rotation- and translation-invariant topological properties of an objective function and that these features indeed contain information useful for certain downstream tasks commonly encountered in optimization.

2.2 Random Filter Mappings Features

The second approach to feature construction, as described in the paper *Random Filter Mappings as Optimization Problem Feature Extractors*, addresses continuous single-objective optimization problems by constructing features using random functions, or what is commonly referred to as the random mapping technique. Random functions for feature extraction have gained prominence, particularly in fields like computer vision and time series analysis, where randomly initialized convolutions are used to obtain representations of images or time series data. The proposed approach demonstrates that similar techniques can be effectively applied to construct features for continuous single-objective optimization problems.

The paper investigates hypotheses **H2a** and **H2b** and presents several important contributions. Firstly, the proposed features are shown to create representations that complement existing ones, enhancing the existing feature set. Additionally, these features can be effectively used for problem classification, providing valuable insights into the nature of the optimization problems. They also aid in the prediction of high-level properties. Finally, the proposed approach proves useful for algorithm selection in cases where new objective functions are somewhat similar to previously encountered ones.

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 26 July 2024, accepted 16 September 2024, date of publication 26 September 2024, date of current version 10 October 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3468723

RESEARCH ARTICLE

Random Filter Mappings as Optimization Problem Feature Extractors

GAŠPER PETELIN, (Member, IEEE), AND GJORGJINA CENIKJ, (Student Member, IEEE)
Computer Systems Department, Jožef Stefan Institute, 1000 Ljubljana, Slovenia
Jožef Stefan International Postgraduate School, 1000 Ljubljana, Slovenia

Corresponding author: Gašper Petelin (gasper.petelin@ijs.si)

This work was supported by Slovenian Research Agency under Grant P2-0098. The work of Gašper Petelin was supported by the Young Researcher Grant PR-11263. The work of Gjorgjina Cenikj was supported by the Young Researcher Grant PR-12393.

ABSTRACT Characterizing optimization problems and their properties addresses a key challenge in optimization and is crucial for tasks such as creating benchmarks, selecting algorithms, and configuring them. Although several techniques have been proposed for extracting features from single-objective optimization problems, the proposed approach offers an alternative look at these problems and their properties. We propose an approach for creating problem representations by utilizing domain-specific filters. These filters have randomly initialized weights and are applied to samples of the optimization problem to extract relevant properties. Proposed features are subsequently used to classify problem instances from the Comparing Continuous Optimizers benchmark demonstrating that problem instances of the same problem tend to be situated near each other in a high-dimensional feature space. Additionally, we demonstrate that the proposed feature extraction method can be used to recognize complex characteristics of optimization functions, including multimodality and the presence of global and funnel structures. We also explore the extent to which these identified features can assist in the selection of algorithms. Our findings reveal that these features are suitable for constructing meta-models for algorithm selection, provided that the problems encountered do not substantially differ from those seen in the training phase. The proposed approach offers a versatile tool for feature extraction, highlighting its applicability across multiple tasks within the domain of optimization.

INDEX TERMS Domain-specific filters, random mapping feature extraction, representation learning, single objective numerical optimization.

I. INTRODUCTION

Several tasks in numerical optimization require the representation of single-objective continuous optimization problems in terms of numerical features, commonly referred to as problem landscape features. Automated algorithm selection (selecting the best algorithm for each problem) [1] and configuration (finding the best hyper-parameters for each algorithm) [2] rely on the existence of numerical features that characterize problem instances. These features enable the training of machine-learning-based models, where the calculated features are used to predict algorithms' performance, choose the best algorithm for a given problem instance, or produce a ranking of algorithms based on expected performance. Additionally, numerical features play a critical role in benchmarking by facilitating analyses of problem instance similarity [3], evaluating the representativeness and redundancy of problem spaces in existing benchmark suites, and identifying over or under-represented areas of problem landscapes in available benchmark suites [4], [5].

In the past decade, a notable effort has been dedicated to problem landscape feature construction in continuous and discrete optimization. The field of literature presents numerous methodologies available for capturing various high and low-level properties of optimization functions, which can assist in tasks like algorithm selection and benchmarking. Fitness Landscape Analysis (FLA) [6], [7] features have been proposed, which characterize problem instances in terms of hand-crafted high-level features. However, since expert knowledge is required to construct such features, they are

The associate editor coordinating the review of this manuscript and approving it for publication was Jenny Mahoney.

not very beneficial for automated algorithm selection and configuration. For this reason, several works have proposed Exploratory Landscape Analysis (ELA) features [8], which are calculated using mathematical and statistical functions applied to candidate solutions of the objective function. Even though they are widely used in different studies, the ELA features have some disadvantages, such as high computational complexity for high-dimensional problems, sensitivity to variations in the sample size and sampling method [9], [10], lack of invariance to transformations of the optimization problem (such as scaling and shifting) [3], [10], [11] and limited generalization capabilities [12], [13]. The constraints of current methods suggest that additional investigation is required to create features that can provide novel perspectives on black-box optimization problem attributes. These features could potentially enhance the development of algorithm selection systems and contribute to the improvement of benchmark quality.

A. OUR CONTRIBUTION

We introduce a new approach to construct features called Random Filter Mapping (RFM). This method involves applying randomly initialized filters over a set of samples from the objective function, which produces filter responses. The representation of the optimization problems is then constructed by aggregating the produced responses into numerical features to capture useful information about the objective functions. To evaluate the proposed features, we provide an in-depth analysis of the following research questions:

RQ1: Can the proposed features adequately differentiate between problem classes in the COmparing Continuous Optimizers (COCO) benchmark suite [14]? This matter has been explored in prior research [15], [16], [17]. It is important to determine if problems with comparable properties are closely clustered in the latent space. Moreover, we evaluate the effectiveness of these features by comparing them to the ELA feature groups. Our results indicate that the suggested features demonstrate relatively good performance in classifying problem instances into the 24 BBOB problem classes, ranking third in terms of classification accuracy when compared to 10 ELA feature groups. In connection with this question, we also investigate the information content of individual features. As these features are generated using random mappings, they tend to exhibit relatively low capability in distinguishing between problem instances, achieving a maximum accuracy of around 30% when a single feature is used individually. However, the accuracy improves as more features are generated and considered together.

RQ2: Can the proposed features be employed for detecting high-level properties of the objective function as previously explored in [8], [18], and how do they compare with ELA features? We assess the effectiveness of the proposed features in recognizing high-level attributes of problem classes that have not been encountered before. This task is relatively challenging as it involves predicting properties such as multimodality and global/funnel structure in a function that hasn't been observed previously. Our results demonstrate that the proposed features are ranked among the top feature groups for identifying global/funnel structure, but they face a slightly greater challenge in identifying the multimodality of the problem.

RQ3: Can features be used for algorithm selection, and how do they compare with existing ones? We demonstrate that the proposed features excel at performing algorithm selection when the training and test sets do not significantly differ, employing a leave-one-instance-out approach, only slightly underperforming compared to the comprehensive ELA feature set. However, in a more challenging validation strategy with a leave-one-problem-out approach, where the training and test sets differ greatly, both the proposed and existing features only marginally outperform the simple baseline.

Apart from the aforementioned questions, we explore why these features are effective and the type of values they generate. This exploration is crucial due to the counterintuitive nature of generating features from randomness. We demonstrate that these features are quite adept at transforming objective functions into higher-dimensional spaces, where they become separable. Subsequently, it becomes the responsibility of the machine learning model to explore how to leverage this separability for various tasks.

B. OUTLINE

We organize our paper as follows. In Section II literature review is provided, describing existing feature extraction approaches in the optimization domain and approaches from other domains. Section III provides a detailed description of how feature extraction is performed, followed by Section IV which includes a comparison of the proposed approach with existing methods, an evaluation of the proposed features for the problem classification and high-level property prediction task, and a hyperparameter sensitivity analysis. In Section V we provide some limitations of the study together with the discussion. Finally, in Section VI, we conclude our work and provide some further research directions.

C. REPRODUCIBILITY

The code used to extract features and analyze results is available on GitLab repository at https://repo.ijs.si/gpetelin/random-convolutions, while the features used in the analysis are available at Zenodo https://doi.org/10.5281/zenodo.7903995.

II. RELATED WORK

This section, describing the related work, is split into three parts. Part II-A gives an overview of the feature extraction techniques for single objective optimization problems, part II-B refers to similar random mapping techniques that were successfully used in other machine learning domains,

and part II-C introduces some of the existing filters (often also referred to as kernels) approaches used for low-dimensional point cloud information extraction.

A. FEATURE EXTRACTION FOR SINGLE OBJECTIVE OPTIMIZATION PROBLEMS

In the single-objective continuous optimization domain, the characterization of optimization problems in terms of their properties is crucial for: *i)* obtaining a deeper understanding of problems and their characteristics; *ii)* constructing benchmarks that contain problems with a wide variety of different properties; *iii)* designing or selecting algorithms that have a high likelihood of working well on a given problem based on its characteristics.

Characterizing optimization problems can be performed in multiple ways. One way of characterizing the properties of problems are so-called Fitness Landscape Analysis (FLA) [6], which are high-level problem descriptors capturing properties such as global structure, separability, variable scaling, multimodality, search space homogeneity, etc. Such properties can be extremely useful in getting insight into problems, however, a major downside is that they are not computed automatically but are usually assigned by human experts, which limits their usefulness for performing automated algorithm selection and configuration. FLA features are therefore often known only for some optimization functions found in benchmarks like COCO [14], [19] and CEC [20]. Due to such limitations, low-level landscape features were adopted. These features are computed by first obtaining a set of candidate solutions and evaluating them on the problem of interest. Based on obtained samples, features are then computed automatically without the need for human knowledge.

One of the most well-known low-level features are the so-called Exploratory Landscape Analysis (ELA) [8] features. When initially proposed, they only contained a few feature groups, however, over the years the number of feature groups increased. While all the feature groups were not proposed in the original ELA paper [8], it is common to refer to original features as well as features proposed at later points [21], [22], [23], [24], [25] as ELA features. When we talk about ELA features or feature groups we refer to the methods that are implemented in the *flacco* [26], [27] library. As opposed to high-level features that are defined by a human and are not computed based on the obtained samples, ELA features are intrinsically tied to the samples from which they are obtained. In [9], [10], and [28] it is demonstrated that ELA features are sensitive to the sample size and the exact sampling strategy used to extract them. Another downside of ELA features is the computational cost associated with computing them and the fact that computational can be prohibitively expensive for high-dimension problems with a large number of samples [29]. Besides the features available as part of the *flacco* library, other types of features were also proposed [17], [18] with a more comprehensive review available in [7].

B. RANDOM MAPPING FEATURE EXTRACTION TECHNIQUES IN MACHINE LEARNING

As explored in the previous section, constructing informative features of single objective optimization problems is challenging and requires carefully crafted techniques. While such techniques can be extremely helpful, they can encode some inductive biases stemming from the understanding of how properties of a single objective function should be encoded. In recent years, feature construction techniques have emerged where random mapping techniques are used to get insight into a problem. Random mapping techniques have been used in many different domains of machine learning but are most prominent in the domains of tabular data, computer vision, and time series processing. In [30] authors explore the idea of mapping the data to a randomized low-dimensional feature space and using existing machine learning techniques to perform classification/regression. With such a technique of performing randomized mappings, they obtain state-of-the-art in terms of accuracy and computational cost. Similar techniques of randomized mappings were also adapted in the field of computer vision where papers such as [31] and [32] discovered that random filters (as opposed to learned filters during the optimization process) can capture useful information that can be used for the classification task. In [33] these phenomena of using random filters are further investigated where authors try to determine why random filters sometimes perform well for image classification. They conclude that in the computer vision domain, good network architectures can sometimes perform well, even when using randomly initialized weights. In [34] random convolutions are used as a data augmentation technique while [35] incorporates randomness with randomly shuffling convolution filters. Similarly, such approaches have also been used to construct representations that are useful in the time series domain. In [36] authors explore the use of 1D filters of different sizes with randomized weights that can be used for the task of time series classification. Such an approach based on random convolutions can achieve accuracy comparable to state-of-the-art time-series classification algorithms. This approach was further refined in [37] where improvements to the filter initializations significantly reduce the number of required filters.

C. POINT CLOUD FILTERS

Extracting useful information with the help of filters has become extremely popular in many different machine-learning domains such as computer vision [38], [39], natural language processing [40], [41], time-series and signal analysis [42], [43] as well as point cloud classification that we explore in more details next. Each of these domains has specially tailored types of filters that encode different domain-specific indictive biases [44].

In this section, we provide a summary of filters that are predominantly designed to operate on 3D point clouds, as they bear the closest resemblance to our newly suggested filters. This is because the samples acquired from the optimization problems can be viewed as a high-dimensional point cloud. In practice, numerous filters have been proposed to classify point clouds. Some examples of such filters proposed specifically for point clouds are Kernel Point Convolution [45], Global Context Aware Convolutions [46], Spherical filter [47] for point cloud classification with graph neural networks, and PointCNN [48] where convolution is applied on so-called X-transformed points. Despite a lot of research in extracting information with the help of filters, most of the approaches are inherently limited to the domain for which they were developed. For example, point clouds are most often 3-dimensional, therefore filters are specifically designed to handle only 3 dimensions. Moreover, existing point cloud filters assume that all dimensions of a point cloud are equally important which might not be the case in samples obtained from an optimization problem. A more detailed overview of different approaches to constructing filters can be found in [49]. In the domain of black-box optimization, convolutional approaches have already been explored in [18] and [50]. However, this work applies filters that are tailored to work with images which makes their use on a set of samples obtained from optimization problems somewhat cumbersome, especially for problems with three or more dimensions.

III. METHODOLOGY

In this section, we describe the Random Filter Mapping (RFM) methodology of extracting features from samples of the objective function. The full feature extraction process can be roughly divided into the following steps: *i)* initialize a set of filters with random sizes, weights and radii, *ii)* generate candidate solutions and evaluate them on a given problem, *iii)* apply scaling to obtained samples, *iv)* for each set of samples repeatedly apply filters at random points to compute filter responses, *v)* aggregate responses from each individual filter to obtain a numerical vector representation of a problem. Figure 1 provides a rough overview of the proposed feature extraction methodology with two randomly initialized filters, each one being applied twice. This produces two filter response vectors for each of the two filters. Response vectors are then aggregated into two numerical features describing the numerical optimization function. It is extremely important to note that when filters are randomly initialized and applied to extract features from samples from different problems, the same filters should be applied to all the problems. Randomly initializing filters for each problem separately will produce inconsistent and meaningless features.

A. INITIALIZING RANDOM FILTERS

Random filters are the building block of the proposed feature extraction methodology. This section provides a detailed description of how filters are designed and initialized.

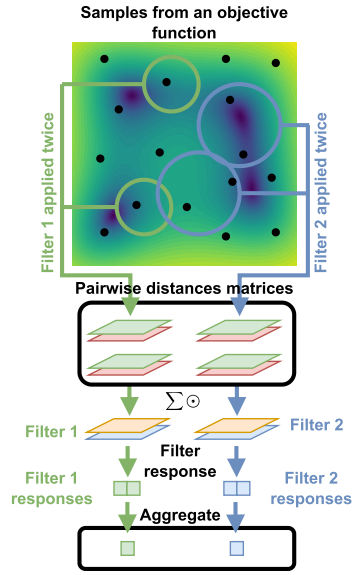


FIGURE 1. An example of applying two randomly initialized filters over the domain of samples two times, obtaining four filter response values (two for each filter) that are then aggregated into two feature values.

The initialization of each filter is parameterized using two hyperparameters. The first hyperparameter of the randomly initialized filter is its **size**. The size of the filter describes how many samples are needed so that the filter can be successfully applied. The size hyperparameter also determines the size of two matrices that are used to transform samples into filter responses. Figure 2 shows an example of a randomly initialized filter of size 3. The filter of size 3 is composed of two 3×3 matrices. The first matrix W_x (on the left) is used to weigh the distances between the proposed candidate solutions while the second matrix W_y (right) is used to quantify the differences between objective values of samples. The second hyperparameter of the filter is the filter **radius**. This hyperparameter controls how close together are the samples. The weights of both matrices are samples from the normal distribution $N(0, 1)$. Using this filter weight initialization approach, the majority of weights are initialized with small magnitudes and are close to zero, but have the ability to adopt larger values. A filter of size s will have $2\binom{s}{2}$ weights. The number of weights of the filter only depends on its size and not on the dimensionality of the problem. Only the lower part of the matrices is initialized while the diagonal and all the values above it are set to 0, due to the fact that the distance matrix is symmetric. More details about the exact process of how random filters are then applied to extract features are provided in subsection III-D.

The above description provides details only on how a single filter is initialized. To achieve good representations, many diverse filters are needed. With diversity, we refer to filters with different sizes, radii, and weights. With the goal of achieving variety, many filters with different hyperparameters and weights are initialized. Filter sizes are uniformly samples to be between 3 and 14. This means that the smallest filters

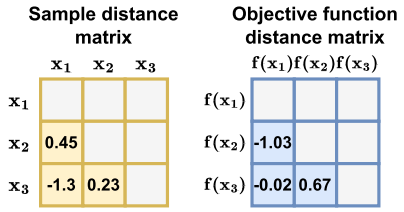


FIGURE 2. An example of a filter of size 3 that consists of 2 matrices and 6 weights. The left matrix W_x captures the distances between candidate solutions while the right matrix W_y weights the differences in their objective function values.

will be applied on only 3 samples while the largest filters of size 14 require 14 samples for the filter response to be computed. The radius hyperparameter is randomly selected to be between 0 and $0.5\sqrt{D}$ with D being the dimensionality of the problem to which the filter is being applied to. The reason for such scaling of the radius parameter is the following: If samples are scaled to be between $[0, 1]^D$ as described in the subsection III-C, then the Euclidean distance between two candidate solutions $\vec{x_i}$ and $\vec{x_j}$ can be at most $\sqrt{D}$. In other words, two candidate solutions that are enclosed in the range $[0, 1]^D$ can be at most $\sqrt{D}$ apart when measured with Euclidian distance. As a result, the factor for correcting the dimensionality, $\sqrt{D}$, is utilized to reduce the filter radius' sensitivity to the problem dimension. With this correction and appropriate sample scaling, a filter having a radius of $0.5\sqrt{D}$ can encompass the entire $[0, 1]^D$ domain.

As opposed to other domains (such as computer vision and time series analysis), where filter hyperparameters are most often hand-selected while weights are learned through gradient descent, with the proposed methodology both hyperparameters and weights are selected randomly. It is extremely important to note that once a filter is initialized, the weight and hyperparameters of the filter never change (i.e. weights are never updated during feature extraction and learning). In other words, no gradient is computed with respect to the weights of the filters.

B. OBTAINING SAMPLES FROM THE OBJECTIVE FUNCTION

Similar to the existing features extraction techniques, the first step of the proposed feature extraction methodology is to create a set of N candidate solutions $\vec{x_1}, \vec{x_2}, \ldots \vec{x_N}$. Here, the length of the vector $\vec{x_i}$ is equivalent to the problem dimensionality D. The candidate solutions are then evaluated on a given optimization problem to obtain their objective function value $f(\vec{x_i})$. Our methodology does not dictate what sampling methodology has to be used. One can use any sampling technique but we only test the approach with Latin Hypercube sampling (LHS) from [51].

C. SAMPLE SCALING

When samples are obtained, they are first scaled before being passed to the feature extraction pipeline. This serves as a preprocessing step that simplifies feature extraction and makes some hyperparameters (such as the filter range described in Section III-A) independent of the ranges from which candidate solutions were obtained. In our case, the candidate solutions $\vec{x_1}, \vec{x_2}, \ldots \vec{x_N}$ are scaled in such a way that each vector component is between 0 and 1. For example, if candidate solutions were obtained from the range $[-5, 5]^D$, a so-called min-max linear transformation on each component of the vector is applied, moving the points in a range $[0, 1]^D$. A similar scaling process is also applied for the objective values $f(\vec{x_i})$ of candidate solutions. Objective values are min-max scaled to be in the range $[0, 1]$ in such a way that the largest objective function value equals 1 while the smallest/best objective value equals 0 (assuming minimization of the objective function). Such scaling ensures that objective functions are comparable if scaled differently.

D. APPLYING RANDOM FILTER

When filters are initialized and samples are properly scaled, the next step is repeatedly applying each individual filter at random points over the whole domain to produce a vector of responses for each filter separately. Each individual filter is repeatedly applied in the following way. First, a point is randomly selected (using uniform random sampling) to be between $[0, 1]^D$. We will refer to this point as an **anchor point** and mark it with $\vec{x}_{anchor}$. When the anchor point is selected, all the samples that are closer to the anchor point than the filter's dimension-adjusted radius are considered to be viable candidates to which the filter can be applied. In this case, closeness to the anchor point is evaluated as the Euclidian distance between vectors $\vec{x}_{anchor}$ and $\vec{x_k}$. Given candidates inside a filter range, a subset of samples is selected. This subset of selected samples has to be equal to the filter size. Figure 3 shows an example of how a filter of size 4 is applied. Four points (black) are randomly selected from a set of points that are at most filter-defined radius away from the anchor point (marked with a cross). Selected four points are then used for filter application.

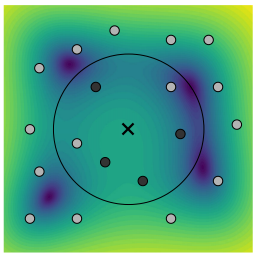


FIGURE 3. Selecting a subset of three samples (black dots) that are at most selected radius away from the anchor point marked with a cross. A filter is then applied on the pairwise distances of the selected subset of four points to obtain a filter response.

After selecting a subset of samples, the distances between each pair of samples in that subset are calculated. The Euclidean pairwise distances between the subsampled candidate solutions are denoted by M_x, while the pairwise distances between the function values of those candidates are denoted by M_y. In other words, M_x describes how far apart

are the subsampled candidate solutions, while M_y describes the distances in their objective values. To give a concrete example. Let us assume that for a filter of size 3, candidate solutions $\vec{x}_{162}, \vec{x}_{391}$, and $\vec{x}_{470}$ were included in the subsample. The M_x would contain the following values: $M_x^{(1,0)} = dist_{Euc}(\vec{x}_{162}, \vec{x}_{391})$, $M_x^{(2,0)} = dist_{Euc}(\vec{x}_{162}, \vec{x}_{470})$ and $M_x^{(2,1)} = dist_{Euc}(\vec{x}_{391}, \vec{x}_{470})$. The matrix M_y quantifying distances in objective values would be comprised of values $M_y^{(1,0)} = dist_{Euc}(f(\vec{x}_{162}), f(\vec{x}_{391}))$, $M_y^{(2,0)} = dist_{Euc}(f(\vec{x}_{162}), f(\vec{x}_{470}))$, $M_y^{(2,1)} = dist_{Euc}(f(\vec{x}_{391}), f(\vec{x}_{470}))$.

The obtained pairwise distances are then multiplied by the weights of the filter to get the filter response in the following way:

$$response = \sum W_x \odot M_x + \sum W_y \odot M_y \tag{1}$$

In summary, anchor points are iteratively selected and samples close enough are considered for filter application. From those samples that meet proximity criteria, a random subset of samples is selected on which the filter is applied to produce a response. Filter application, in this case, is simple multiplication between pairwise distances of the selected samples and filter weights.

E. AGGREGATING FILTER RESPONSE VALUES

When a single filter is repeatedly applied randomly over the whole domain several times, a vector is obtained that holds all the response/activation values of the filter. Such filter response values however have to be aggregated together into a meaningful single numerical value (see [36], [52]). The rationale for this aggregation is as follows: A particular filter may generate strong responses when a subset of samples shows a specific property, such as a local optimum, funnel, or gradient. By repeatedly applying filters and aggregating the results, we ensure that such strong responses are not merely occurring for one particular subset of samples, but they persist across various samples and are a more reliable indicator of the problem's local and global structure

Aggregate functions in our case have to be permutation invariant meaning that values have to be the same regardless of the order in which they are passed to the aggregate function. The literature proposes many such aggregate functions. In [36] when performing time-series classification, the proportion of positive values (*ppv*) and maximum value (*max*) are used as the aggregate functions. In other domains [53], popular aggregate functions are also mean value (*mean*), standard deviation (*std*), and skewness (*skew*) of the response values. In this paper, we use all five aggregate functions. This means that when a single filter is applied multiple times, the response values are aggregated with five different aggregate functions forming five features that describe a single objective function. It is important to note that in some cases filter responses can be invalid (e.g. when the filter radius is small and there are not enough samples for a filter to be applied). Such response values are ignored when performing aggregations meaning that they are removed before the aggregation function is applied. The most extreme case is when all filter applications fail (such as a filter with a radius of 0.0) and aggregation is not possible. In such cases where aggregation cannot be performed, the corresponding feature will have a missing value.

F. FEATURE EXTRACTION PSEUDOCODE

Pseudocode 1 outlines the individual steps of the Random Filter Mapping (RFM) features generation technique, providing a clearer understanding of the step-by-step process. The function *initialize_filters* will initialize $n_{filters}$ filters with random weights, sizes, and radii (see III-A). Next, for each problem p from a set of problems P, samples are obtained using *problem_samples* (see III-B) and scaled with *scale_samples* (see III-C). With this, each filter is applied multiple times (denoted with $n_{applications}$) with randomly selected anchor points $\vec{x}_{anchor}$ obtained with function *random_point*. A subset of points, on which a filter is applied, is randomly selected using *get_subsamples* (see III-D). The pairwise distances between the samples and the distances between the function values of the selected samples are calculated using function *pairwise_distances* and represented as M_x and M_y, respectively. Lastly, all the filter responses are calculated (see III-D, eq 1) and aggregated (see III-E) using multiple aggregation functions to form problem-specific features. The full Python implementation of the code is available at https://repo.ijs.si/gpetelin/random-convolutions.

Algorithm 1 Algorithm Pseudocode for Random Filter Feature Extraction

```
n_application ← 1000;
n_filters ← 100;
F ← initialize_filters(n_filters);
foreach problem p in P do
    S ← problem_samples(p);
    Ŝ ← scale_samples(S);
    foreach filter f in F do
        rs ← [];
        foreach i = 1 to n_application do
            x_anchor ← random_point();
            dim_adj ← f.radiu * √p.dim;
            subsamples ←
            get_subsamples(Ŝ, dim_adj, f.size, x_anchor);
            M_x, M_y ← pairwise_distances(subsamples);
            r ← ∑f.W_x ⊙ M_x + ∑f.W_y ⊙ M_y;
            rs.append(r);
        end
        p.features.append({max(rs), skew(rs), std(rs),
                           ppv(rs), mean(rs)});
    end
end
```

IV. RESULTS

In this section, we begin by presenting the experimental setup. We investigate the similarity of the suggested features

across various problem classes of the COCO benchmark and analyze how the proposed RFM features compare to the existing ELA features. Next, we demonstrate the performance of these features in the tasks of problem classification, high-level property prediction, and algorithm selection. Finally, we provide some additional visualizations and explanations on how features behave.

A. EXPERIMENTAL SETUP

To test the feature extraction method, the COCO benchmark suit is used. Our study utilized a total of 2400 problem instances from the 24 problem classes included in the COCO benchmark suite, with 100 instances used from each class. In all of the experiments, we use 100 filters unless otherwise specified. Feature extraction is performed with $200 \times D$ samples (similar to [16]). That means that for problems of dimensions 2D, 3D, 5D, and 10D, we used 400, 600, 1000, and 2000 samples, respectively. All the samples were obtained with Latin Hypercube sampling. This holds true for our proposed feature extraction approach as well as approaches used to obtain ELA features. The other two important parameters are the filter size and the radius for obtaining samples. When initializing random filters, filter sizes are uniformly sampled to be between 3 and 14. Such filter size range was selected since those are common sizes in other domains such as computer vision [54]. The radius of the filter was sampled to be between 0 and 0.5. Note that this radius is adjusted for the dimension of the problem as described in III-A. Each filter is applied at 5000 randomly selected anchor points unless otherwise specified. 5000 filter response values are summarised with five aggregate functions to transform 5000 numerical values into 5 features. All the missing values that are obtained during the feature creation are set to zero. We utilized various software tools for our study, including the *numpy* library [55] for feature extraction and the *flacco* library [26] for obtaining ELA features. The classification/regression was carried out using the *scikit-learn* library [56].

B. FEATURE SIMILARITY

Features extracted with the RFM technique are often numerous and highly similar. This subsection provides some insight into the relationships between features. First, we investigate the similarity between individual features shown in Figure 4 for 2D and 10D problems. The displayed features are generated by 100 randomly initialized filters and aggregated using 5 aggregate functions, resulting in a set of 500 features. To analyze the relationships between these features, cosine similarity is calculated, with each feature comprising 2400 numerical values (one for each of the 100 problem instances from the 24 problem classes). We see that some feature pairs have a high similarity while others do not. For example, filters aggregated with the *max* aggregation function have strong similarity with other *max*-aggregated filters. Similarly, high similarities are obtained between features aggregated with the *ppv* and *std* aggregate functions. On the other hand, features obtained by using the *mean* and *skew* aggregate functions are not that similar to each other or with any other features. The results indicate that the obtained features are likely to contain a substantial amount of redundant information, given their high cosine similarity. It also suggests that employing various aggregation functions might be advantageous, as they extract diverse information from the filter responses.

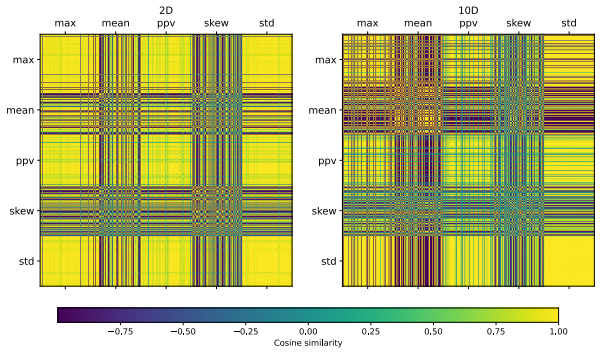


FIGURE 4. Cosine similarity between the 500 extracted features computed on 2400 2D and 10D COCO problem instances.

C. PROBLEM INSTANCE SIMILARITY

Similarly to the previous example, where features are compared using all problem instances, it is possible to compare problem instances between themselves using all RFM feature values. Figure 5 shows the cosine similarity between the vector representations of 2400 problem instances for 2D and 10D problems. Although most features carry only a low amount of information, interesting patterns are still revealed. Focusing first on the 2D problem class 5, we can observe a high similarity between instances of this class, which would indicate that instances belonging to this class are easily identifiable with the proposed features. On the other hand, there are 2D problem instances where it is harder to visually determine to what class they belong. Such are the classes 22 and 23 where their instances have a high similarity score between themselves. However, for 10D problems, instances have a much higher similarity between themselves and it is therefore harder to visually differentiate between them. Please be aware that while cosine similarity shows high values and distinguishing between instances visually might be difficult, it's still feasible to identify a subset of features that offer improved separation between different problem classes.

D. CORRELATION WITH ELA FEATURES

In order to delve deeper into the characteristics of the proposed features, we compare them with various groups of ELA features (a detailed description of each ELA group is available in [26]). To evaluate the similarity between each pair of features, we employ the Pearson correlation. The resulting correlation between all the features is displayed in Figure 6, based on an analysis of 2400 problem instances from COCO.

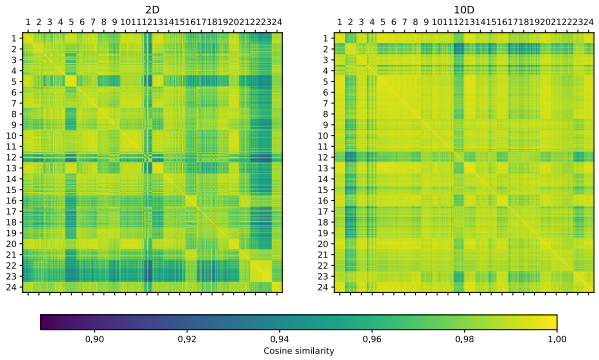


FIGURE 5. Cosine similarity between 2400 problem instances belonging to 24 problem classes for 2D and 10D problems.

For the sake of simplicity in visualization, only 20 randomly selected RFM features were used. We see that the proposed features are somewhat correlated with other ELA feature groups. The strongest correlation (positive or negative) can be observed with the ELA Distribution Features [8], ELA Levelset Features, and Dispersion Features [21]. It should be noted that there are significant variations in the degree of correlation between individual RFM features and the existing ELA feature groups. While certain features show a strong correlation with ELA features, others do not. This suggests that the proposed features (perhaps because of their large quantity) can capture some of the knowledge contained within the existing ELA feature groups. This also raises the question of why random filter mapping features exhibit similarities to the existing ELA feature groups, and whether some existing feature groups could be represented using the proposed filters.

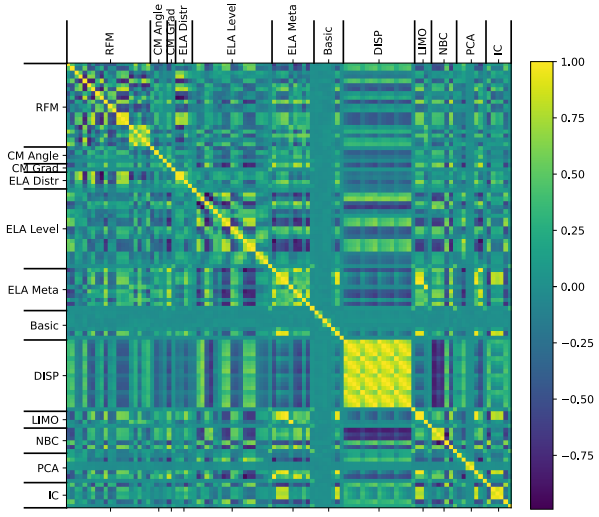


FIGURE 6. Pearson correlation between ELA features and newly proposed RFM features computed across 2400 2D COCO problem instances belonging to 24 distinct problem classes.

E. RANDOM FILTER RESPONSES

In previous sections, we show that random filters can be used to differentiate between problem instances. However, this does not explain how such filters work and what information they capture. Therefore, we investigate the filter responses across the entire domain of the objective function. We only visualize responses of a 2D problem since higher dimensional problems are harder to visualize. Figure 7 shows three objective functions and responses obtained with two randomly selected filters (in this case filters initialized with seeds 396 and 512). While interpreting the filter responses is challenging, one possible explanation of what exactly is learned by individual filters is the following. Filters work by looking at points that are close together and producing a response based on distances between them. In that setting, filters with small radii are sensitive to the "noisy" functions such as the third function in Figure 7. Conversely, the bigger filter could be susceptible to detecting larger local optima. Some filters may be responsive to gradients and exhibit strong responses when applied to samples with a particular gradient. As a result, when these filter responses are combined, the optimization problem can be described as having noise, optima, and gradients.

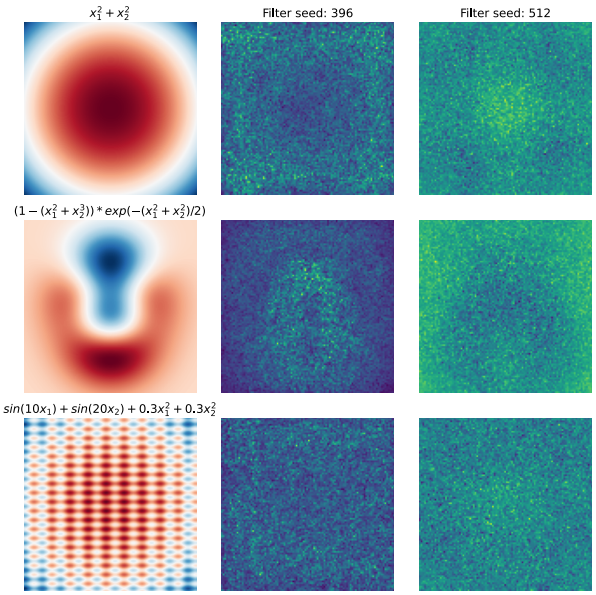


FIGURE 7. Three 2D objective functions and their responses obtained with two randomly selected filters. The first column shows the objective function, while the second and the third columns show the filter responses. In this case, filters initialized with seeds 396 and 512 were selected for extracting responses.

F. PROBLEM CLASSIFICATION

In this section, we assess the efficacy of the proposed features in the classification of problem instances into the 24 problem classes from the COCO benchmark. This evaluation is of significance since it not only confirms that similar functions are represented similarly using proposed features, but also sheds light on the underlying structure of optimization problems. If similar optimization problems have comparable feature representations, it indicates that certain problem structures might be more prevalent and identifiable.

Consequently, it may aid in the creation of more efficient optimization techniques that can take advantage of these shared structures. In our study, we chose to use the random forest [57] algorithm for classification, as it is well-suited for tabular data [58]. The dataset (2400 instances) is split into 90% training and 10% testing sets, using stratified random splitting to ensure balanced class distribution. This process is repeated 30 times for improved performance and model robustness.

1) COMPARISON TO ELA FEATURES

First, we aim to investigate the expressiveness of the proposed features for problem classification in comparison to ELA feature groups. In our study, we use the ELA feature groups *LIMO*, *IC*, *CM Grad*, *PCA*, *ELA Level*, *CM Angle*, *ELA Distr*, *ELA Meta*, *NBC* and *DISP*. All of these feature groups are configured with their default hyperparameters as specified in the *flacco* library. The random forest model is then trained to predict one of the 24 problem classes in the COCO benchmark suite, where the input features are: *i)* the features from each ELA feature group individually, *ii)* concatenated features from all ELA feature groups and *iii)* the proposed RFM features. Figure 8 depicts the classification accuracy obtained for each problem dimension (indicated in different color), using each of the aforementioned input features. The y-axis represents the features used for training the random forest model, where *ELA ALL* represents all concatenated ELA features, *RFM* represents the proposed RFM features, while all others are the individual feature groups.

Our findings suggest that the proposed features perform well, particularly for low-dimensional problems when compared to other existing features. For 2D problems, the proposed features are the third most powerful features when discriminating between problem instances, performing worse than only the Information Content (*IC*) feature group and *ELA Meta*. On the other hand, when working with higher-dimensional problems, the performance drops substantially to slightly below 60%. A possible explanation for this is that the proposed features are inherently limited to problems with lower dimensions or (as we explore in subsections IV-F5 and IV-F4) that for higher dimensional problems, filters often fail to produce responses and that higher dimensions are more sensitive to the selection of hyperparameters of the filters.

Investigating which problem instances are most frequently misclassified is also an aspect to consider. The confusion matrix of the proposed feature extraction technique is shown in Figure 9 for 2D and 10D problems. It is worth noting that the figure was compiled by averaging performance data over 30 runs to ensure more reliable outcomes. We see that most of the problem instances are classified correctly with a few exceptions. The most commonly misclassified problem instances are the ones belonging to problem classes 6 (Attractive-Sector Function), 7 (Step Ellipsoidal Function), 10 (Ellipsoidal Function), 11 (Discus Function), 15 (Rastrigin Function), 17 (Schaffers F7 Function) and 18 (moderately

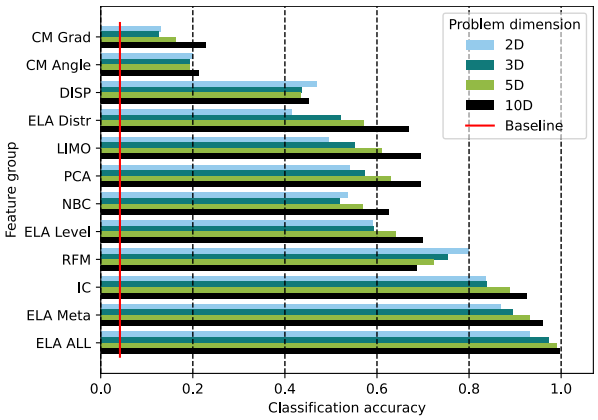


FIGURE 8. Comparison of the classification performance of the proposed RFM, individual ELA feature groups and a concatenation of all ELA feature groups (denoted as *ELA ALL*). The red line shows a classification accuracy of $\frac{1}{24}$ that would be obtained if the problem class was determined randomly.

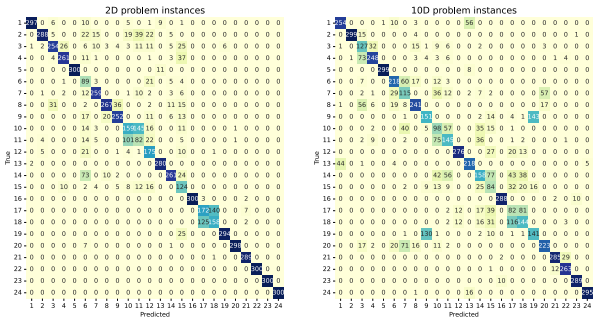


FIGURE 9. Confusion matrix when classifying problem instances into one of the 24 2D/10D problem classes with proposed random convolution features. Values are aggregated over all 30 train/test splits.

ill-conditioned function 17) where classification accuracy is below 60% for 2D problems. This means that some problems, when represented in high-dimensional feature space, are mapped close together and thus hard to differentiate for machine learning algorithms. Especially interesting are functions such as 17 and 18 (Schaffers F7 Function and its moderately ill-conditioned variant) where differentiating between them is almost impossible.

2) PREDICTIVE POWER OF A SINGLE FILTER

When filters are applied to get a response vector, a question that is immediately raised is if some filters in combination with an aggregate function inherently carry more information than others when performing classification. In other words, are there combinations of filters and aggregate functions that can outperform all others? To better demonstrate the information captured by the individual filters in combination with an aggregate function, Figure 10 plots the classification accuracy of 1000 filters where responses of each filter are summarized by one of the five proposed aggregate functions (5000 features in total). In this setup, each problem instance is represented only by a single numerical feature obtained with

one filter and aggregate function combination that is mapped to one of the 24 classes. Problem instances are then split into train and test sets used to create a model and evaluate the performance of the filter and aggregate function pair. We can see that almost all filter/aggregate function pairs can capture some information and that their representation outperforms the random class classifier for problems of all dimensions. However, most of the filter/aggregate function combinations perform substantially better than the random classifier with accuracies between 5% and 20%. Furthermore, from the plot, it is evident that certain infrequent combinations of filters and aggregate functions can attain accuracies exceeding 25%. Conversely, some filters combined with a single aggregate function exhibit poor performance, with an accuracy of only $\frac{1}{24}$, which is comparable to that of the random classifier. For the 2D, 3D, 5D and 10D problems, the probability of filter/aggregate function pair achieving an accuracy of less than $\frac{1}{24}$ is 2.9%, 3.4%, 6.0% and 13.8% respectively. The last plot in Figure 10 also plots the accuracy of the model when features are not obtained by random mappings using the proposed filters but are instead randomly sampled from a normal distribution $N(0, 1)$. In other words, the features with which the classifiers in the bottom plot are trained, are random features that do not carry any information. As one might expect the accuracy of a model trained on such features is similar to the random classifier with 58.5% of the models performing worse than random. This serves as a demonstration and additional check that shows that the features produced with the RFM technique are not the same as random features. In general, this analysis of the classification accuracy of individual filter/aggregate function pairs shows that most combinations of filters and aggregate functions will produce features that are of low quality but that some rare combinations of randomly initialized filters and aggregate functions can be used to differentiate between previously unseen problem instances with relatively good accuracy.

3) AGGREGATE FUNCTION COMPARISON

When a filter is applied with different anchor points and filter responses are computed, it is important to aggregate them into a single value. The choice of an aggregate function can significantly affect the quality of the obtained representations. Here we explore how the choice of an aggregate function influences the representations that are obtained and the consequent classification accuracies. In IV-F2 we showed that some filter/aggregate function pairs can achieve accuracy up to 30%. This however neglects one important fact, which is the influence of the aggregate function used to obtain the representation. Figure 11 shows classification accuracies of 1000 filters same as in Figure 10 but this time combined by the aggregate function used. The figures on the right are normalized stacked distributions of the left figures to better show which aggregate functions achieve good accuracy. We see that some aggregate functions generally produce better classification accuracy. For example, the aggregate functions *skew* and *max* rarely achieve classification accuracy above 20% with any of the 1000 filters. On the other hand, the aggregate functions *mean*, *std* and *ppv* most often achieve higher accuracies, meaning that the representations they build are better for the task of problem classification. The accuracies above 30% are almost exclusively achieved with the *mean* and *ppv* aggregate functions.

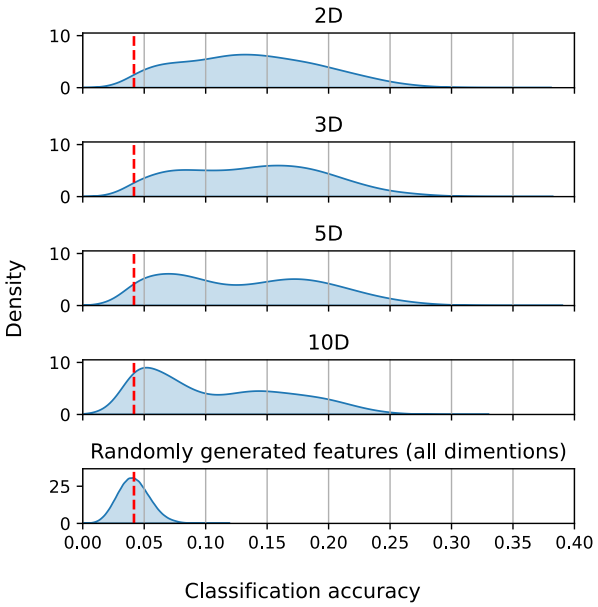


FIGURE 10. Classification accuracy of a single filter aggregated with one of the five aggregate functions for problems of different dimensions. The red line indicates the accuracy of the random classifier at $\frac{1}{24}$. The vertical axis shows the kernel density estimate.

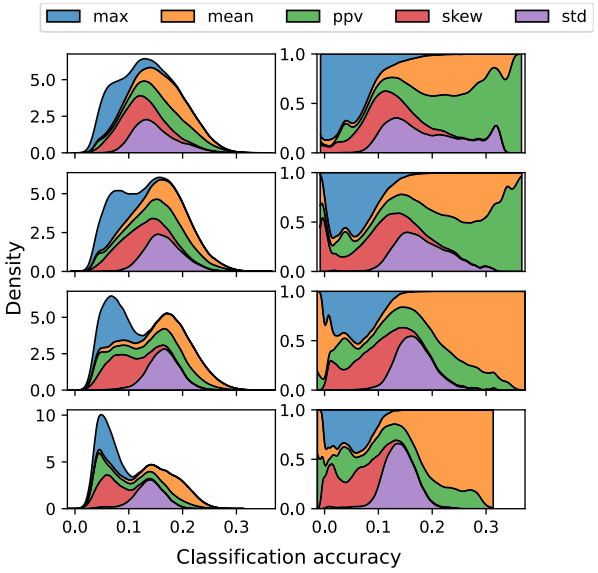


FIGURE 11. Classification accuracy of a single filter aggregated with one of the five different aggregate functions.

Previously we measured the performance of a single filter aggregated with a single aggregate function, meaning that problem instance classification was performed based on a single numerical value. In Figure 12 we show the

classification accuracy with 100 features. In this case, each feature is obtained by using one of five aggregate functions that aggregate values for 100 randomly initialized filters. Substantial variations in classification accuracies can be observed when employing different aggregation functions. Similarly to before, *ppv* is the best-performing aggregate function across all problem dimensions followed by the *mean* and *std* aggregate functions. The worst-performing aggregate functions are again the *skew* and *max* functions. This further demonstrates the importance of using aggregate functions that can capture useful information.

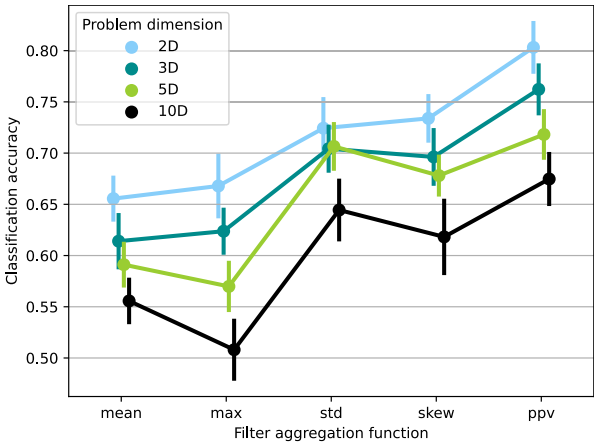


FIGURE 12. Classification accuracy of 100 filters aggregated with one of the five different aggregate functions.

4) FILTER HYPERPARAMETER IMPORTANCE

When constructing filters, one has to select the filter radius (maximum distance from an anchor point) and the filter size (number of sample points used to compute filter response). The selection of these parameters can have a large effect on the quality of the constructed features. Figure 13 shows the mean classification accuracy for filters with different radii and sizes aggregated with the *ppv* aggregate function. Each cell shows the average classification accuracy obtained with a different filter size (marked in rows) and a different filter radius(marked in columns). This was achieved by averaging classification accuracies of 100 filters for each radius/size pair. The figure thus demonstrates an important fact about how to most optimally set filter radii and sizes. For example, for 2D problems, the most informative radii and filter size combinations are filters with radii between 0.1 and 0.2 and sizes between 5 and 10. Such filters can on average achieve classification accuracies up to 20%. For 2D problems, the worst-performing filters are ones initialized with small radii and large sizes. The cause of this is that when the anchor point is selected there might not be enough samples around for the filter to be used, thus producing features with missing values. This is even more evident for higher-dimensional problems where filters with small radii carry almost no information. For 10D problems, filters with radii between 0.0 and 0.1 achieve classification accuracy no better than the random classifier. A conclusion can be drawn from this that the selection of filter parameters can have a large impact on distinguishing between problem instances. Although due to computational cost, we only test filters with a maximum size of 14, results may suggest that for higher dimensional problems larger filters could also be useful.

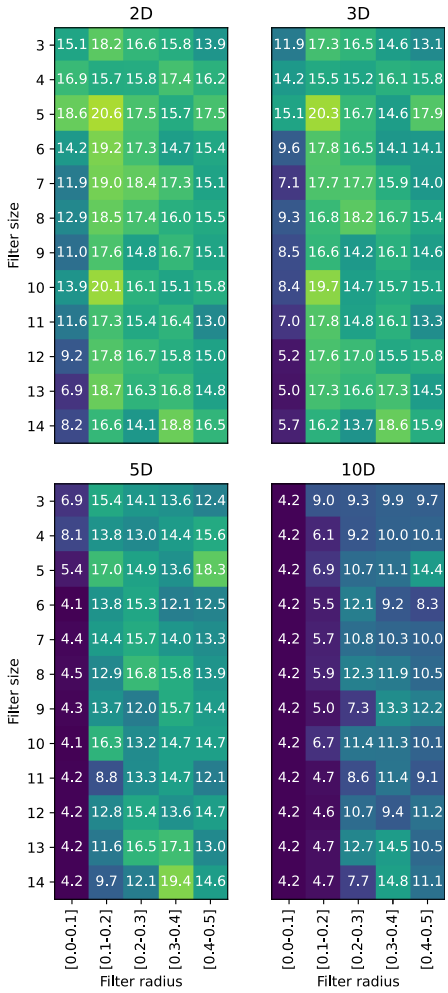
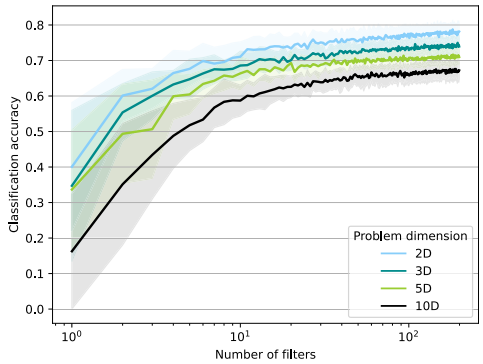


FIGURE 13. Mean classification accuracy for filters with different size and radius parameters aggregated with *ppv* aggregate function.

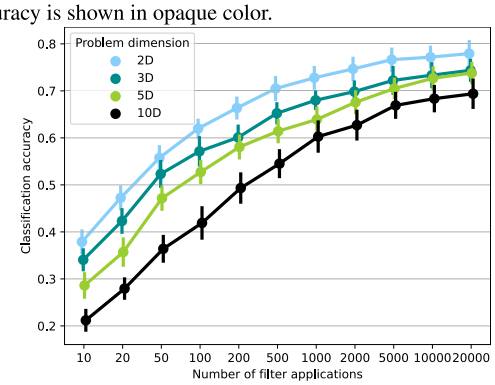
5) NUMBER OF FILTERS AND NUMBER OF FILTER APPLICATIONS

One specifically interesting to investigate aspect is the number of filters needed to achieve a certain classification accuracy. Figure 14a shows the number of filters necessary to achieve a certain classification accuracy. Error bars show the standard deviation in classification accuracy when repeating the process 100 times with different random initialization of the filters. We see that for problems of all dimensions, a relatively low number of filters (30 filters, making up a total of 150 features) is needed to reach a point where adding additional filters becomes less and less effective. For problems of dimensions less than five, 30 filters will

achieve an accuracy of above 60%. For the 10D problems, the accuracy achieved with 30 filters is somewhat lower, around 50%. This means that even a small set of randomly initialized filters can extract enough useful information to correctly classify most problem instances. Although, if computational power is not limited, having more filters are always better.



(a) Number of filters and the obtained classification accuracy with 5000 filter applications for each filter. The standard deviation in accuracy is shown in opaque color.



(b) Classification accuracy using 100 filters where each filter is applied a different number of times. The standard deviation in accuracy is shown with error bars.

FIGURE 14. Number of filters and number of filter applications necessary to achieve particular classification accuracy.

Each filter that is initialized has to be applied multiple times with different randomly selected anchor points. The number of anchor points determines how many values are aggregated with one of the five aggregate functions. In Figure 14b, we illustrate the relationship between the number of times filters are applied and the resulting classification accuracy. We observe that the accuracy is proportional to the number of filter applications for problems of all dimensions. For instance, when 100 filters are applied only 10 times, the accuracy for 2D problems is less than 40%, and for 10D problems, it is slightly above 20%. However, as the number of filter applications increases, we notice that the accuracy improvement becomes diminishing. For example for 2D problems, doubling the number of applications from 500 to 1000 only increases accuracy by around 3%. This shows that there is a clear trade-off between the number of times filters are applied before values are aggregated and the quality of the information that is extracted. It also demonstrates that if computational power is not limited, one should use as many filter applications as possible.

G. HIGH-LEVEL PROPERTY PREDICTION

The significance of high-level properties, such as multimodality, global structure, and funnel structure, cannot be understated in the context of automated algorithm selection, as they frequently determine the complexity of an optimization problem. It is therefore common to try to determine the high-level properties of a particular objective function in practice. In this section, we evaluate how well the proposed features are able to capture the high-level properties of the problem instances in the COCO benchmark suite. Table 1 lists the high-level properties of all the COCO problem instances used in this study.

TABLE 1. Characteristics of high-level properties for the 24 COCO functions as defined in [18]. When using leave-one-problem-out when predicting properties, functions marked with * are ignored due to being the only ones with specific properties.

BBOB problem class	Multimodal	Global structure	Funnel
1: Sphere	none	none	yes
2: Ellipsoidal separable	none	none	yes
3: Rastrigin separable	high	strong	yes
4: Büche-Rastrigin	high	strong	yes
5: Linear Slope	none	none	yes
6: Attractive Sector	none	none	yes
7: Step Ellipsoidal	none	none	yes
8: Rosenbrock	low	none	yes
9: Rosenbrock rotated	low	none	yes
10: Ellipsoidal high cond.	none	none	yes
11: Discus	none	none	yes
12: Bent Cigar	none	none	yes
13: Sharp Ridge	none	none	yes
14: Different Powers	none	none	yes
15: Rastrigin multimodal	high	strong	yes
16: Weierstrass	high	medium	none
17: Schaffer F7	high	medium	yes
18: Schaffer F7 mod. ill-cond	high	medium	yes
19: Griewank-Rosenbrock	high	strong	yes
20: Schwefel	medium	deceptive*	yes
21: Gallagher 101 Peaks	medium	none	none
22: Gallagher 21 Peaks	low	none	none
23: Katsuura	high	none	none
24: Lunacek bi-Rastrigin	high	weak*	yes

Our validation approach involves training a classifier using the leave-one-problem-out (LOPO) cross-validation strategy, where one problem class is designated for testing while the others are used for training. This process is repeated for all 24 problems in the dataset, resulting in the creation and evaluation of 24 models. Our approach differs from the one used in [18], where the model is trained on all 24 problem classes, and only a few instances from each class are used for testing. This validation strategy is much more challenging, as all instances belonging to a particular problem are moved into the testing set. We perform 30 repetitions of training the Random Forest classifier, similar to the approach used in the previous section on classification.

For each of the high-level properties, we build separate ML models. Therefore a model that is trained to determine the level of multimodality is tasked with predicting four levels *[none/low/medium/high]* of multimodality. Similarly, a model tasked with predicting the global structure has to predict one of three levels *[none/medium/strong]*. Note here that *weak* and *deceptive* global structures are excluded due to having only one problem class with those two values, making it infeasible to include them in the training and test data. Lastly, a model is also constructed to predict funnel structure where the targets are only two values *[none/yes]*. The accuracy is measured using the macro-averaged F1 score instead of simple classification accuracy, as described in the chapter on problem instance classification. The reason for not using classification accuracy is that the classes are imbalanced in this case.

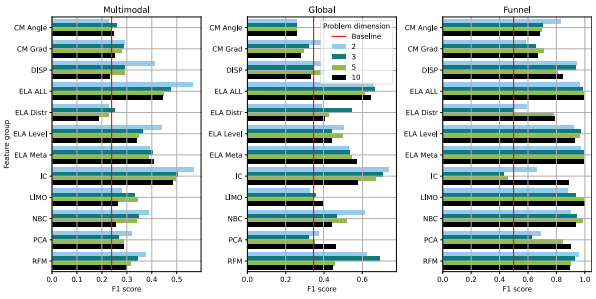


FIGURE 15. Macro-averaged F1 score obtained when predicting the high-level properties of an optimization function using the leave-one-problem-out validation. The red line shows a macro-averaged F1 score that would be obtained if the high-level property prediction was random.

Figure 15 shows the macro-averaged F1 score when predicting the levels of multimodality, global structure, or funnel structure with the proposed features and compares them to ELA features groups. The models in this instance underwent 30 iterations of training to achieve more reliable outcomes. The number of features and number of filter applications is the same as in section IV-F. The vertical red line on the graph shows the F1 score that would result from random predictions of high-level properties. Turning our attention to the concept of multimodality, the suggested RFM method outperforms the baseline but does not provide significant insights when determining the level of multimodality in the function, achieving an F1 score of around 0.35. In contrast, the best performing are the *IC* features which are comparatively better at capturing the multimodality level of an objective function achieving the F1 score of around 0.5. Next, we shift our focus to predicting the global structure. We see that in this setting, the proposed features achieve a relatively high F1 score of between 0.43 and 0.7, and are especially good at low-dimensional problems. However, the best feature group for detecting the global structure is still the *IC* feature group with an F1 score between 0.57 and 0.77.

Lastly, we focus on how features perform when detecting the funnel structure. In this case, most of the feature groups are able to identify the funnel structure. In this case, the RFM features are among the best-performing features achieving the F1 score of 0.9 while the best feature group archives F1 score of around 0.96. This demonstrates that RFM features can be effective at capturing some of the high-level properties.

H. ALGORITHM SELECTION

Arguably, the most crucial task in characterizing optimization functions using a feature set is facilitating algorithm selection (AS) through various machine learning techniques. Within the context of the COCO benchmark, two primary methods for algorithm selection and performance prediction during validation emerge prominently: ''leave-one-instance-out'' (LOIO) and ''leave-one-problem-out'' (LOPO), emphasized in [59] and [60]. In the LOPO validation approach, the process of automated algorithm selection involves excluding one COCO problem class from the training dataset to use as the test set. This results in a meta-model being trained on the remaining 23 problem classes. However, applying the LOPO validation to AS models presents difficulties due to the diverse characteristics of COCO problem classes, as highlighted in [60]. The prediction phase may introduce problem instances with previously unseen properties, significantly complicating the LOPO validation method. Conversely, the LOIO validation method offers a more forgiving environment for model training, as it incorporates all problem classes. This method involves training on certain instances from each problem class while setting aside others for testing. Consequently, the meta-model is likely to have encountered functions with similar characteristics during training, facilitating the selection of the most appropriate algorithm or the prediction of performance more effectively. To ensure fairness, we present both validation strategies. In our approach, we treat algorithm selection as a regression problem, with the meta-model predicting performance relative to other optimization algorithms. The meta-model and preprocessing steps we employ are consistent with those utilized throughout the paper.

In our research, we utilize an algorithm portfolio A that includes Genetic Algorithms (GA) [61], Differential Evolution (DE) [62], Particle Swarm Optimization (PSO) [63], Evolution Strategies (ES) [64], and CMA-ES [65]. These algorithms were selected as they are commonly implemented in most of the optimization frameworks and have been extensively used in practice. All algorithms are being executed for 1000D function evaluations per problem instance. To evaluate the accuracy of our predictions, we employ the following metric as used in [66].

Initially, we determine the relative performance of each algorithm per execution by adjusting the solution values to a common scale. This adjustment ensures that the algorithm yielding the optimal solution is awarded a score of 0, and the algorithm with the least favorable solution is assigned a score of 1. We then compute the average of these scores across 30 independent trials to derive more stable final scores.

Consequently, if an algorithm consistently emerges as the best across all trials, it will receive a final score of 0. Conversely, if it consistently ranks as the least effective in every trial, it will be given a final score of 1. The following equation provides a more detailed explanation of this scoring method:

$$ns_{a,p,r} = \frac{y_{a,p,r} - \min_{a \in A} y_{a,p,r}}{\max_{a \in A} y_{a,p,r} - \min_{a \in A} y_{a,p,r}} \quad (2)$$

$$s_{a,p} = \frac{1}{R} \sum_{r \in |R|} ns_{a,p,r} \quad (3)$$

In this equation, the optimization algorithm is denoted by a with p being a problem instance that the optimization algorithm is executed on. Due to stochasticity, algorithms from algorithm portfolio $|A|$ are executed numerous times with r being the particular run of an algorithm and $|R|$ being the total number of runs. The $y_{a,p,r}$ thus denotes the best solution discovered by the algorithm a on problem p in run r. A normalized score for an algorithm for a particular run is represented by $ns_{a,p,r}$. The obtained score is then averaged into $s_{a,p}$. The prediction error of the meta-model, when selecting an algorithm for a specific problem, is determined by subtracting the lowest actual score $\min \min_{a \in A} s_{a,p}$ from the normalized score $s_{a,p}$ of the algorithm predicted to be the best.

To illustrate with a specific example, suppose we are working with three algorithms: DE, GA, and CMA-ES. Given an objective function, these algorithms will find solutions with the following values: [3.5, 4.2, 3.4] in the first run and [3.4, 4.0, 3.1] in the second run. The two runs are independently scaled using the previously mentioned equation to obtain scaled performances. In our case, when the two vectors are scaled, we obtain performances of [0.125, 1, 0] and [0.333, 1, 0]. Following this, the average of the two runs is calculated to obtain the mean relative performance of the algorithms on the problem, which is [0.229, 1.000, 0.000]. In this scenario, selecting CMA-ES as the best algorithm results in an error of 0, indicating a perfect match. On the other hand, choosing DE as the top algorithm leads to an error of 0.229, reflecting its relative inferiority to the best algorithm in terms of performance. When loss is formulated in this manner, if a non-optimal algorithm is selected, the penalty is not as severe if the performance is still similar to that of the best algorithm.

To evaluate the feature sets, we construct three distinct algorithm selection meta-models, each being a multi-output regression model tasked with predicting the relative performance of algorithms within a portfolio, subsequently selecting the algorithm deemed most efficient. Precision is assessed using Eq 3. The Single Best Solver (SBS) operates as a rudimentary regression model, calculating output as the average of relative scores from training, devoid of feature consideration, akin to selecting the best algorithm from the training set for any given problem regardless of its characteristics. The RFM employs a multi-output random forest regressor, utilizing random filter mappings for training, with each problem instance depicted through an RFM vector translated into relative algorithm scores. The ELA meta-model mirrors the RFM approach, except it represents problems using ELA features rather than RFM, aiming for a nuanced understanding of algorithm performance across varied problem landscapes.

As discussed in papers such as [66], we use this error metric to alternatives such as classification accuracy due to its targeted approach to misclassification. This metric imposes a penalty for selecting an incorrect algorithm, with the severity of the penalty depending on how the chosen algorithm's performance compares to others. Specifically, if two or more algorithms have nearly the same performance with only a slight difference in their outcomes, choosing the slightly lesser algorithm results in a minor penalty. This approach indicates a more lenient strategy, tolerating selections where the chosen algorithm's performances are close to that of the best-performing algorithm.

Figure 16 displays the performance of models using different feature sets with both the LOIO and LOPO validation strategies. The lower the score, the better the AS meta-model is at selecting an algorithm. Focusing first on the LOIO validation, we observe that for all dimensions, the proposed RFM features substantially outperform the baseline SBS meta-model, with the error being approximately half of the error of the SBS. When compared with the meta-model using ELA features, the error is slightly lower for all dimensions with the RFM feature. However, with the LOPO validation, the error of all three meta-models increases compared to LOIO. This is expected, as LOPO is much more challenging than LOIO. With LOPO, RFM features only slightly outperform the SBS meta-model. A similar situation is true for the ELA features, where the performance is not substantially better than that of an SBS meta-model.

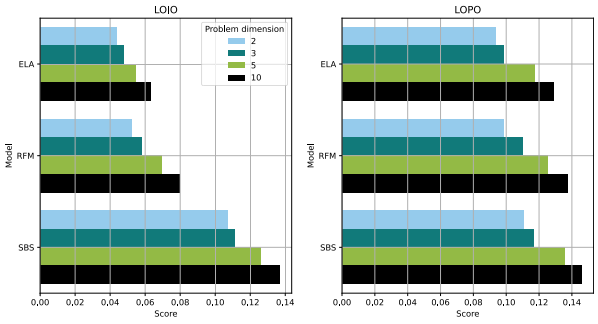


FIGURE 16. The accuracy of the Single Best Solver (SBS), along with the ELA and RFM feature-based models, was evaluated using the LOIO and LOPO validation strategies. The score (lower is better) is computed using Equation 3.

The observations suggest that the proposed RFM features are effective in constructing characteristics that facilitate the selection of algorithms, particularly when there is a similarity between training and test instances. Additionally, it is noted that RFM features exhibit behavior similar to that of ELA features, though they score marginally lower

on both the LIPO and LOIO tasks. However, it's important to highlight that both sets of features face challenges in accurately selecting the most suitable algorithm for functions that are completely out of distribution, a finding that aligns with existing research [13], [60] in the field.

I. EXPLORING GENERATED FEATURE VALUES

It is crucial to not only consider the application of constructing random mappings with filters for classification purposes but also to delve into the underlying reasons for its expressiveness. In this subsection, we explore the features created by the filters and how "close" in latent space are the instances belonging to the same problem class. Please note that the selection of these specific filters and problem classes was made to illustrate the working of the approach. Other filters, classes, or aggregate function combinations might not produce such nice separable plots. Figure 17 shows how problem instances are distributed when features are created with the proposed technique. Here, each problem instance is represented with one feature obtained by an *std*-aggregated filter initialized with seed 703 (radius 0.165/size 3). To enhance the clarity of visualizations, only problem instances belonging to classes 1, 8, 11, 16, and 22 are displayed. We can observe that the selected filter aggregated with the *std* aggregate function can separate some of the instances belonging to different problem classes. For example, we can see that instances from classes 1 and 22 can be separated while instances belonging to classes 1 and 11 and grouped closer together and thus harder to differentiate due to overlapping distributions. This essentially shows that with the proposed approach it is often the case that problem instances can be mapped into higher-dimensional space where they can often be easily separated.

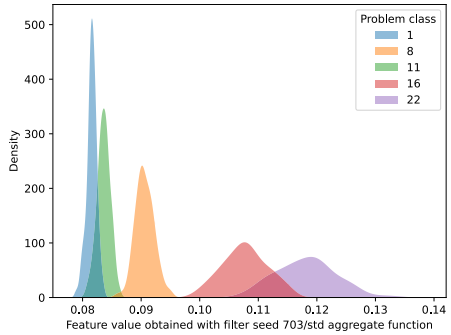


FIGURE 17. Kernel density estimate representing the distribution of feature values for 500 problem instances across 5 different classes. Filter initialized with seed 703 was in this case aggregated with the *std* aggregate function.

Using the same filter as before, this time we plot mappings obtained with multiple aggregate functions. Figure 18 shows the distribution of problem instance feature values for 5 different problem classes. Essentially, every subplot illustrates the data distribution generated by a particular filter (seed 703) and a combination of two aggregated functions. For instance, the third column of the first row depicts the distribution pattern of problem instances resulting from the *mean* and *max* aggregation of values obtained from filter 703. It can be observed that even a single filter using multiple aggregate functions can be beneficial for differentiating problem classes. For example, using *mean* and *max* aggregate functions with the filter initialized with seed 703, problem instances from problem classes 1, 8, and 11 pose a challenge in distinguishing them as they are all mapped to a common feature space. On the other hand, using *mean* and *std* aggregate functions can separate problem instances belonging to classes 1 and 8 while still struggling to distinguish between problem instances of classes 1 and 11. This demonstrates that using diverse aggregate functions can often be beneficial when constructing features.

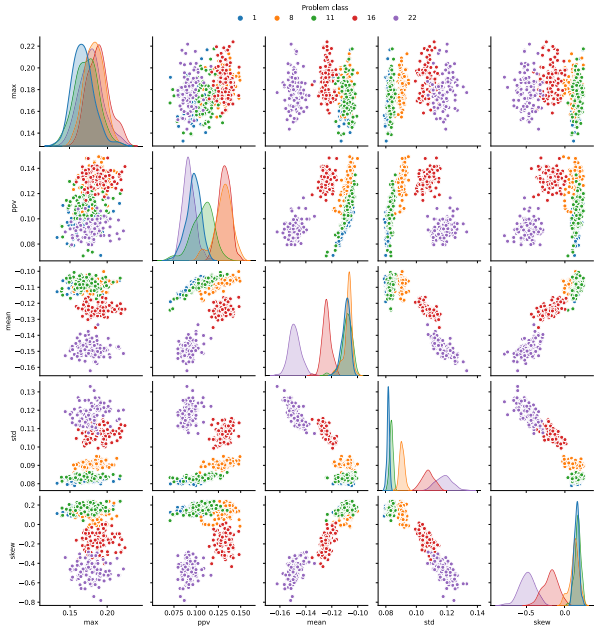


FIGURE 18. Response of a filter that achieves the highest classification accuracy aggregated by all five aggregate functions. Filter responses are displayed for 100 problem instances from five different problem classes.

V. LIMITATIONS AND DISCUSSION

This section lists some of the limitations of the proposed feature extraction methodology. While we believe we are the first to propose this particular approach to feature extraction, it is not without its limitations. One of the biggest downsides is that most of the generated features are "weak" features that (on their own) do not carry a lot of information. Most features individually achieve poor classification accuracy. Only in combination with other features does the classification accuracy increase. This is in contrast to ELA features where some individual features carry a large amount of information and can achieve high classification accuracy on their own. A closely linked problem is also that the number of generated features might be extremely large. This may pose a problem for machine learning algorithms that are sensitive to noisy features. Furthermore, generating a large number of features incurs high computational costs. A lot

of computational resources might be wasted computing filter responses that in the end turn out to be uninformative. The same problem plagued algorithms in the time series domain such as ROCKET, where the original paper [36] proposes 10,000 filters and 20,000 features but was later improved in subsequent papers. For higher-dimensional problems (e.g., 20D, 50D, and 100D problems) commonly studied as part of the COCO benchmark, the proposed methods often require a large number of filters. This is the primary reason why the paper includes only lower dimensions, due to the high computational cost.

The proposed filters incorporate some inductive biases that might limit their usability, especially for high-dimensional problems. For example, the methodology assumes that the feature construction should only be based on the Euclidian distance between samples. But it is well known that for higher-dimensions Euclidian distance has severe limitations [67], [68]). Moreover at most 14 samples are considered when calculating filter response. Such assumptions may result in inferior features. Unfortunately, optimal filter construction for high-dimensional point clouds is still a relatively new and largely unexplored area. To the best of our knowledge, there are no filters proposed specifically for the domain of optimization problems. Moreover, in contrast to feature groups such as ELA, the features that are obtained are not explainable. A single feature does not carry any meaning and is hard to interpret for humans. On the other hand, such features can be extremely powerful for machine learning models. The proposed RFM approach should be considered as a feature extraction framework that can be extended/improved with new and modified filters in combination with new aggregation functions. Hand-crafted filters could be created that are sensitive to specific properties of an objective function (e.g. filter that is sensitive to particular noise or funnel structure) then objective functions could be potentially described as a composite of smaller elements that filters are able to detect and transform into features.

Lastly, the COCO function classification is in itself a metric that is not perfectly aligned with that kind of feature and would be ideal when performing algorithm selection/configuration meaning that a feature extraction method that can differentiate between problem classes might not provide useful representation for other tasks. Our aim with the suggested RFM features is to establish a technique that transforms comparable objective functions into a feature space, ensuring their proximity. This approach aims to create a representation that might be valuable for various other tasks.

VI. CONCLUSION AND FUTURE WORK

Characterizing optimization problems with numerical features is a precursor for building automated algorithm selection/configuration and can greatly help in deepening the understanding of the properties of optimization problems. We present a new technique for feature extraction in numerical optimization problems. This approach merges conventional practices of designing domain-specific filters with random mapping-based methods. We show that by using domain-tailored filters with randomly initialized weights, one can extract ''weak'' features that on their own carry little information but when combined can create useful numerical representations of optimization functions. Our proposed methodology has demonstrated the ability to differentiate COCO problem instances with accuracy comparable to some of the better-performing ELA feature groups. We delved into the hyperparameters of the approach to gain a better comprehension of the feature extraction methodology. Furthermore, we were able to offer a more comprehensive understanding of our proposed method by visualizing the behavior of newly designed domain-specific filters and examining the resulting features. Lastly, we also explore how well the proposed approaches predict the high-level properties of the objective function where we show that the features are relatively good at detecting global/funnel structure but struggle to accurately predict the level of multimodality. We believe that the proposed approach opens up a new way of constructing features that may help in capturing knowledge that other hand-crafted approaches might miss. Additionally, the proposed filters provide an opportunity to conduct end-to-end learning without the construction of intermediate feature representations.

The immediate future steps are to further investigate how to best construct domain-tailored filters that could potentially extract more informative features. In this paper, we construct task-agnostic features but it would also be interesting to investigate updating the filter weights by using gradient-based methods. Such filters could potentially reveal unique structures in optimization problems that randomly initialized filters might miss. Finally, we demonstrate that the features can distinguish between different problem instances and that they are able to identify high-level properties. The next step would be to investigate whether the suggested features are viable for algorithm selection and configuration.

REFERENCES

[1] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, ''Automated algorithm selection: Survey and perspectives,'' *Evol. Comput.*, vol. 27, no. 1, pp. 3–45, Mar. 2019.

[2] N. Belkhir, J. Dréo, P. Savéant, and M. Schoenauer, ''Per instance algorithm configuration of CMA-ES with limited budget,'' in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2017, pp. 681–688.

[3] U. Skvorc, T. Eftimov, and P. Korošec, ''Understanding the problem space in single-objective numerical optimization using exploratory landscape analysis,'' *Appl. Soft Comput.*, vol. 90, May 2020, Art. no. 106138.

[4] G. Cenikj, R. D. Lang, A. P. Engelbrecht, C. Doerr, P. Korosec, and T. Eftimov, ''SELECTOR: Selecting a representative benchmark suite for reproducible statistical comparison,'' in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2022, pp. 620–629.

[5] M. A. Muñoz and K. Smith-Miles, ''Generating new space-filling test instances for continuous black-box optimization,'' *Evol. Comput.*, vol. 28, no. 3, pp. 379–404, Sep. 2020.

[6] K. M. Malan and A. P. Engelbrecht, ''A survey of techniques for characterising fitness landscapes and some possible ways forward,'' *Inf. Sci.*, vol. 241, pp. 148–163, Aug. 2013.

[7] K. M. Malan, ''A survey of advances in landscape analysis for optimisation,'' *Algorithms*, vol. 14, no. 2, p. 40, Jan. 2021.

[8] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph, "Exploratory landscape analysis," in *Proc. 13th Annu. Conf. Genetic Evol. Comput.*, 2011, pp. 829–836.

[9] Q. Renau, C. Doerr, J. Dreo, and B. Doerr, "Exploratory landscape analysis is strongly sensitive to the sampling strategy," in *Proc. 16th Int. Conf.*, Leiden, The Netherlands. Cham, Switzerland: Springer, Sep. 2020, pp. 139–153.

[10] U. Skvorc, T. Eftimov, and P. Korošec, "The effect of sampling methods on the invariance to function transformations when using exploratory landscape analysis," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2021, pp. 1139–1146.

[11] R. P. Prager and H. Trautmann, "Nullifying the inherent bias of non-invariant exploratory landscape analysis features," in *Proc. Int. Conf. Appl. Evol. Comput. (Part EvoStar)*. Cham, Switzerland: Springer, 2023, pp. 411–425.

[12] G. Cenikj, G. Petelin, and T. Eftimov, "A cross-benchmark examination of feature-based algorithm selector generalization in single-objective numerical optimization," *Swarm Evol. Comput.*, vol. 87, Jun. 2024, Art. no. 101534.

[13] G. Petelin and G. Cenikj, "How far out of distribution can we go with ELA features and still be able to rank algorithms?" in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2023, pp. 341–346.

[14] N. Hansen, S. Finck, R. Ros, and A. Auger, "Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions," INRIA, Rocquencourt, France, Tech. Rep., RR-6829, 2009.

[15] G. Petelin, G. Cenikj, and T. Eftimov, "TLA: Topological landscape analysis for single-objective continuous optimization problem instances," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2022, pp. 1698–1705.

[16] Q. Renau, J. Dréo, C. Doerr, and B. Doerr, "Towards explainable exploratory landscape analysis: Extreme feature selection for classifying BBOB functions," in *Proc. Int. Conf. Appl. Evol. Comput.* Cham, Switzerland: Springer, 2021, pp. 17–33.

[17] R. Morgan and M. Gallagher, "Analysing and characterising optimization problems using length scale," *Soft Comput.*, vol. 21, no. 7, pp. 1735–1752, Apr. 2017.

[18] M. V. Seiler, R. P. Prager, P. Kerschke, and H. Trautmann, "A collection of deep learning-based feature-free approaches for characterizing single-objective continuous fitness landscapes," 2022, *arXiv:2204.05752*.

[19] O. Mersmann, M. Preuss, and H. Trautmann, "Benchmarking evolutionary algorithms: Towards exploratory landscape analysis," in *Proc. Int. Conf. Parallel Problem Solving Nature*. Cham, Switzerland: Springer, 2010, pp. 73–82.

[20] G. Wu, R. Mallipeddi, and P. Suganthan, "Problem definitions and evaluation criteria for the cec 2017 competition and special session on constrained single objective real-parameter optimization," Comput. Intell. Lab., Zhengzhou Univ., Zhengzhou China Technical Report, Nanyang Technol. Univ., Singapore., Tech. Rep. 61780053, Oct. 2016.

[21] M. Lunacek and D. Whitley, "The dispersion metric and the CMA evolution strategy," in *Proc. 8th Annu. Conf. Genetic Evol. Comput.*, Jul. 2006, pp. 477–484.

[22] P. Kerschke, M. Preuss, C. Hernández, O. Schütze, J.-Q. Sun, C. Grimme, G. Rudolph, B. Bischl, and H. Trautmann, "Cell mapping techniques for exploratory landscape analysis," in *EVOLVE-A Bridge Between Probability, Set Oriented Numerics, and Evolutionary Computation V*. Berlin, Germany: Springer, 2014, pp. 115–131.

[23] M. Preuss, "Improved topological niching for real-valued global optimization," in *Applications of Evolutionary Computation*. Berlin, Germany: Springer, Apr. 2012, pp. 386–395.

[24] M. A. Muñoz, M. Kirley, and S. K. Halgamuge, "Exploratory landscape analysis of continuous space optimization problems using information content," *IEEE Trans. Evol. Comput.*, vol. 19, no. 1, pp. 74–87, Feb. 2015.

[25] P. Kerschke, M. Preuss, S. Wessing, and H. Trautmann, "Detecting funnel structures by means of exploratory landscape analysis," in *Proc. Annu. Conf. Genetic Evol. Comput.*, Jul. 2015, pp. 265–272.

[26] P. Kerschke and H. Trautmann, *Comprehensive Feature-Based Landscape Analysis of Continuous and Constrained Optimization Problems Using the R-Package Flacco*. Cham, Switzerland: Springer, 2019, pp. 93–123.

[27] Q. Renau, "Landscape-aware selection of metaheuristics for the optimization of radar networks," Ph.D. thesis, Inst. Polytechnique De Paris, Paris, France, 2022.

[28] J. Bossek, C. Doerr, and P. Kerschke, "Initial design strategies and their effects on sequential model-based optimization: An exploratory case study based on BBOB," in *Proc. Genetic Evol. Comput. Conf.*, Jun. 2020, pp. 778–786.

[29] R. D. Lang and A. P. Engelbrecht, "An exploratory landscape analysis-based benchmark suite," *Algorithms*, vol. 14, no. 3, p. 78, Feb. 2021.

[30] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Proc. Adv. Neural Inf. Process. Syst.*, 2007, pp. 1–8.

[31] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Proc. IEEE 12th Int. Conf. Comput. Vis.*, Sep. 2009, pp. 2146–2153.

[32] D. Cox and N. Pinto, "Beyond simple features: A large-scale feature search approach to unconstrained face recognition," in *Proc. IEEE Int. Conf. Autom. Face Gesture Recognit. (FG)*, Mar. 2011, pp. 8–15.

[33] A. M. Saxe, P. W. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Y. Ng, "On random weights and unsupervised feature learning," in *Proc. ICML*, 2011, p. 6.

[34] Z. Xu, D. Liu, J. Yang, C. Raffel, and M. Niethammer, "Robust and generalizable visual representation learning via random convolutions," 2020, *arXiv:2007.13003*.

[35] Y. Oh, M. Jeon, D. Ko, and H. J. Kim, "Randomly shuffled convolution for self-supervised representation learning," *Inf. Sci.*, vol. 623, pp. 206–219, Apr. 2023.

[36] A. Dempster, F. Petitjean, and G. I. Webb, "ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels," *Data Mining Knowl. Discovery*, vol. 34, no. 5, pp. 1454–1495, Sep. 2020.

[37] A. Dempster, D. F. Schmidt, and G. I. Webb, "MiniRocket: A very fast (almost) deterministic transform for time series classification," in *Proc. 27th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, Aug. 2021, pp. 248–257.

[38] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, and J. Cai, "Recent advances in convolutional neural networks," *Pattern Recognit.*, vol. 77, pp. 354–377, May 2018.

[39] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 12, pp. 6999–7019, Dec. 2022.

[40] R. Wang, Z. Li, J. Cao, T. Chen, and L. Wang, "Convolutional recurrent neural networks for text classification," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–6.

[41] Y. Luan and S. Lin, "Research on text classification based on CNN and LSTM," in *Proc. IEEE Int. Conf. Artif. Intell. Comput. Appl. (ICAICA)*, Mar. 2019, pp. 352–355.

[42] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1D convolutional neural networks and applications: A survey," *Mech. Syst. Signal Process.*, vol. 151, Apr. 2021, Art. no. 107398.

[43] K. Kashiparekh, J. Narwariya, P. Malhotra, L. Vig, and G. Shroff, "ConvTimeNet: A pre-trained deep convolutional neural network for time series classification," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–8.

[44] J. Baxter, "A model of inductive bias learning," *J. Artif. Intell. Res.*, vol. 12, pp. 149–198, Mar. 2000.

[45] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. Guibas, "KPConv: Flexible and deformable convolution for point clouds," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 6410–6419.

[46] Z. Zhang, B.-S. Hua, W. Chen, Y. Tian, and S.-K. Yeung, "Global context aware convolutions for 3D point cloud understanding," in *Proc. Int. Conf. 3D Vis. (3DV)*, Nov. 2020, pp. 210–219.

[47] H. Lei, N. Akhtar, and A. Mian, "Spherical kernel for efficient graph convolution on 3D point clouds," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 10, pp. 3664–3680, Oct. 2021.

[48] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "PointCNN: Convolution on X-transformed points," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Berlin, Germany: Curran Associates, 2018, pp. 1–11.

[49] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," 2021, *arXiv:2104.13478*.

[50] Y. He and S. Y. Yuen, "Black box algorithm selection by convolutional neural network," in *Proc. 6th Int. Conf.*, Siena, Italy. Berlin, Germany: Springer, Jul. 2020, pp. 264–280.

[51] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in Python," *IEEE Access*, vol. 8, pp. 89497–89509, 2020.
[52] S. Geisler, D. Zügner, and S. Günnemann, "Reliable graph neural networks via robust aggregation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 13272–13284.
[53] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, "Principal neighbourhood aggregation for graph nets," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 13260–13271.
[54] W. S. Ahmed and A. a. A. Karim, "The impact of filter size and number of filters on classification accuracy in CNN," in *Proc. Int. Conf. Comput. Sci. Softw. Eng. (CSASE)*, Apr. 2020, pp. 88–93.
[55] C. R. Harris et al., "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020.
[56] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011.
[57] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, Oct. 2001.
[58] P. Gijsbers, M. L. P. Bueno, S. Coors, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren, "AMLB: An AutoML benchmark," 2022, *arXiv:2207.12560*.
[59] A. Nikolikj, R. Trajanov, G. Cenikj, P. Korosec, and T. Eftimov, "Identifying minimal set of exploratory landscape analysis features for reliable algorithm performance prediction," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2022, pp. 1–8.
[60] R. Tanabe, "Benchmarking feature-based algorithm selection systems for black-box numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 26, no. 6, pp. 1321–1335, Dec. 2022.
[61] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: Past, present, and future," *Multimedia Tools Appl.*, vol. 80, no. 5, pp. 8091–8126, Feb. 2021.
[62] K. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach To Global Optimization*. Berlin, Germany: Springer, 2006.
[63] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. ICNN*, vol. 4, 1995, pp. 1942–1948.
[64] T. Bäck, "Evolution strategies: An alternative evolutionary algorithm," in *Proc. Eur. Conf.*, Brest, France. Cham, Switzerland: Springer, Sep. 2005, pp. 1–20.
[65] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evol. Comput.*, vol. 9, no. 2, pp. 159–195, Jun. 2001.
[66] G. Petelin, G. Cenikj, and T. Eftimov, "TinyTLA: Topological landscape analysis for optimization problem classification in a limited sample setting," *Swarm Evol. Comput.*, vol. 84, Feb. 2024, Art. no. 101448.
[67] R. Morgan and M. Gallagher, "Sampling techniques and distance metrics in high dimensional continuous landscape analysis: Limitations and improvements," *IEEE Trans. Evol. Comput.*, vol. 18, no. 3, pp. 456–461, Jun. 2014.
[68] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional space," in *Proc. 8th Int. Conf.*, London, U.K. Berlin, Germany: Springer, Jan. 2001, pp. 420–434.

GAŠPER PETELIN (Member, IEEE) received the master's degree from the University of Ljubljana, Slovenia. He is currently pursuing the Ph.D. degree with the Jožef Stefan International Postgraduate School. He is a Young Researcher with the Jožef Stefan Institute. His research interests include time-series analysis, black-box optimization, and representation learning.

GJORGJINA CENIKJ (Student Member, IEEE) received the master's degree from the Jožef Stefan International Postgraduate School, Ljubljana, Slovenia, where she is currently pursuing the Ph.D. degree. She is a Young Researcher with the Jožef Stefan Institute. Her main research interests include natural language processing, machine learning, and representation learning.

2.2.1 Statistical Analysis

Here, we present a statistical analysis of models for all three tasks—classification, high-level property prediction, and algorithm selection—using a methodology similar to the one described in Section 2.1.3.

For 2D classification problems as shown in Figure 2.3, the `rfm` achieves the same performance as the best-performing `ela_meta` and `ic` feature sets. However, when differentiating between problems with higher dimensions, the performance of `rfm` decreases relative to the best-performing feature sets. As a result, the difference between the proposed feature sets and the `ela_meta` feature set becomes statistically significant. Across all dimensions, the proposed features consistently and significantly outperform the baseline, indicating that the features are meaningful to some extent for differentiation between problems.

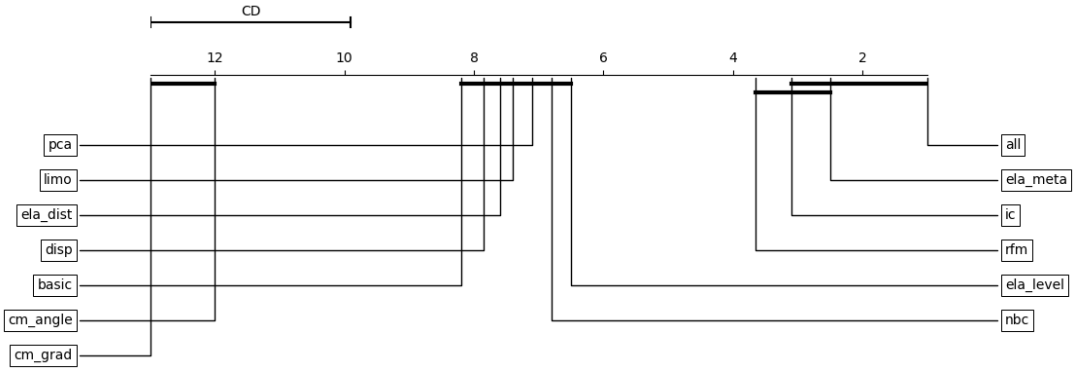


Figure 2.3: Critical difference diagram illustrates the results of the Neymani test. Feature portfolios connected by lines indicate that there is no statistically significant difference between their classification accuracies, while those not connected show a significant difference. The plot is only for 2D objective functions.

For high-level property prediction, the results are somewhat less clear, depending on the problem dimension and the high-level property we are trying to detect. For multimodality, the proposed features do not significantly outperform the baseline, except for problem dimension 2. In terms of global structure detection, the `rfm` features are among the best for lower-dimensional problems, with no statistical difference between the proposed features and the best-performing `ic` features. However, for dimensions 5 and higher, performance drops substantially, though the features still outperform the baseline. For the last high-level property of funnel detection, the `rfm` features outperform the baseline for all problem dimensions, though there is no statistical difference between them and the best-performing feature sets.

For AS the conclusions with both LOIO and LOPO methodologies are almost identical to the topological features. Using the same approach as before, we conclude that for LOIO, the `ela` feature groups significantly outperform the proposed topological features at a significance level of 0.05. Additionally, AS models based on topological features significantly outperform the proposed baseline. However as discussed in the previous section, for LOPO, none of the AS models, including those based on both `rfm` and `ela` features, statistically outperform the proposed baseline.

2.2.2 Discussion

This section explores a novel approach for feature extraction in single-objective optimization problems using randomly initialized filters, focusing on their ability to differentiate

between distinct problem instances and capture high-level problem properties, and algorithm selection. The study evaluates the following hypotheses: **H2a** - *Features based on randomly initialized filters calculated using distances between optimization problem samples are able to differentiate between distinct single-objective optimization problems and capture their high-level properties*, and **H2b** - *Performance of predictive models for optimization algorithm selection based on such features is comparable to the performance of models based on existing exploratory landscape analysis features.*

This study demonstrates that the proposed RFM features are capable of distinguishing between optimization problems and effectively predicting high-level properties such as multimodality and the presence of global or funnel structure. Across all instances, the proposed features outperform the baseline in a statistically significant manner and are often comparable to state-of-the-art approaches, confirming hypothesis **H2a**, as both problem differentiation and high-level property prediction are successfully achieved.

The second approach in this study explores the application of RFM features for algorithm selection. The results reveal that although these features significantly surpass the baseline in the LOIO evaluation, they fail to outperform it in the LOPO evaluation. Moreover, and crucially, the proposed features perform worse than the ELA features in LOIO, with no notable difference observed in LOPO. These outcomes lead us to reject hypothesis **H2b**, as ELA features demonstrate superior performance under some evaluation methods.

The main contribution of the proposed approach for creating features based on randomness is, therefore, the demonstration that these features encode useful information about the objective function and can be used for some downstream tasks in the field of optimization.

Chapter 3

Feature Construction for Time Series Analysis

This chapter delves into feature construction techniques within the context of time series analysis. It specifically examines two critical aspects: the robustness and explainability of these techniques. The first paper explores the selection of optimal forecasting algorithms by leveraging time series features, emphasizing how these processes can be made more explainable and interpretable. Meanwhile, the second part investigates the robustness of feature construction methodologies against common time series distortions and assesses their implications for subsequent classification tasks.

The paper *Towards Understanding the Importance of time series Features in Automated Algorithm Performance Prediction* (Petelin et al., 2023), published in *Expert Systems with Applications*, focuses on applying an explainability approach to algorithm selection pipelines. To the best of our knowledge, this is the first study to investigate the use of explainability techniques for algorithm selection in this domain. The paper explores which features are considered important when performing algorithm selection for forecasting models and examines whether these important features remain consistent across various feature importance techniques and algorithm selection models.

The second part of this chapter presents the research that was recently submitted and is currently under review for publication. This study examines the impact of common distortions on time series classification, evaluating how different algorithms respond to these perturbations. The focus is on measuring the decline in classification accuracy as the severity of distortions increases, providing insights into the robustness of various methods. For the sake of consistency, this work is structured as a paper, similar to previous studies.

3.1 Towards Understanding the Importance of Time Series Features in Automated Algorithm Performance Prediction

When practitioners aim to select the most suitable time series forecasting algorithm, they are often faced with a large pool of options to choose from. However, this selection process is not always straightforward, as different algorithms typically have requirements and limitations regarding the characteristics of the time series they can handle. These requirements may include stationarity, the length of the time series, heteroscedasticity, and one or more seasonal patterns. Consequently, it is highly beneficial to automate this process by recommending the appropriate forecasting algorithm based on the specific properties of the time series.

The first paper examines the use of time series meta-features for predicting the performance of various time series forecasting algorithms. The study evaluates multiple feature construction techniques to create representations of time series data, which are subsequently used to train meta-models for forecasting algorithm performance prediction. This paper investigates hypothesis **H3**.

The paper demonstrates that existing feature construction techniques can effectively predict the performance of forecasting algorithms for new time series, significantly outperforming the baseline model in performance prediction. Beyond this, the primary goal is to explore how algorithm selection models make predictions and identify the most relevant features driving these decisions. The study highlights key features such as time series median, standard deviation, autocorrelation at various lags, and skewness, which are consistently identified as important across multiple algorithm selection models and feature importance techniques. This consistency underscores the reliability of these features for forecasting performance prediction and their value in algorithm selection. Overall, the findings emphasize the critical role of feature extraction and analysis in automated algorithm performance prediction, demonstrating that the careful selection and understanding of relevant time series features can substantially enhance forecasting tasks.

Expert Systems With Applications 213 (2023) 119023

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Towards understanding the importance of time-series features in automated algorithm performance prediction

Gašper Petelin [a,b,*], Gjorgjina Cenikj [a,b], Tome Eftimov [a]

[a] Computer Systems Department, Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia
[b] Jožef Stefan International Postgraduate School, Jamova cesta 39, 1000, Ljubljana, Slovenia

ARTICLE INFO

Keywords:
Time-series forecasting algorithm performance prediction
Time-series analysis
Landscape analysis

ABSTRACT

Accurate and reliable forecasting is a crucial task in many different domains. The selection of a forecasting algorithm that is suitable for a specific time series can be a challenging task, since the algorithms' performance depends on the time-series properties, as well as the properties of the forecasting algorithms. The methodology and analysis presented in this paper are contributing towards understanding the performance of time-series forecasting methods. Instead of using time-series meta-features only to obtain a good meta-model that can predict the performance of a forecasting algorithm, the methodology can link which features are important for which forecasting methods. We used time-series meta-features extracted using the *tsfresh* and *catch22* libraries. We also found that the importance of the meta-features changes depending on the meta-model that is used. There are only a few meta-features that always appear important for a given forecasting method no matter which meta-model will be used for learning, which further provides opportunities to select a model-agnostic feature portfolio. In addition, different feature importance techniques can provide different results that are related to the methodology that is used by the meta-model. By using the feature importance obtained by a meta-model and a specified feature importance technique, we can define a representation of a forecasting method behavior, which can further provide an insight into which forecasting methods have similar behavior.

1. Introduction

Nowadays, a lot of industrial real-world problems involve the analysis of time-series, i.e. chronologically collected data points. Tracking the price fluctuations and price of a security over time in the financial, investment, and business domains, assessing disease risk using longitudinal patient history data in the medical domain, and weather forecasting are only a few examples to be mentioned. Due to their ubiquitous presence in various domains, numerous algorithms have been proposed for time-series forecasting and analysis (Dama & Sinoquet, 2021). Supporting research in this direction, the Makridakis Competitions have been largely responsible for pushing forward the development and evaluation of novel time-series forecasting models by evaluating their strengths, weaknesses, and suitability to different time-series instances, as well as releasing diverse time-series datasets from various fields. These competitions published a dataset of time-series data instances, where different researchers test time-series forecasting algorithms whose performances are further stored. In the end, the algorithm that is the best on average across the test data instances is selected as the winner.

Despite a lot of studies that have been already conducted in time-series forecasting (Deng, Karl, Hutter, Bischl, & Lindauer, 2022), the main challenge is which algorithm to select for each dataset, or even more precisely, for each data instance. This is a well-known problem called algorithm selection (AS), which endeavors to identify one or several algorithms which are well-suited for a given task (Salisu, Abdulrahman, Adamu, Ado, & Rilwan, 2017). The ever-increasing number of Machine Learning (ML) algorithms, along with their potentially infinite hyperparameter search spaces and the choice of constituents of composite methods such as ensembles, produce an exponential number of configuration combinations, which makes AS a challenging problem with high computational costs (Cohen-Shapira & Rokach, 2021).

One way to perform AS is the use of meta-learning (Brazdil, Giraud-Carrier, Soares, & Vilalta, 2009; Vanschoren, 2019), which involves training a ML model to automatically predict the best performing algorithm(s) for an unseen data instance, based on its (meta-) features. The performance of an AS model is related to reliable performance prediction, so automated algorithm performance prediction is a crucial step in an AS pipeline. For this purpose, in this paper, we are focusing on the automated algorithm performance prediction task.

* Corresponding author at: Computer Systems Department, Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia.
E-mail addresses: gasper.petelin@ijs.si (G. Petelin), gjorgjina.cenikj@ijs.si (G. Cenikj), tome.eftimov@ijs.si (T. Eftimov).

https://doi.org/10.1016/j.eswa.2022.119023
Received 18 June 2022; Received in revised form 1 September 2022; Accepted 9 October 2022
Available online 15 October 2022

Algorithm performance prediction can be performed in different learning scenarios. One way is to merge it with the AS step and treat it as a multi-class classification task, where the goal is to predict a single best algorithm, or treat it as a multi-label classification task, where several algorithms can be selected. The other learning scenarios involve solving a regression task, where the ML model predicts the performance achieved by each algorithm (Talagala, Li, & Kang, 2021), or ranking, where the aim is to rank the algorithms according to some evaluation metric (Cohen-Shapira & Rokach, 2021; Tyrrell, 2020). In the latter scenario, the AS step is performed on top of the regression results. The advantage of developing regression models for automated algorithm performance prediction over classification ones is to track the magnitude of differences between performances of different algorithms, as they predict numerical performance values.

The success and reliability of an automated performance prediction task are conditioned on several aspects, which should be addressed with great care: (i) the quality of data used to train the ML model, (ii) the representation learning method used to extract features to represent the time-series data instances, and (iii) the selection of the ML model that will be used for learning. In this study, the main focus is on the intersection between the second and the third step or providing explanations of which features are the most important for reliable algorithm performance prediction.

Transforming time-series data into informative and interpretable features can be a computationally expensive and challenging task. Extracted features have to capture different dynamic time-series properties that are informative enough to be used in further pipelines like regression, classification (Ruiz, Flynn, Large, Middlehurst, & Bagnall, 2021) or clustering (Eftimov et al., 2022). Multiple such sets of features that define the data instance representation have been proposed, which extract informative features across different domains and applications, the most notable ones being the Canonical Time Series Characteristics (*catch22*) and the representations provided by the Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (*tsfresh*) library. Recently, many representations are proposed that are based on deep neural network representations (i.e. ROCKET (Dempster, Petitjean, & Webb, 2020), MiniROCKET (Dempster, Schmidt, & Webb, 2021)), with the main drawback that they are black-box and limit the opportunity for learning explainable ML models. In Henderson and Fulcher (2021), different techniques and libraries for extracting time-series meta-features are explored and the trade-offs between them are analyzed. The limitation of most of the approaches where time-series meta-features are extracted and used for AS is that the features cannot capture all of the necessary information.

Our contribution: To go beyond this, in this paper, we analyze the use of time-series features on the performance of ML models predicting the performance of time-series forecasting algorithms. In particular, we use the *catch22* and *tsfresh* libraries to generate feature representations of the time-series in the M4 dataset (Makridakis, Spiliotis, & Assimakopoulos, 2018, 2020), which are then used to train ML regression models to predict the performance of 61 time-series forecasting algorithms. The main motivation is therefore not to create a meta-learning approach that selects the best algorithm or a set of algorithms and combines them to get good forecasting results, but to explain what features are being used when predicting performance and to quantify their importance. We also explore how feature importances change with different meta-learning models and how they change when predicting performance for a specific forecasting algorithm. To provide a robust and fair analysis of the importance of each feature, we employ several methods for determining feature importance, including permutation feature importance, the SHapley Additive exPlanation (SHAP) (Lundberg & Lee, 2017) framework, as well as the feature importance derived from the intrinsic interpretability of models based on decision trees. The obtained feature importance values are analyzed to get a better insight into how they are related to predicting forecasting algorithms' performance. We provide a detailed exploration of which meta-features are important for different forecasting algorithms and meta-models used to predict performance.

The paper is organized as follows: Section 2 describes the existing approaches for algorithm selection related to the domain of forecasting. In Section 3 we introduce the methodology used to obtain feature importances, followed by the results in Section 4 and limitations of the study in Section 5. Finally, in Section 6 we conclude our work and put forward some future research directions.

2. Related work

A large amount of research on diverse sets of time-series data has been conducted (Makridakis & Hibon, 1979; Newbold & Granger, 1974) to determine if and how different time-series properties affect the accuracy of forecasting methods. Analysis has shown that different properties can have a large influence on the performance of forecasting methods, although being able to capture more complex time dependent relations, does not necessarily produce better results. The link between forecasting accuracy and time-series features was explored in Meade (2000) where authors trained a regression model to predict the performance of nine forecasting methods based on 25 extracted times-series features. At that time, methods for feature creation together with datasets for forecasting were fairly limited.

Several research works have already explored the meta-learning paradigm for automated AS for time-series forecasting. The Feature-based FORecast Model Selection framework (Talagala, Hyndman, & Athanasopoulos, 2018) applies a Random Forest(RF) model with time-series features as inputs, for the selection of a single forecast model. The Feature-based Forecast Model Averaging framework (Montero-Manso, Athanasopoulos, Hyndman, & Talagala, 2020) applies a gradient boosting model to select the weights for a weighted forecast combination, using 41 features extracted by the tsfeatures (Hyndman, Kang, Montero-Manso, Talagala, Wang, Yang, & O'Hara-Wild, 2020) library. The Feature-based Forecast Model Performance Prediction framework (Talagala et al., 2021) treats the forecasting AS problem as a multi-class ranking problem, i.e. meta-learning is used to rank forecast models by simultaneously predicting forecast errors, allowing the user to identify a subset of forecasting models.

The benefits of ensembles in forecasting have been pointed out in Gastinger, Nicolas, Stepić, Schmidt, and Schülke (2021), where it was demonstrated that there is no single best ensemble model and hyperparameter configuration which is well suited for all time-series instances, introducing the need for meta-learning. The meta-learning in this case is implemented by training a binary classifier for each combination of ensemble model and hyperparameter configuration, which indicates whether the model should be used for a specific time-series instance. The instances are represented by the meta-features extracted using the tsfeatures (Hyndman et al., 2020) library and seven general meta-learning features, which are not specific to time-series.

A two-step meta-learning approach has also been proposed for time-series forecasting ensembles (Vaiciukynas, Danenas, Kontrimas, & Butleris, 2021), which first involves ranking the forecasting models by predicting the error they would achieve on the testing instance, and then determining the number of models that should be used in the forecasting ensemble. Both tasks are performed using RF regression models. The time-series instances are represented using 390 meta-features, obtained by generating 130 different features on the original time-series instances, as well as two transformations of each instance. Permutation feature importance is used to analyze the contribution of the meta-features to the meta-models' performance. However, the features are first grouped into several sets, and the importance is reported on a set level, and not on the level of individual features. In contrast, we are interested in how individual features contribute to the meta-model's performance.

We also need to emphasize that meta-learning studies that select the most appropriate algorithms or combine them into an ensemble use

an apriori selected algorithm portfolio (i.e., set of forecasting methods that are used in the study). Nowadays, one of the biggest open research questions is how to select which algorithms should be used in order to generalize the meta-learning approach and to decrease the bias presented in the selection of the set of methods. We are not addressing this question in our study, since our goal is to provide a general methodology to see how the time-series features are linked to the performance of any forecasting method. We are using a set of forecasting algorithms that entered the M4 competition. However, we would like to point out that the field of forecasting methods is increasingly moving towards deep learning based models that can be trained on many instances (sometimes referred to as global models (Hewamalage, Bergmeir, & Bandara, 2022)) and are able to generalize the learned knowledge to previously unseen instances. Such models include recurrent neural networks (Werbos, 1990) with further improvements (Chung, Gulcehre, Cho, & Bengio, 2014; Salinas, Flunkert, Gasthaus, & Januschowski, 2020), convolutional neural networks (Borovykh, Bohte, & Oosterlee, 2018; Chen, Kang, Chen, & Wang, 2020; Li et al., 2021) and transformers (Vaswani et al., 2017). Focus is also shifting towards neural networks that try to combine approaches from deep learning and interpretability from classical statistical modes such as N-BEATS: Neural basis expansion analysis for interpretable time series forecasting (Oreshkin, Carpov, Chapados, & Bengio, 2019) and Temporal fusion transformers (Lim, Arık, Loeff, & Pfister, 2021).

3. Methodology

Fig. 1 gives an overview of the proposed methodology for explaining the importance of features used for representing time-series instances for predicting the performance of ML algorithms achieved on them. The methodology involves three main steps:

- *Data collection*, which involves extracting time-series features of the data instances and collecting forecasting performance data achieved on them. Here, features of individual time-series instances are extracted and the performance of forecasting algorithms is collected from a publicly available repository;
- *Building a diverse portfolio of performance prediction models*, where forecasting performance of time-series algorithms is predicted based on the extracted time-series features using different ML models;
- *Feature importance analysis*, where multiple feature importance approaches are used to determine which features are the most influential ones for predicting forecasting performance and how feature importance changes with different ML models.

3.1. M4 dataset

The M4 time-series forecasting dataset (Makridakis et al., 2018, 2020) was used for the fourth edition of the Makridakis Forecasting Competition. The dataset consists of 100,000 time-series collected either yearly, quarterly, monthly, weekly, daily or hourly. All the time-series instances are divided into a training set for training the forecasting models and a test set designed to evaluate forecasts. Training instances can also greatly differ in length. The shortest time-series consists of only 13 yearly observations while the longest time-series collected daily consists of 9,919 observations.

At the end of the competition, 61 forecasting algorithms were submitted together with forecasts for all 100,000 time-series. Submitted algorithms can be broadly categorized as statistical forecasting algorithms, forecasting algorithms based on ML models or hybrid approaches. The best performing algorithm was a hybrid approach (Smyl, 2020) which combined exponential smoothing with a recurrent neural network. Other top performing approaches mostly relied on combining forecasts of different statistical approaches to achieve good results.

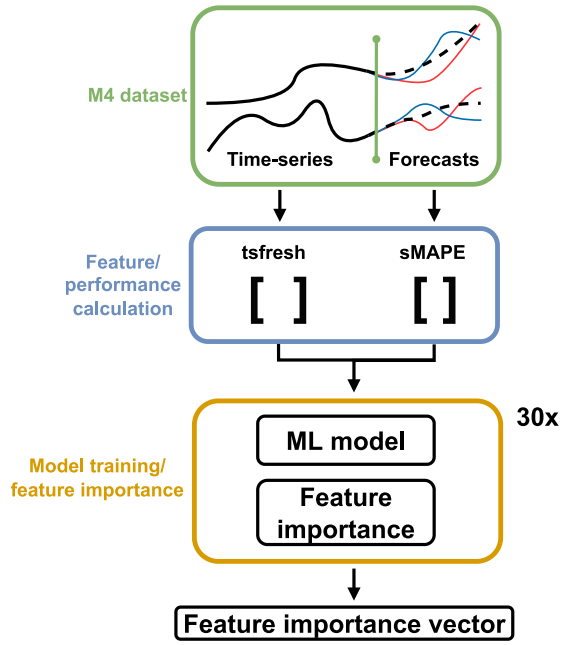


Fig. 1. Pipeline for calculating feature importance. Features and forecasting performance is first extracted from M4 dataset. Extracted features and forecasting performances are repeatedly split into 30 train/test sets on which performance prediction algorithms are trained and feature values are calculated.

Table 1
Hyperparameters for the meta-models used to predict performance.

Model	Parameter	Value
Neural network	activation	ReLU
	number of hidden layers	3
	hidden layer size	500
	optimizer	Adam
RF [MT]	number of trees	100
	max depth	No limit
	min samples in a leaf node	No limit
RF [ST]	number of trees	25
	max depth	No limit
	min samples in a leaf node	2
XGBoost	learning rate	0.3
	maximum tree depth	6
KNN [Cosine]	neighbors	5
KNN [Euclidean]	neighbors	5
Mean	–	–

3.2. Extracting time-series features and collecting the performance data

The time-series data instances were used in the feature extraction process to obtain tabular data that is further linked to the algorithm performance.

3.2.1. Feature extraction

First, a set of time-series features are extracted from the training part of each time-series instance in the dataset using the methods described later in this section. When using forecasting algorithms to predict the future behavior of time-series, a common practice is to first transform time-series to achieve/improve stationarity or to stabilize the variance. This is especially important for some forecasting algorithms (e.g. MA, AR or ARMA) that can only operate on time-series that are strictly/weakly stationary process (Patterson, 2011; Van Greunen,

Heymans, Van Heerden, & Van Vuuren, 2014). One transformation that makes time-series stationary is to compute the differences between consecutive observations, also known as differencing. Differencing can remove trends and seasonality which helps with stabilizing the mean of a time-series. The other property that is desirable when forecasting is to stabilize the variance of a time-series, which can be achieved by applying a logarithmic transformation. Before extracting features, we also apply differencing and logarithmic transformation to the time-series. Due to the specific structure and properties of time-series instances contained in the M4 dataset, feature extraction methods may sometimes produce invalid features (i.e. features with missing values) of features with the same value for all time-series. Features with missing values or identical values are therefore removed.

In our case, we used two well-known features sets for extracting features for time-series data instances, Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (*tsfresh*) (Christ, Braun, Neuffer, & Kempa-Liehr, 2018) and The Canonical Time Series Characteristics (*catch22*) (Lubba et al., 2019), which are most commonly used in time-series forecasting tasks. These methods were used three times depending on which data were applied: (i) raw original time-series data, (ii) its differencing transformation, and (iii) its logarithmic transformation.

tsfresh is a library for extracting a diverse set of features that proved to be effective in capturing and extracting quality information in different scientific areas (financial, biological, industrial applications, etc.). The library can extract just under 800 features from the different feature groups (i.e. statistical, information based, model-based, stationarity, frequency-based, domain-specific features, etc.).

catch22 is a subset of features that were selected as being the most informative/discriminatory features from the set of over 7,700 features from Highly Comparative Time Series Analysis (*hctsa*) (Fulcher, Little, & Jones, 2013) evaluated on the University of California, Riverside (UCR) dataset (Bagnall, Lines, Bostrom, Large, & Keogh, 2017). A subset of 22 features was determined in three steps: (i) remove the features that cannot be calculated or are sensitive to the mean and variance, (ii) evaluate the predictive performance of features and discard features that are below some preselected threshold, (iii) group features into 22 clusters and select the feature that is interpretable, achieves good performance and can be efficiently computed.

We also need to point out here that a common approach for extracting features from time-series is also by applying random convolutional kernels to the existing time-series dataset as done in ROCKET (Dempster et al., 2020) and later in MiniROCKET (Dempster et al., 2021). Similarly, feature generation can also be performed by representing time-series in a frequency domain (Chen et al., 2019; Srinivasan, Eswaran, et al., 2005) or by using a discrete wavelet transformation (Chaovalit, Gangopadhyay, Karabatis, & Chen, 2011). While such methods are extremely powerful when classifying time-series instances, the downside of such approaches is mainly that extracted features are not meaningful to humans and are hard to interpret. Since they provide black-box representations, they were omitted from our analysis.

3.2.2. Performance data

When forecasts are performed, one has to measure the error or forecast compared to the actual observations. The symmetric mean absolute percentage error (sMAPE) (Hyndman & Koehler, 2006) is one such measure that quantifies the prediction accuracy of a forecasting method. The forecasted values are first subtracted from the actual observed values and the absolute value is taken. The obtained difference is further normalized by the sum of absolute values of the forecasted and true predictions. Calculated errors are averaged over the whole forecasting horizon. The following equation is used to calculate the sMAPE metric:

$$sMAPE = \frac{2}{h}\sum_{t=n+1}^{n+h}\frac{\left|Y_t - \hat{Y}_t\right|}{\left|Y_t\right| + \left|\hat{Y}_t\right|}$$

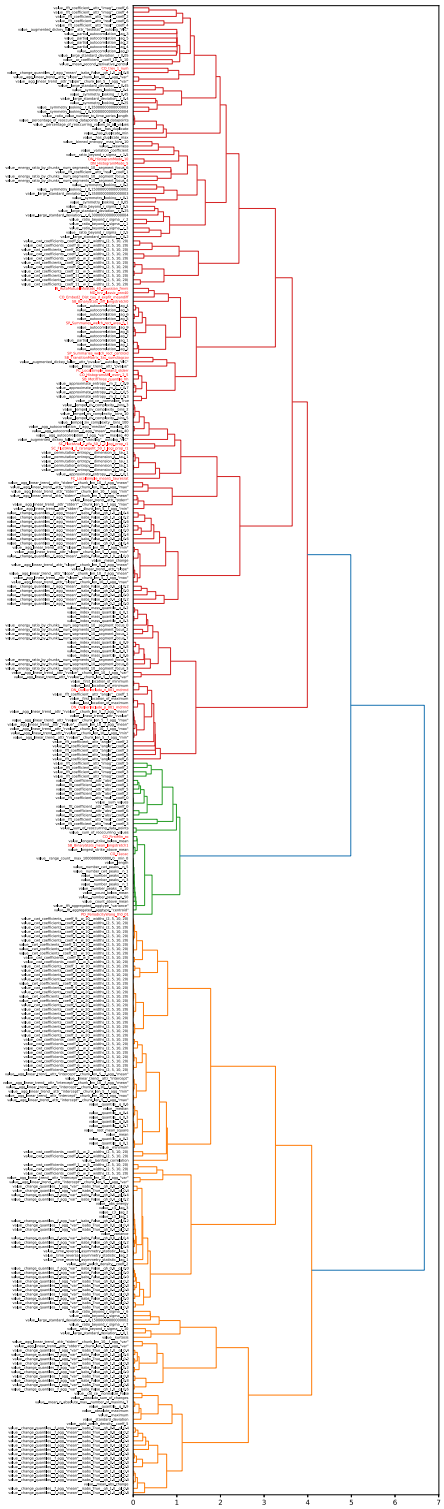


Fig. 2. Dendrogram of *tsfresh* (black) and *catch22* (red) features based on the Pearson correlation between features. The similarity measure is transformed into a distance measure with $1 - abs(pearson(F_i, F_j))$ where F_i and F_j are extracted feature vectors spanning all 100,000 time-series instances. Clusters are merged with the Ward linkage function.

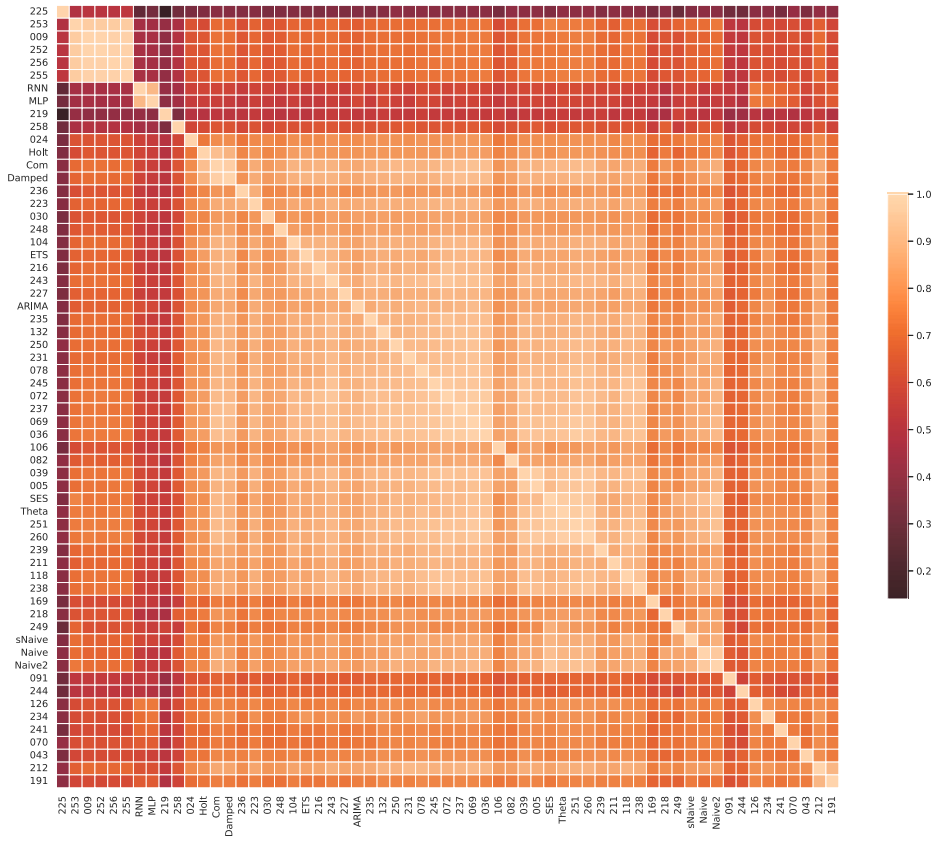


Fig. 3. Pearson correlation between forecasting performance calculated using sMAPE on 100,000 time-series instances.

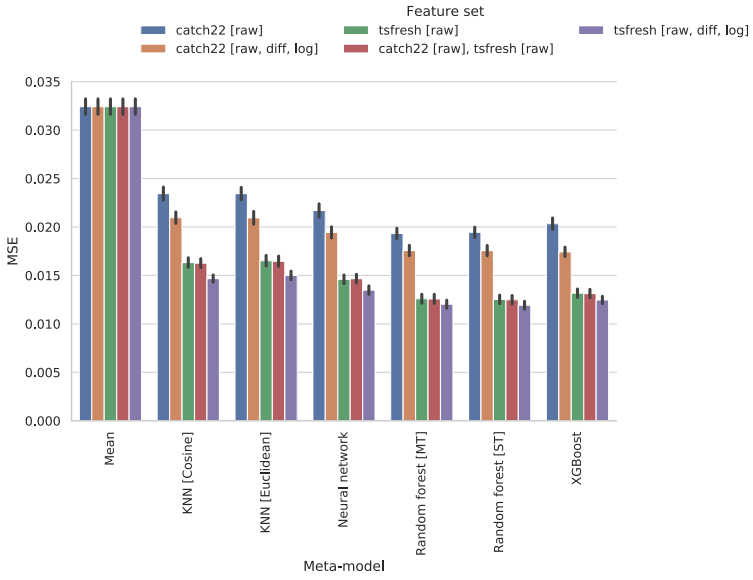


Fig. 4. MSE achieved by different ML models and different sets of features when predicting forecasting errors for 61 time-series algorithms.

where n is the number of available datapoints in the training part of each time-series, h is the forecasting horizon and Y_t and $\hat{Y}_t$ are the true and the forecasted value of time-series and time t. Lower sMAPE values indicate lower prediction errors, so the goal is to minimize this measure. Note that sMAPE is a modification of the MAPE metric and improves on some shortcomings of MAPE such as having a lower bound

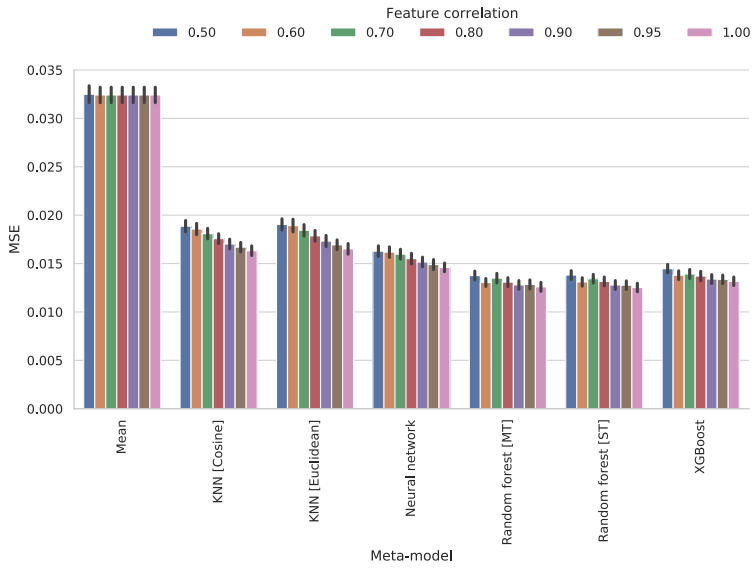


Fig. 5. MSE for different ML models with *tsfresh* features and different thresholds for removing correlated features.

of zero and upper bound of two. Even with this modification, sMAPE can have problems if both the forecast and the actual values are equal to zero, making the denominator also zero. The M4 dataset shifts the time-series in such way to avoid this problem.

Based on the known test parts of the time-series and forecasts made with each of the 61 models, we do the performance extraction by computing the forecasting error between all pairs of forecasting algorithms and time-series instances, expressed using the sMAPE.

3.3. Building a diverse portfolio of performance prediction models

To predict the performance of the 61 forecasting algorithms on the time-series data instances represented by the extracted features, we build a diverse set of performance prediction models, which we refer to as meta-models. The analysis of the meta-models is further used to provide explanations about the importance of the time-series features. For this purpose, the obtained time-series feature data and performance data is first split into training and testing portions, making up 90% and 10% of the original data, respectively. The procedure is repeated 30 times using random sampling, to obtain 30 splits of the original data. Further, we used different ML algorithms including *Neural Network*, *Random Forest (RF)*, *XGBoost*, *K Nearest Neighbors (KNN)*, and *Mean baseline* to train the meta-models on each of the train splits and evaluated on the corresponding test sets. More details about the algorithms used to train the meta-models are presented in Appendix A. Repeating the model training on different train/test splits guarantees that we get a robust estimation of the expected error when predicting forecasting performance and subsequently what features are used for making forecasts and how features differ with different data splits handling the inherent bias that originates from the data quality in different splits.

The meta-models were either trained as multi-target regression (MTR) models, where one model predicts performance for all forecasting algorithms, or as multiple single-target regression (STR) models, where one model is trained for each forecasting algorithm. Due to the computational complexity involved in training the models and calculating feature importance, *Neural Network* and *KNN* were trained only as MTR models. *RF* was trained twice, once as a MTR model and once as a collection of multiple STR models. Lastly, *XGBoost* is designed to only work as a STR model therefore one is trained for each of the 61 forecasting algorithms.

When training the meta-models used to predict forecasting algorithms' performance, an extensive hyperparameter search is desirable or even required for obtaining models with state-of-the-art performance. Due to the high computational costs, we did not extensively tune the hyperparameters to obtain the best possible performance for the meta-learning models used to predict performance. The goal was primarily to ensure that selected models are diverse (use different learning principles) to see how this influences the feature importance values and the explanations provided for each model.

3.4. Feature importance for providing explanations

For each of the 30 train/test splits and corresponding performance predicting meta-model, we evaluate what features are deemed to be important. Feature importance can differ depending on several aspects: *(i) the meta-model* that predicts the performance based on the extracted features, *(ii) the forecasting algorithm* whose performance is being predicted and *(iii) the method for explaining feature importance*. We analyze the feature importance considering each of the above-mentioned aspects. To do this, we explore the following combinations of feature importance measures:

*(i) Permutation feature importance evaluated on neural network (**P/NN**), multi-task random forest (**P/RF**), singletask random forest (**P/SRF**), XGBoost (**P/XGB**) and Mean predictor (**P/Mean**)*

*(ii) SHAP feature importance evaluated with neural network (**S/NN**), multi-task random forest (**S/RF**), singletask random forest (**S/SRF**), XGBoost (**S/XGB**) and Mean predictor (**S/Mean**)*

*(iii) Feature importance obtained during XGBoost training with different feature importance metrics (**C/XGB, G/XGB, TC/XGB, TG/XGB, W/XGB**)*

*(iv) Feature importance obtained during random singletarget forest training (**RF/SRF**).*

More details about the feature importance models are presented in Appendix B.

4. Results

Here, the results from our analysis are presented, which are split in three parts. First, we provide a separate analysis of the time-series feature space and the performance space, treating them independently. Next, we explore the time-series feature importance in combination with different performance predicting models and time-series forecasting models.

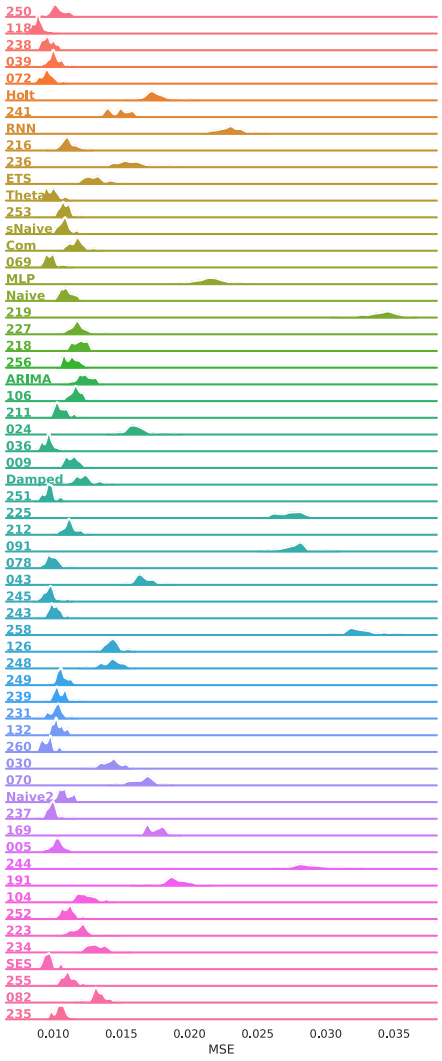


Fig. 6. Ridgeline plot representing the MSE between the true forecasting performance evaluated with MAPE and the performance predicted with RF for each of the forecasting algorithms over 30 train/test splits. The row labels contain the feature names, while the column labels contain the names of the meta-models.

4.1. Experimental setup

For the implementation of the meta-models, we used the *scikit-learn* (Pedregosa et al., 2011), *keras* (Chollet et al., 2015) and *XGBoost* (Chen & Guestrin, 2016) python libraries. The workflow and analysis are performed using Snakemake (Mölder et al., 2021), ensuring reproducibility. The experiments were executed on a system using the Ubuntu operating system, an Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50 GHz, and 1 TB of RAM. The relevant code can be found at the Gitlab repository https://repo.ijs.si/gpetelin/m4-feature-importance. Associated data is available at https://zenodo.org/record/6637637. Reasonable default hyperparameters were selected to balance model performance during train and inference time. Table 1 describes the hyperparameters of the meta-models. Apart from this, feature importance methods produce importance values that can differ in scale. For that purpose, comparison of raw values between different feature importance methods is not always possible. In such cases, feature importance is determined based on the rank of the feature compared to all other features.

4.2. Feature space analysis

Applying the features extraction techniques described in Section 3.2.1, we obtain six different sets of features for each time-series instance. Each one was extracted using the *tsfresh* or *catch22* libraries, in combination with the raw time-series, the time-series that were transformed with the first difference, and the time-series with a logarithmic transformation applied.

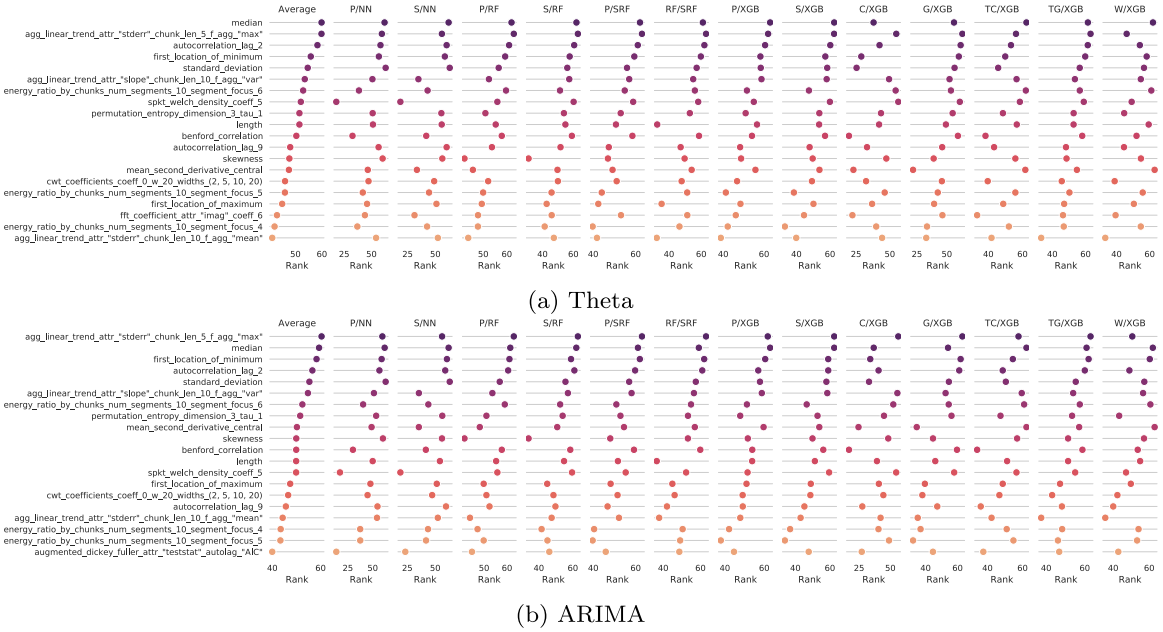
Fig. 2 shows a hierarchical clustering based on Pearson correlations between the *tsfresh* and *catch22* features obtained from the time-series where no transformation was applied. Using this figure, we can observe that a lot of extracted features are linearly correlated. In addition, high correlations between *tsfresh* features are also presented, which is expected, since some features are extracted using the same feature extraction method, only with different parameters. Example of two such features would be feature *cwt_coefficients_coeff_1_w_10_widths_(2, 5, 10, 20)* being strongly correlated with feature *cwt_coefficients_coeff_0_w_10_widths_(2, 5, 10, 20)*, with correlation of 0.96.

Comparing *tsfresh* (black) and *catch22* (red) features in Fig. 2, we can observe that there are a lot of features that are strongly linearly correlated. The second more interesting observation is that *catch22* features do not capture large parts of the time-series landscape. This means that there is a large group of *tsfresh* features that are not correlated with any *catch22* features.

To further explore the correlations between raw and transformed time-series feature, one can compare the Pearson correlation between the raw features and the features obtained from the transformed time-series. Focusing on *catch22*, the mean correlation between raw features and their corresponding feature from log transformed time-series is 0.88. This signals that on average, features from log transformed time-series contain information correlated with raw features. For the differenced time-series, corresponding features are not highly correlated, with the average correlation being 0.24. A similar pattern can also be observed with the *tsfresh* features. In that case, the features extracted from log transformed and differenced time-series have Person correlations with the raw features of 0.66 and 0.35, respectively. It is also the case in both feature extraction methods that some log transformed and differenced features have correlation of 1.0 with their raw counterparts. An example of how extracting features from transformed time-series does not always bring additional benefits is the feature *first_location_of_maximum*, where raw and log transformation produce the same values and therefore do not offer any additional value.

4.3. Performance space analysis

The M4 dataset consists of 61 forecasting algorithms and their predictions for 100,000 time-series instances. Fig. 3 shows the Pearson correlations between sMAPE forecasting performance on all 100,000 time-series instances that exist in the M4 dataset. We can observe that performances of forecasting algorithms are usually strongly correlated, meaning that if one algorithm performs well on a given time-series instance, there is a high probability that another algorithm will also perform well. Based on the performance we can also observe clusters of similarly performing forecasting algorithms. One such example are the algorithms *MLP* (perceptron with pre-applied detrending and deseasonalization) and *RNN* (recurrent network with pre-applied detrending and deseasonalization), both ML-based forecasting algorithms with highly correlated performance. Similarly, algorithms *sNaive*, *Naive* and *Naive2* form another cluster of algorithms where forecasting performance is highly correlated. This is a consequence of the algorithms being similar in nature (random walk models), with small modifications. One forecasting algorithm that is not correlated with other algorithms is the forecasting algorithm *225* that achieves poor forecasting performance evaluated with sMAPE on most of the time-series in the dataset.



(a) Theta

(b) ARIMA

Fig. 7. Feature ranking for Theta (a) and ARIMA (b) forecasting algorithms based on the ranking averaged over 30 train/test splits. The row labels contain the feature names, while the column labels contain the feature importance and meta-model names.

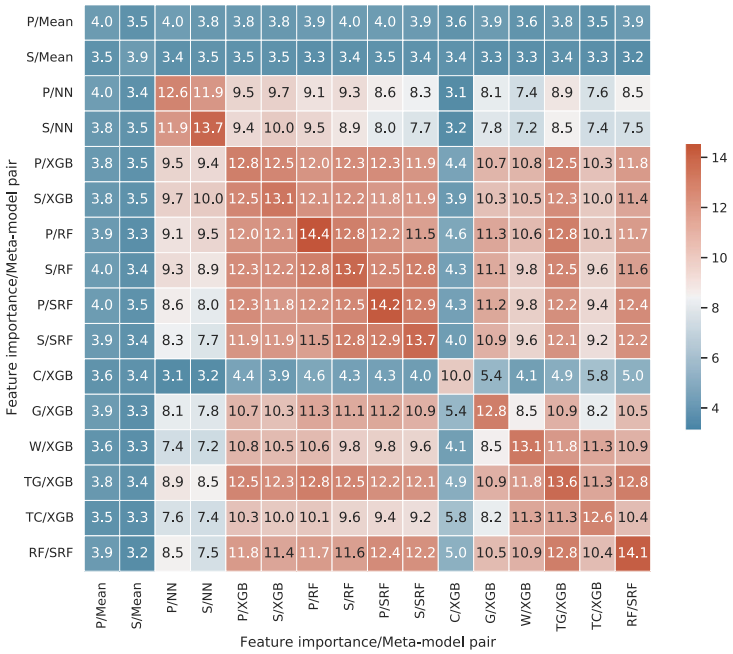


Fig. 8. Agreement on what 15 features are the most important ones as determined by different meta-model/feature importance methods for Theta forecasting algorithm, averaged over 30 train/test splits.

4.4. Meta-model performance analysis

To provide a reliable analysis of the importance of each feature when predicting the performance of a time-series forecasting algorithm, we build diverse meta-models to predict the performance of different forecasting algorithms from the obtained time-series features. Fig. 4 shows the performance of different meta-models when using different sets of features averaged over 30 splits. For additional information, black lines on top of each show the standard deviation in MSE. Each meta-model (i.e., KNN with cosine similarity, KNN with Euclidean similarity, RF, XGBoost, Neural Network, and Mean as a baseline) was evaluated with five different feature sets.

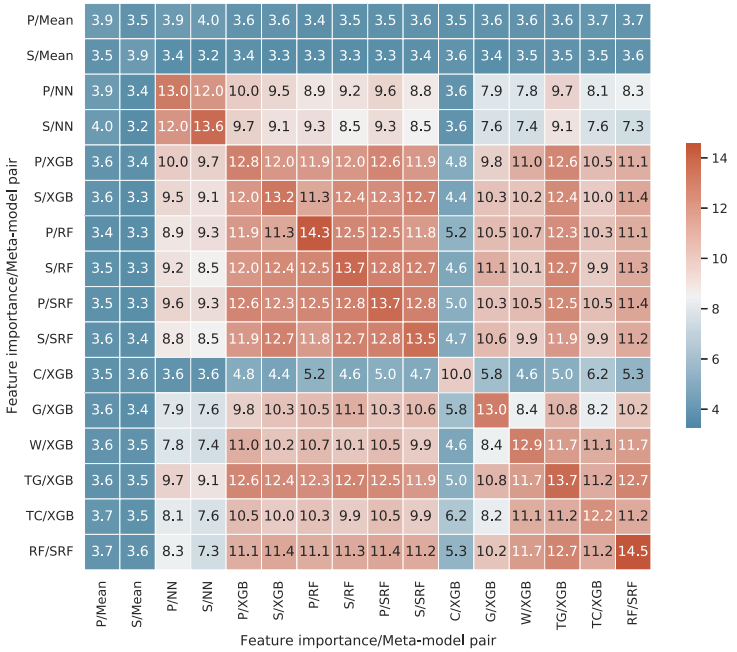


Fig. 9. Agreement on what 15 features are the most important ones as determined by different meta-model/feature importance methods for ARIMA forecasting algorithm, averaged over 30 train/test splits.

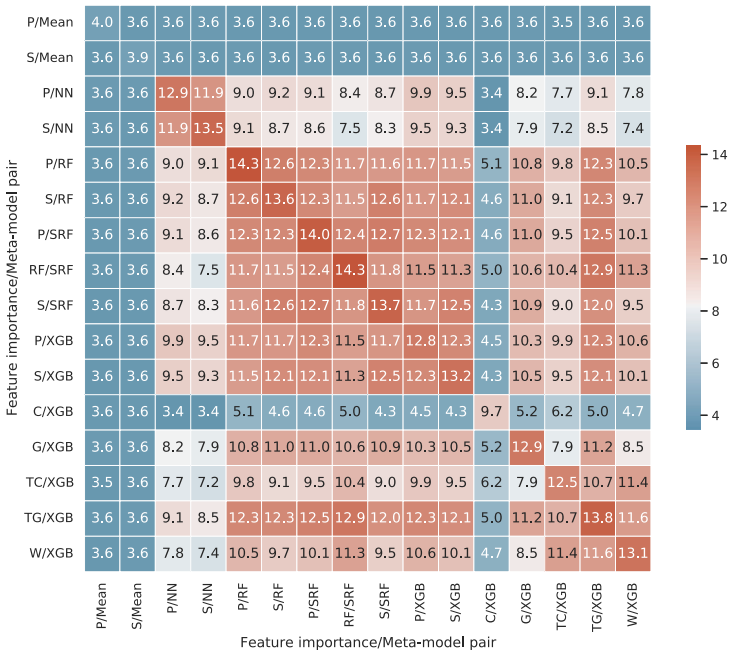


Fig. 10. Agreement on what 15 features are the most important ones as determined by different meta-model/feature importance methods for all forecasting algorithms, averaged over 30 train/test splits.

As feature sets we selected the *tsfresh* features calculated using the raw time-series (*tsfresh [raw]*); *catch22* features calculated using the raw time-series (*catch22 [raw]*); fusion of all *tsfresh* features (*tsfresh [raw, diff, log]*) calculated from the raw time-series, the transformed time-series using differencing, and the transformed time-series using logarithmic transformation; the same fusion using the *catch22* features (*catch [raw, diff, log]*) that fuse raw, differenced and log transformed time-series features; and a fusion of *tsfresh* and *catch22* features calculated from the raw time-series data (*catch22 [raw], tsfresh [raw]*).

The meta-models are evaluated using a Mean Squared Error (MSE) between the ground truth sMAPE performance and the predicted

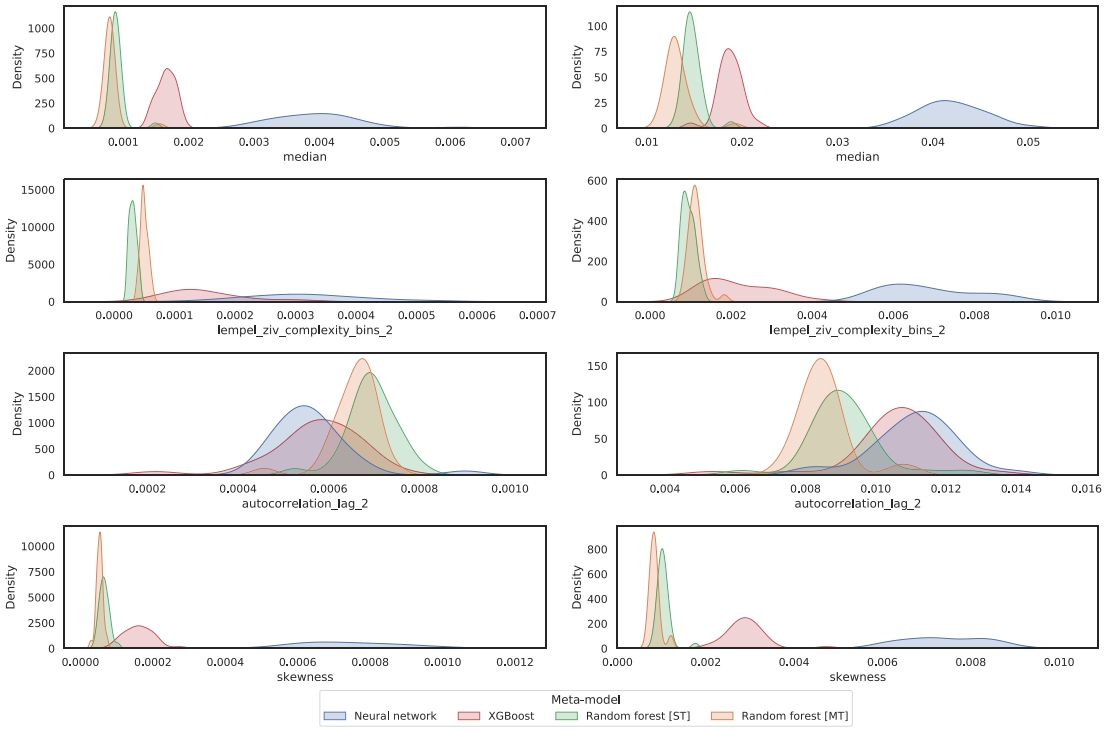


Fig. 11. Kernel density estimation feature importance of features *median*, *lempel_ziv_complexity_bins_2*, *autocorrelation_lag_2* and *skewness* for Theta forecasting algorithms as obtained with Permutation feature importance (left) and SHAP (right). Note that axis use different scales.

sMAPE performance, averaged across all 61 targets. We can observe that the baseline model which always predicts the mean performance for each forecasting algorithm, independent of the feature set used to represent the time-series data instances, achieves a MSE of 0.032. The other meta-models, which rely on the above-mentioned feature sets achieve much better estimates of the performance of the 61 forecasting algorithms.

Comparing the sets of features, we can observe that the *tsfresh* features offer better predictive performance compared to the *catch22* features, independent of the meta-model that is used. By fusing both the *catch22* and the *tsfresh* feature sets, the predictive performance is almost identical to the performance obtained using only the *tsfresh* features. We can therefore conclude that the 323 *tsfresh* features are much more informative than the 22 features extracted with *catch22* when predicting the performance of the 61 forecasting algorithms on the M4 dataset. This is not surprising, since in the feature space analysis (see Section 4.2), we have shown that the *tsfresh* features set is much larger and covers parts of the feature landscape that are not covered by the *catch22* features, i.e. the *catch22* features capture only a subset of the information captured by the *tsfresh* feature set.

The second interesting observation is the predictive performance obtained using the feature sets extracted from the differenced and log transformed time-series. For *catch22*, fusing the features extracted from the raw, differenced, and log transformed data has a large impact on the predictive performance for all used meta-models. With the extended set of *catch22* features, all models achieve practically better MSE compared to the features extracted just from the raw time-series. With the *tsfresh* features, extracting features from the raw time-series and fusing them with features from the differenced and log transformed time-series does improve the meta-learner performance, but the difference in performance obtained using only the raw time-series features, and the fused time-series features is not as large as with the *catch22* feature set. From the results, it follows that all meta-learners achieve good performance with the *tsfresh* features extracted only from the raw time-series data. Therefore, we decide to only use this feature set for further analysis. We need to point out here that if the predictive performance of the meta-learners was the primarily goal, using the *tsfresh* features extracted from the raw time-series and fusing them with the features extracted from differenced and log transformed data would also be beneficial.

Since the feature space analysis shows that the extracted *tsfresh* time-series features can be highly correlated (see Section 4.2), this can sometimes adversely affect the training of certain models and subsequent methods for evaluating feature importance. A way to remove feature correlations is to project data instances to a lower-dimensional space using feature reduction methods, such as Principal Component Analysis or Non-Negative Matrix Factorization. Unfortunately, such an approach makes contributions of individual features difficult to compute. For this purpose, we explore the performance of the meta-models when correlated features are removed, if their correlation exceeds a predefined threshold. Fig. 5 presents the meta-models' performance (MSE averaged across 30 train/test split with standard deviation) when features which have correlations exceeding a certain threshold are removed and thus not used during the training and testing. We have evaluated six different correlation thresholds: 0.50, 0.60, 0.70, 0.80, 0.90, and 0.95. When the threshold is set to 1.00, this means that all *tsfresh* features are kept for the learning process and we did this only for comparing the performance with the meta-models achieved with the reduced feature sets. Removing a subset of highly correlated features slightly degrades the models' performance, indicating that the removed features, despite being linearly correlated, do in fact contain additional information. For the purpose of reducing computational costs associated with computing feature importance for the full set of features and problems that can arise from determining the feature importance of correlated features, we choose to work only with the features that are not linearly correlated with other features above the pre-selected threshold of 0.50.

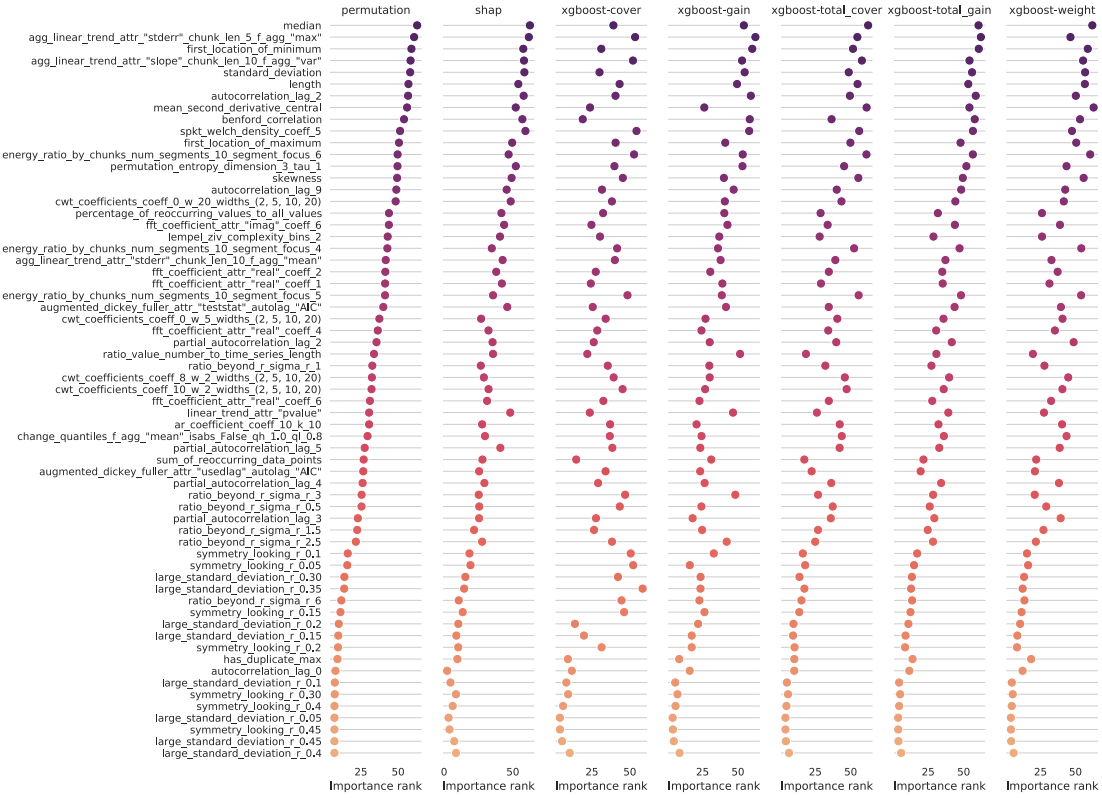


Fig. 12. Feature importance ranking for XGBoost with different feature importance methods aggregated over 30 runs and all forecasting models. For clarity, features are sorted by the permutation feature importance.

Fig. 6 shows the MSE between the actual sMAPE obtained by each forecasting algorithm and the sMAPE predicted with the best performing meta-model RF with the reduced feature set with 0.50 threshold. The MSE is presented for 30 different train and test splits. From the figure, we can see that the performance of the meta-models is generally stable with relatively small deviations in the MSE across different train/test splits.

We can also observe that estimating the sMAPE performance for some of the forecasting algorithms is much more challenging than for others. The MSEs for most of the forecasting algorithms are around 0.01, however for some forecasting algorithms the estimates of the performance are less accurate with MSE above 0.02. The five forecasting algorithms with a larger MSE whose performance is more difficult to predict are algorithms 219, 225, 091, 258 and 244. In general, the algorithms whose performance is hard to predict performed bellow the average in the competition, meaning that their forecasting error evaluated with sMAPE is higher.

4.5. Feature importance for explainability of forecasting algorithms

In this section, we investigate the feature importance obtained for several forecasting algorithms, based on the meta-models and feature importance methods used. In this scenario, a few forecasting algorithms are selected and kept fixed while changing the meta-models and feature importance methods.

Fig. 7 visualizes the top 20 most important features (globally across all forecasting algorithms, meta-models and feature importance models) and how they are ranked with different meta-models/feature importance methods for Theta and ARIMA forecasting algorithms. Note that Theta and ARIMA forecasting algorithms were selected since they are the baseline models with available source code and relatively good forecasting accuracy compared to some other forecasting algorithms. As expected, for the meta-model that only predicts mean values, all features are assigned the same feature importance by both the permutation-based and the SHAP feature importance methods. This confirms that both feature importance methods can discover when features are not used to make a prediction. Because all features have the same importance values, they are omitted in Fig. 7 to improve readability. Other meta-models, however, rely heavily on the features. For those meta-models, large variation in the raking of individual features can be observed based on the meta-model and feature importance methods used. The largest discrepancies in feature importance orders are between model-specific methods (where feature importance is determined while the model is built such as the XGBoost cover feature importance - **C/XGB**) and model agnostic feature importance (where feature importance is determined after the model is built such as SHAP or permutation method). This demonstrates that model-specific features that are built during the models' construction from the train set can substantially differ from other approaches. When comparing feature importance between different meta-models using the same feature importance method, we can observe that tree-based models utilize different features compared to the neural network. Lastly, the feature importance ranking obtained with SHAP and the one obtained with permutation importance is similar when the same meta-model is used. This can be seen when comparing columns pairs such as **P/NN** and **S/NN** or **P/RF** and **S/RF**.

To better demonstrate the similarities between different meta-models and feature importance methods, Figs. 8 and 9 show how many of the top 15 features are shared for different meta-model/feature importance combinations for the Theta and ARIMA forecasting algorithms. This is obtained as an average number of top 15 features

Fig. 13. Feature importance ranks for different meta-models using permutation based feature importance averaged over all forecasting algorithms and 30 runs.

that are shared between all the pairs of 30 runs. We can observe that selecting top 15 most important features can differ depending on what meta-model and feature importance model is used. For example, predicting performance of Theta forecasting algorithm using RF meta-model with permutation feature importance (**P/RF**), top 15 features agree on 14.3 out of 15 features between different train/test splits. Diagonal elements of the figure thus show that even when meta-model and feature importance models are fixed, top features may still differ between different splits. Additionally, when comparing meta-models with different feature importance methods, large disagreements can be observed. For instance, XGBoost with permutation feature importance (**P/XGB**) and gain feature importance (**G/XGB**) achieve average agreement only on 10.8 features for the Theta forecasting algorithm.

Fig. 10 also shows the global top 15 feature agreement between all meta-model/feature importance combinations for 30 runs across all forecasting algorithms. We can again observe that feature importance can vary depending on the meta-model and feature importance method. Therefore, when determining feature importance one is highly dependant on what methods are selected.

Further insight can be obtained when comparing the "raw" feature importance values and not just the rank. Fig. 11 shows the distribution of feature importance values for the Theta forecasting algorithm and four selected features (*median*, *lempel_ziv_complexity_bins_2*, *autocorrelation_lag_2* and *skewness*) as a distribution over 30 train/test splits obtained with the SHAP and permutation feature importance methods. Observation can be made that feature importance distributions differ with different meta-models. For the feature importance of the *median* feature, obtained with permutation feature importance, we can observe that the neural network assigns higher feature importance values than the other methods. It is also interesting to note that the

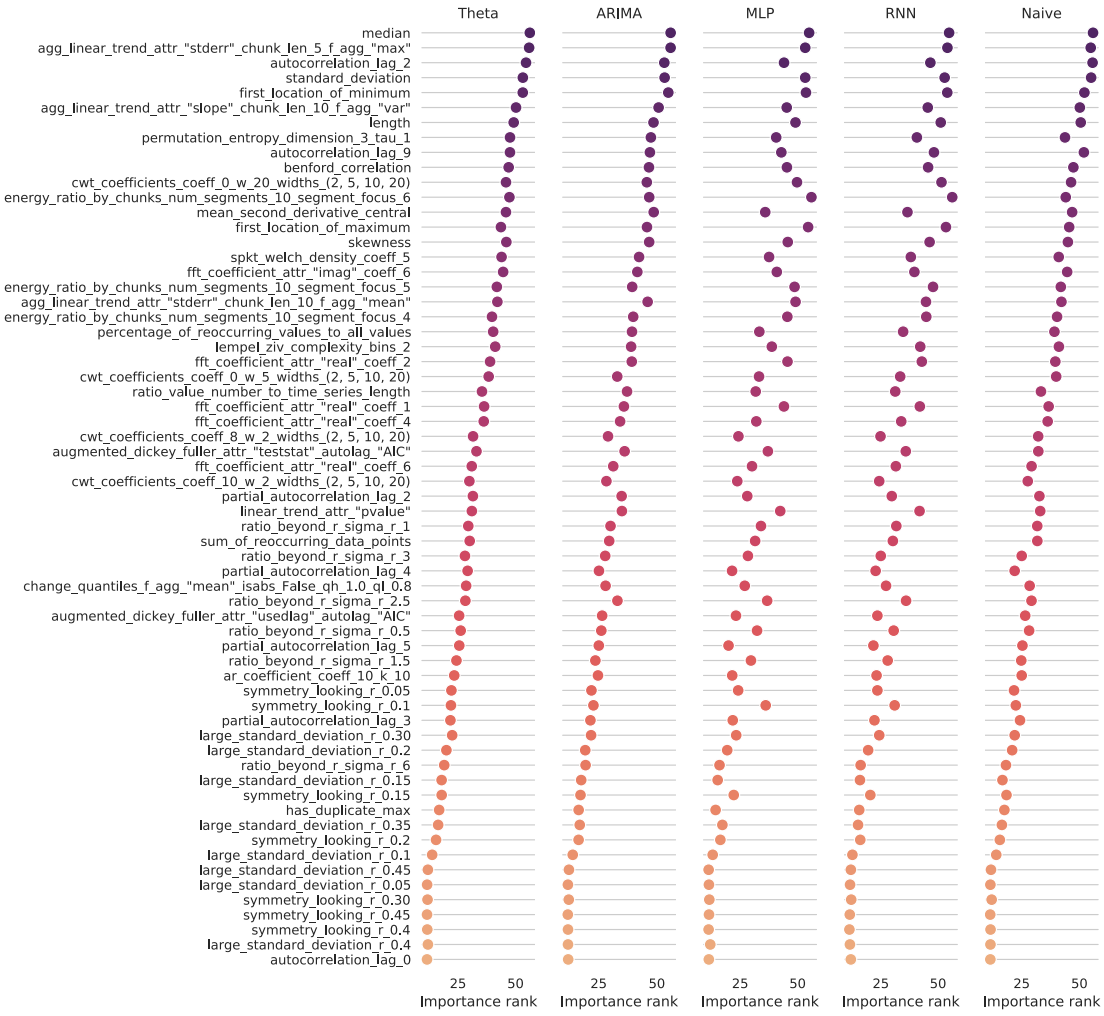


Fig. 14. Feature importance ranks for different forecasting models using permutation based feature importance averaged over all meta-models and 30 runs.

feature importance distributions have different variance evaluated over 30 train/test splits. In this case, the feature importance distributions of the neural network based meta-model has higher variance compared to the tree based models. In this small example, we can see that while feature importance ranking might be similar between SHAP and the permutation based method, the raw values can have prominently different distributions.

4.6. Feature importance for explainability of meta-models

This subsection investigates how feature importance values change when the meta-model is fixed and we change feature importance models and forecasting models. Fig. 12 shows the ranking for all the feature importance methods for XGBoost, aggregated over all 30 splits train/test and all forecasting models. In this case, XGBoost was selected because it has multiple methods determining feature importance. From Fig. 12, we can observe large differences between the importance scores assigned by different feature importance methods for the XGBoost model. In general, SHAP and permutation methods usually produce a similar ordering of the features, based on their importance. On the other hand, the feature importance obtained during the building of the XGBoost model that are constructed from the training data differ. Some features, which are ranked lower with SHAP/permutation feature importance, are assigned high importance by the feature importance methods built during XGBoost construction and vice versa. This signals that even when the same model is used, the feature importance method can have a large impact on how the features are ranked.

4.7. Feature importance with fixed feature importance methods

This subsection describes how keeping the feature importance method fixed but changing the meta-model and forecasting model changes feature importance. Fig. 13 the feature rankings obtained using permutation based feature importance for different meta-models. Ranks are averaged over all forecasting methods and over 30 runs. We can observe that tree-based meta-models assign a similar rank to features. The outlier in this case is the neural network meta-model, where the feature order substantially differs from the rest of the meta-models.

Averaging can also be performed across multiple meta-models with the same feature importance method, to show what foresting algorithms are using what features, independent of the meta-model. Fig. 14 shows feature importance ranks for different forecasting algorithms. Observation can be made that some forecasting algorithms rely on different features than others. One example of this are the MLP and RNN forecasting algorithms where the order of features differs compared to the other three forecasting algorithms. Although we do not

Fig. 15. Feature importance averaged across all forecasting algorithms based on the frequency of the time-series.

go into details about what forecasting algorithm uses what features, one possible explanation for this is that some algorithms can have problems on time-series instances where the mean is not zero or where the standard deviation is not constant over time.

4.8. Time-series and forecasting algorithm types

All the time-series in the dataset also have a frequency with which they were collected (hourly, monthly, yearly, ...). When forecasting models are being built, the frequency of the time-series can have a large effect on model selection and further also on the selection of forecasting model hyperparameters. Therefore, Fig. 15 shows SHAP feature importance based on the frequency of a time-series. The frequency of the individual time-series was available as part of the M4 competition. Note that only SHAP values can be used for this purpose, since they are the only ones that give feature importance on an instance level. We can observe that based on the frequency of the time-series, different features are relevant. If we focus on the feature *agg_linear_trend_attr_"stderr"_chunk_len_5_f_agg_"max"*, we can observe that this feature is not assigned high importance values when dealing with hourly time-series. On the other hand, for the time-series collected daily, this is assigned high importance. A conclusion can be made that when performing algorithm selection, it is beneficial to know the frequency with which the time-series was collected.

Fig. 16. Feature importance ranking based on the type of an forecasting algorithm. Types for each of the forecasting algorithms are available in Appendix C.

Similarly to the previous plot where time-series instances are grouped by frequency of observations, a plot can be made by grouping forecasting algorithms by the type of the model (i.e. statistical, machine-learning, . . .). Fig. 16 shows the feature importance based on the forecasting algorithm type (Statistical, Machine Learning, Hybrid, Combination). Feature importance values are obtained by computing the mean over all feature importance and meta-models. Based on the type of the forecasting algorithm the feature importance values differ slightly, but it is important to note that Hybrid category only has one algorithm while the ML one is an average of three algorithms. Due to this small number of samples in two of the categories, drawing a conclusion is not possible.

Lastly, we visualize the similarities between feature importance values that are produced with different meta-models for different forecasting algorithms. Since feature values are high-dimensional vectors, visualizing them is challenging. One of the techniques for visualizing such vectors is t-SNE embeddings (Van der Maaten & Hinton, 2008) where each high-dimensional vector is projected into two-dimensional space with the goal of preserving distances. We select feature importance vectors from nine different forecasting algorithms obtained with four different meta-models. Since we are repeating this 30 times, we embed all 30 vectors for each combination of meta-model and forecasting model. A few interesting observations can be made from the Fig. 17, showing t-SNE embeddings with cosine metric. Even when the meta-model is fixed, the embeddings calculated based on feature importances can have pronounced differences. For example, comparing feature importance embeddings calculated with the RF meta-model for RNN and ARIMA forecasting algorithms, we can notice a large difference in the dispersions of the embeddings, with embeddings for RNN forecasting models being close together for all 30 splits of the data, while ARIMA embeddings are more dispersed. It is also interesting to note that when tree-based models are used, the embeddings are much more closely grouped together compared to the neural network

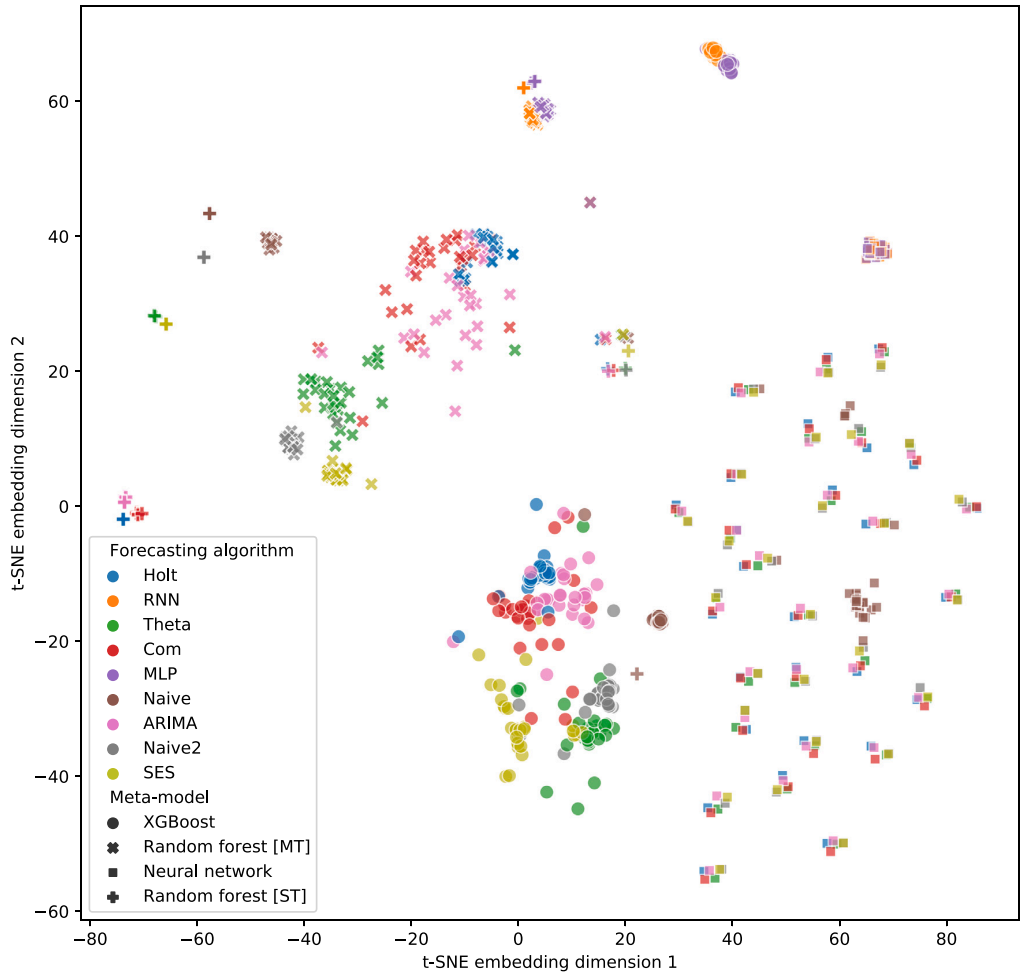


Fig. 17. t-SNE embedding of permutation feature importance values for different forecasting algorithms/meta-learning algorithms.

meta-model. When a neural network is used, the dispersion of embeddings is much larger with clusters of individual forecasting algorithms overlapping.

5. General limitations

This section explains some of the limitations of the methodology for exploring feature importance for predicting the performance of forecasting methods.

The first limitation of the proposed methodology is the selection of the forecasting algorithms. The M4 competition was conducted in 2020 and some of the forecasting methods that were developed later outperform the forecasting methods proposed up to the end of the competition. We limited the scope of the study to only methods that were submitted during the competition. Some later methods that obtain better performance do not report performance for every individual time-series instance included in the competition. Evaluating all the models after 2020 that claim to have improved performance on the M4 dataset would involve constructing all the models together with extensive hyperparameter tuning on models that we are not experts on. For that reason, this study should not serve as an overview of the forecasting models that are available, but as a way of determining feature importance when selecting forecasting algorithms.

Additionally, some of the forecasting methods that were submitted during the competition were trained on multiple time-series instances (aggregated by data frequency). Such is also the case for the three winning forecasting algorithms that first extract patterns from all time-series instances and combine those patterns to forecast future points for an unseen instance. Therefore, it is important to note that when we split the time-series instances in train/test sets and use them to predict performance, some of the forecasting algorithms already introduced information from the test set. Although we realize that such algorithms might bias the results, we still use them in the analysis.

There are numerous libraries for extracting time-series features that could be used as features for a meta-model. We focused only on the libraries that are able to generate features that are interpretable and we, therefore, did not use deep learning-based feature representations since a further analysis of comparing features would not be meaningful. Focusing only on interpretable feature sets, we examined the following possibilities that were most commonly used representations to the best of our knowledge: *hctsa*, *catch22*, *tsfeatures*, *feasts*, *Kats*, *tsfresh*, and *TSFEL*. We ruled out of consideration the *hctsa* feature set since the features are extremely expensive to compute and require a Matlab licence. In Henderson and Fulcher (2021) authors show that features from *catch22*, *feasts*, *Kats*, *tsfeatures*, and *TSFEL* are generally correlated with *hctsa*. We therefore selected *catch22* as a subset of *hctsa* since the features are obtained directly from *hctsa* and are cheaper to compute. *tsfresh* was selected because it includes more features from Fourier transform (i.e. real and imaginary parts, magnitudes, phase angles) while other feature sets include only some summary statistics.

An additional limitation of the study is the selection of error metrics such as sMAPE and MSE. Selecting what metric to use for reporting an error introduces biases. The forecasting error metric was selected based on the following criteria: (i) the sMAPE metric was used in the M4 competition where we obtained performance data. There, the goal was to minimize sMAPE and using another metric for which algorithms were not trained/optimized for might introduce a substantial decrease in performance and thus influence feature importance values, (ii) sMAPE penalizes the relative errors and not the absolute differences which can be large for certain time-series. A more detailed look at the limitations of metrics can be found in Kolassa (2020). However, in future the proposed methodology can be used for any performance metric used as a target, and will further provide insights to the users which features are important depending what is their goal (objective) they want to achieve.

When interpreting feature importance values, one has to also be aware of the limitations of existing methods. Methods that calculate importance during model construction such as RF and XGBoost obtain those features from train data and overfitting the models can distort feature importance values (Strobl, Boulesteix, Zeileis, & Hothorn, 2007). Similar limitations are also present in model agnostic approaches where correlations (Fryer, Strümke, & Nguyen, 2021; Kumar, Venkatasubramanian, Scheidegger, & Friedler, 2020) and model loss (König, Molnar, Bischl, & Grosse-Wentrup, 2021) can have an impact on estimating importance values.

It is also important to note that when calculating Shapley feature importance, only the first 512 time-series instances from the test set were used. This limitation is due to the high computational complexity of SHAP. Similarly, while KNN-regressor appears as one of the meta-models, it was not used for feature importance calculations due to its slow inference time. Due to the time complexity needed to train the meta-models, an exhaustive parameter search was not performed.

Performing this analysis was extremely costly and most of the limitations and trade-offs are therefore related to the available computational power.

6. Conclusion

In this paper, we apply the concept of meta-learning for predicting the performance of time-series forecasting methods, accomplished by training various machine-learning models using features describing characteristics of each time series instance, extracted by the *tsfresh* and *catch22* libraries.

We conduct a comprehensive evaluation of the contribution of the features in the meta-models' predictive performance, using several feature importance methods. A few conclusions follow from the performed analysis. Our results indicate that there are a lot of features that are strongly linearly correlated, however, there is a large subset of *tsfresh* features that are not correlated with any *catch22* features, meaning that catch22 features fail to capture some areas of the time-series landscape. This is also evident from the fact that *tsfresh* features offer better predictive performance compared to the *catch22* features, regardless of the meta-model that is being used, and the fact that there is a negligible difference between the predictive performance obtained by fusing the *catch22* and *tsfresh* feature sets and the performance obtained using only the *tsfresh* features. Looking at the features extracted from log transformed and differenced time-series, we observe high correlations between the features of the log transformed and the raw time series, and lower correlations between the features of the differenced and the raw time series. For *catch22*, fusing the features extracted from the raw, differenced, and log transformed data has a large impact on the predictive performance for all used meta-models, while this difference is lower when using the *tsfresh* features. The analysis of the forecasting algorithm performance revealed strong correlations between the algorithms' performance, meaning that if a single algorithm performs well on a given time-series instance, it is likely that another algorithm will also perform well.

We also find that model-specific features built during the models' construction from the train set can substantially differ from other approaches. The comparison of feature importance between different meta-models using the same feature importance method shows that tree-based models use different features compared to the neural network model. This analysis also reveals that similar feature importance rankings are obtained with SHAP and the permutation importance when the same meta-model is used.

We show that assigning importance to the individual features is a challenging task that depends on the selection of meta-model as well as the method to calculate feature importance values. Regardless of this, there are a few features that are consistently marked as high-importance ones for predicting performance. Such features are for example time-series median, standard deviation, length, autocorrelation at different lags and skewness.

One should also note that feature importance values may vary depending on the forecasting method for which performance is being predicted. Some methods assume that time-series have certain properties such as stationarity and homoscedasticity and when those properties are not met, forecasting performance accuracy can decrease. Features that capture such properties can therefore be assigned high importance for estimating the performance of some forecasting algorithms while being uninformative for others.

The immediate future step for this research is to also include hyperparameters for forecasting algorithms and explore how hyperparameters affect the feature importance. A similar approach can also be extended to multivariate time-series forecasting. Especially interesting would also be applying a similar pipeline for predicting the performance of time-series regression or classification algorithms and assessing the similarity of the importance of features in these tasks, compared to the forecasting algorithm performance prediction task.

CRediT authorship contribution statement

Gašper Petelin: Conceptualization, Methodology, Software, Formal analysis, Writing – original draft, Visualization. **Gjorgjina Cenikj:** Methodology, Formal analysis, Writing – original draft, Visualization, Writing – review & editing, Supervision. **Tome Eftimov:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Supervision, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

Funding in direct support of this work: Slovenian Research Agency: young researcher grant No. PR-11263 to GP, research core fundings No. P2-0098 and project No. N2-0239 to TE, and scholarship by the Ad Futura grant for postgraduate study to GC.

Table C.2
Forecasting algorithms, their type (Statistical, Machine Learning, Hybrid, Combination) and the average sMAPE obtained in the competition.

Rank	For. algo.	sMAPE	Type
0	118	0.114	Hybr.
1	072	0.117	Comb.
2	245	0.117	Comb.
3	069	0.118	Comb.
4	237	0.118	Comb.
5	036	0.119	Comb.
6	238	0.119	Comb.
7	260	0.120	Stat.
8	078	0.120	Comb.
9	039	0.121	Comb.
10	243	0.121	Comb.
11	005	0.121	Stat.
12	132	0.122	Comb.
13	251	0.122	Stat.
14	235	0.123	Stat.
15	Theta	0.123	Stat.
16	223	0.124	Comb.
17	227	0.125	Comb.
18	250	0.125	Comb.
19	239	0.125	Comb.
20	104	0.126	Comb.
21	231	0.126	Comb.
22	Com	0.126	Comb.
23	216	0.127	Comb.
24	Damped	0.127	Stat.
25	ARIMA	0.127	Stat.
26	ETS	0.127	Stat.
27	212	0.129	Comb.
28	211	0.129	ML
29	082	0.129	Stat.
30	236	0.131	Comb.
31	SES	0.131	Stat.
32	248	0.132	Comb.
33	030	0.133	Stat.
34	106	0.135	Stat.
35	Naive2	0.136	Stat.
36	009	0.136	Stat.
37	255	0.136	Stat.
38	252	0.136	Stat.
39	256	0.136	Stat.
40	024	0.138	Stat.
41	Holt	0.138	Stat.
42	234	0.138	Comb.
43	043	0.140	Comb.
44	Naive	0.142	Stat.
45	218	0.143	Stat.
46	169	0.144	Comb.
47	253	0.145	Stat.
48	sNaive	0.147	Stat.
49	241	0.149	Comb.
50	191	0.154	Comb.
51	249	0.159	Stat.
52	070	0.159	Comb.
53	126	0.165	Comb.
54	244	0.166	ML
55	091	0.185	ML
56	258	0.191	Stat.
57	219	0.196	ML
58	RNN	0.212	ML
59	MLP	0.217	ML
60	225	0.261	Stat.

Appendix A. Meta models

Random forest is a ensemble learning model (Ho, 1995) where multiple simpler trees are built and predictions are made by combining the predictions of individual trees. Trees are trained with bootstrap aggregating also know as bagging (Breiman, 1996) where each individual decision tree is trained on randomly sampled training data with replacement. Such an approach means that a specific tree can learn on a training set that contains one or more identical instances. Using such a training procedure ensures that trees that are built are more diverse.

Neural networks are parametric models composed of a series of interconnected units, i.e. neurons, each associated with a corresponding weight. The mapping between the input and output data is learned by iteratively adjusting the weights of each neuron through a process referred to as backpropagation, which attempts to find the minimum of the error function in the weight space using the method of gradient descent (Rojas, 1996). Neural networks generally require larger quantities of training data compared to traditional ML models, and are well suited for modeling complex, non-linear relations between the input and output data (Lancashire, Lemetre, & Ball, 2008; Tu, 1996).

The **K-Nearest Neighbors Algorithm** (KNN) is a non-parametric algorithm which, in order to predict the value of a target variable for a given instance, performs aggregation of the values of the target variable of the K instances which are most similar to the new instance, based on some similarity measure. The algorithm uses instance-based, lazy learning, meaning that it is not trained to learn any weights, but instead uses entire instances to predict the output and the entire computation is performed at prediction time.

Extreme Gradient Boosting (XGBoost) (Chen et al., 2015) uses a principle of combining multiple weak learners to obtain better predictions. It is essentially a decision-tree-based ensemble that uses a gradient boosting framework (Friedman, Hastie, & Tibshirani, 2000) and is designed to be extremely computationally efficient and scalable for large amounts of data. XGBoost works especially well on tabular data (Shwartz-Ziv & Armon, 2022)

Appendix B. Feature importance

Feature importance and feature selection are closely linked and are one of the most studied topics in the field of ML. Selecting informative features can drastically decrease the computational cost of training ML models, improve models' performance and even provide feedback to the users on what features are important and what features are not important or redundant.

Feature importance/selection methods can be divided into multiple groups depending on the principle of how importance is determined. One class of methods are filter feature importance methods where feature importance is determined based on the various statistical tests for their correlation with the predicted variable. Such methods are independent of any ML model and are generally used as a prepossessing step to remove uninformative features (Pearson's Correlation, LDA, ANOVA, Chi-Square). Feature importance/selection can also be determined with so called wrapper methods (Forward Selection, Backward Elimination, Recursive Elimination) model is trained and evaluated on a subset of features. Features are then added or removed based on the performance of the trained model. Our primary goal is not to select a subset of features to determine the importance of all the used features. For that reason we primarily focus on feature importance methods that can either be obtained when model is already trained or that are calculated during the model construction and are commonly known as embedded feature importance approaches.

Permutation feature importance determines the importance of a feature by observing the difference in the model's prediction error when values of the feature in question are randomly shuffled. The feature is considered important if shuffling the values increases the model's error. This method can produce reliable results for features that are not correlated or correlations between features are low. A problem arises when the model being evaluated is trained on correlated features. In this case, randomly shuffling only one feature might not produce the correct importance of a feature, since the model will still have access to the information present in the feature in question through its correlated feature, resulting in a less pronounced change in performance, which may not be indicative of the feature's importance. Using correlated features can decrease the importance of correlated features by splitting the importance between them.

Shapley value Additive feature importance methods use an explanation model, which is an interpretable approximation of the original model, to provide interpretability for complex models, such as ensembles or deep neural networks, which may not be innately interpretable. The explanation model is a linear function of binary variables indicating whether each feature is used in the model. The explanation model then assigns an effect score to each feature, such that summing the effects of all features approximates the output of the original model.

SHapley Additive exPlanation (SHAP) (Lundberg & Lee, 2017) values are a type of additive feature importance methods derived using game theory, which satisfy the desirable properties of local accuracy, missingness, and consistency. The feature scores are assigned based on the change in the expected model prediction when conditioning on that feature and explain how to get from the base value that would be predicted if none of the features are known, to the current output obtained for a particular observation.

Random forest The importance of a particular feature used by a RF model can be determined by observing the decrease in the impurity after splitting the data by the feature in question, in each internal node in the trees. Since the RF model internally uses several trees, the overall feature importance is computed by averaging its importance over all trees, and normalizing the importance scores. One of the drawbacks of this approach is that the importance scores are computed on the training set and do not reflect the usefulness of the features to the generalization of the model on the test set. High importance can therefore be obtained for features that the model uses to overfit on the training data. Furthermore, the approach tends to assign higher importance to numerical features and categorical features with high cardinality, and neglect the importance of correlated features.

XGBoost Similarly to random forest, feature importance can also be obtained during the construction of XGboost model. (List options) 'weight': the number of times a feature is used to split the data across all trees., 'gain': the average gain across all splits the feature is used in., 'cover': the average coverage across all splits the feature is used in., 'total_gain': the total gain across all splits the feature is used in, 'total_cover': the total coverage across all splits the feature is used in.

Appendix C. Forecasting algorithm type

Table C.2 show that types for all of the forecasting algorithms that were submitted as a part of the M4 competition together with the average performance of the algorithm.

References

Bagnall, A., Lines, J., Bostrom, A., Large, J., & Keogh, E. (2017). The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, *31*(3), 606–660.

Borovykh, A., Bohte, S., & Oosterlee, C. W. (2018). Dilated convolutional neural networks for time series forecasting. *Journal of Computational Finance*.

Brazdil, P. B., Giraud-Carrier, C. G., Soares, C., & Vilalta, R. (2009). Metalearning - applications to data mining. In *Cognitive technologies*.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140.

Chaovalit, P., Gangopadhyay, A., Karabatis, G., & Chen, Z. (2011). Discrete wavelet transform-based time series analysis and mining. *ACM Computing Surveys*, *43*(2), 1–37.

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785–794). New York, NY, USA: ACM, http://dx.doi.org/10.1145/2939672.2939785, URL: http://doi.acm.org/10.1145/2939672.2939785.

Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., et al. (2015). Xgboost: extreme gradient boosting. *R Package Version 0.4-2*, *1*(4), 1–4.

Chen, Y., Kang, Y., Chen, Y., & Wang, Z. (2020). Probabilistic forecasting with temporal convolutional neural network. *Neurocomputing*, *399*, 491–501.

Chen, J., Li, K., Rong, H., Bilal, K., Li, K., & Philip, S. Y. (2019). A periodicity-based parallel time series prediction algorithm in cloud computing environments. *Information Sciences*, *496*, 506–537.

Chollet, F., et al. (2015). *Keras*. GitHub, URL: https://github.com/fchollet/keras.

Christ, M., Braun, N., Neuffer, J., & Kempa-Liehr, A. W. (2018). Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, *307*, 72–77.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. ArXiv preprint arXiv:1412.3555.

Cohen-Shapira, N., & Rokach, L. (2021). Automatic selection of clustering algorithms using supervised graph embedding. *Information Sciences*, *577*, 824–851, https://doi.org/10.1016/j.ins.2021.08.028. URL: https://www.sciencedirect.com/science/article/pii/S0020025521008288.

Dama, F., & Sinoquet, C. (2021). Time series analysis and modeling to forecast: a survey. ArXiv preprint arXiv:2104.00164.

Dempster, A., Petitjean, F., & Webb, G. I. (2020). ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, *34*(5), 1454–1495.

Dempster, A., Schmidt, D. F., & Webb, G. I. (2021). Minirocket: A very fast (almost) deterministic transform for time series classification. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining* (pp. 248–257).

Deng, D., Karl, F., Hutter, F., Bischl, B., & Lindauer, M. (2022). Efficient automated deep learning for time series forecasting. ArXiv preprint arXiv:2205.05511.

Eftimov, T., Petelin, G., Cenikj, G., Kostovska, A., Ispirova, G., Korošec, P., et al. (2022). Less is more: Selecting the right benchmarking set of data for time series classification. *Expert Systems with Applications*, *198*, Article 116871.

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The Annals of Statistics*, *28*(2), 337–407.

Fryer, D., Strümke, I., & Nguyen, H. (2021). Shapley values for feature selection: the good, the bad, and the axioms. *IEEE Access*, *9*, 144352–144360.

Fulcher, B. D., Little, M. A., & Jones, N. S. (2013). Highly comparative time-series analysis: the empirical structure of time series and their methods. *Journal of the Royal Society Interface*, *10*(83), Article 20130048.

Gastinger, J., Nicolas, S., Stepić, D., Schmidt, M., & Schülke, A. (2021). A study on ensemble learning for time series forecasting and the need for meta-learning. In *2021 international joint conference on neural networks* (pp. 1–8). IEEE.

Henderson, T., & Fulcher, B. D. (2021). An empirical evaluation of time-series feature sets. In *2021 international conference on data mining workshops* (pp. 1032–1038). IEEE.

Hewamalage, H., Bergmeir, C., & Bandara, K. (2022). Global models for time series forecasting: A simulation study. *Pattern Recognition*, *124*, Article 108441.

Ho, T. K. (1995). Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition (Vol. 1)* (pp. 278–282). IEEE.

Hyndman, R., Kang, Y., Montero-Manso, P., Talagala, T., Wang, E., Yang, Y., et al. (2020). Tsfeatures. URL: https://pypi.org/project/tsfeatures/. (Accessed 28 February 2022).

Hyndman, R. J., & Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting*, *22*(4), 679–688.

Kolassa, S. (2020). Why the "best" point forecast depends on the error or accuracy measure. *International Journal of Forecasting*, *36*(1), 208–211.

König, G., Molnar, C., Bischl, B., & Grosse-Wentrup, M. (2021). Relative feature importance. In *2020 25th international conference on pattern recognition* (pp. 9318–9325). IEEE.

Kumar, I. E., Venkatasubramanian, S., Scheidegger, C., & Friedler, S. (2020). Problems with Shapley-value-based explanations as feature importance measures. In *International conference on machine learning* (pp. 5491–5500). PMLR.

Lancashire, L. J., Lemetre, C., & Ball, G. R. (2008). An introduction to artificial neural networks in bioinformatics–application to complex microarray and mass spectrometry datasets in cancer studies. *Briefings in Bioinformatics*, *10*(3), 315–329. http://dx.doi.org/10.1093/bib/bbp012, https://doi.org/10.1093/bib/bbp012.

Li, Y., Li, K., Chen, C., Zhou, X., Zeng, Z., & Li, K. (2021). Modeling temporal patterns with dilated convolutions for time-series forecasting. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *16*(1), 1–22.

Lim, B., Arık, S. Ö., Loeff, N., & Pfister, T. (2021). Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, *37*(4), 1748–1764.

Lubba, C. H., Sethi, S. S., Knaute, P., Schultz, S. R., Fulcher, B. D., & Jones, N. S. (2019). catch22: Canonical time-series characteristics. *Data Mining and Knowledge Discovery*, *33*(6), 1821–1852.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Proceedings of the 31st international conference on neural information processing systems* (pp. 4768–4777). Red Hook, NY, USA: Curran Associates Inc.

Makridakis, S., & Hibon, M. (1979). Accuracy of forecasting: An empirical investigation. *Journal of the Royal Statistical Society: Series A (General)*, *142*(2), 97–125.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018). The M4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, *34*(4), 802–808.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2020). The M4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, *36*(1), 54–74.

Meade, N. (2000). Evidence for the selection of forecasting methods. *Journal of Forecasting*, *19*(6), 515–535.

Mölder, F., Jablonski, K. P., Letcher, B., Hall, M. B., Tomkins-Tinch, C. H., Sochat, V., et al. (2021). Sustainable data analysis with snakemake. *F1000Research*, *10*.

Montero-Manso, P., Athanasopoulos, G., Hyndman, R. J., & Talagala, T. S. (2020). FFORMA: Feature-based forecast model averaging. *International Journal of Forecasting*, *36*(1), 86–92, M4 Competition. https://doi.org/10.1016/j.ijforecast.2019.02.011. URL: https://www.sciencedirect.com/science/article/pii/S0169207019300895.

Newbold, P., & Granger, C. W. (1974). Experience with forecasting univariate time series and the combination of forecasts. *Journal of the Royal Statistical Society: Series A (General)*, *137*(2), 131–146.

Oreshkin, B. N., Carpov, D., Chapados, N., & Bengio, Y. (2019). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. ArXiv preprint arXiv:1905.10437.

Patterson, K. (2011). An introduction to ARMA models. In *Unit root tests in time series* (pp. 68–122). Springer.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Rojas, R. (1996). *Neural networks: A systematic introduction*. Berlin, Heidelberg: Springer-Verlag.

Ruiz, A. P., Flynn, M., Large, J., Middlehurst, M., & Bagnall, A. (2021). The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, *35*(2), 401–449.

Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, *36*(3), 1181–1191.

Salisu, M., Abdulrahman, S., Adamu, A., Ado, Y., & Rilwan, A. (2017). An overview of the algorithm selection problem. *International Journal of Computer (IJC)*.

Shwartz-Ziv, R., & Armon, A. (2022). Tabular data: Deep learning is not all you need. *Information Fusion*, *81*, 84–90.

Smyl, S. (2020). A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, *36*(1), 75–85.

Srinivasan, V., Eswaran, C., et al. (2005). Artificial neural network based epileptic detection using time-domain and frequency-domain features. *Journal of Medical Systems*, *29*(6), 647–660.

Strobl, C., Boulesteix, A.-L., Zeileis, A., & Hothorn, T. (2007). Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, *8*(1), 1–21.

Talagala, T. S., Hyndman, R. J., & Athanasopoulos, G. (2018). *Meta-learning how to forecast time series*: *Monash econometrics and business statistics working papers 6/18*, Monash University, Department of Econometrics and Business Statistics, URL: https://ideas.repec.org/p/msh/ebswps/2018-6.html.

Talagala, T. S., Li, F., & Kang, Y. (2021). FFORMPP: Feature-based forecast model performance prediction. *International Journal of Forecasting*, https://doi.org/10.1016/j.ijforecast.2021.07.002. URL: https://www.sciencedirect.com/science/article/pii/S0169207021001138.

Tu, J. V. (1996). Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology*, *49*(11), 1225–1231. http://dx.doi.org/10.1016/s0895-4356(96)00002-9, https://doi.org/10.1016/s0895-4356(96)00002-9.

Tyrrell, B. (2020). 'Algorithm-performance personas' for siamese meta-learning and automated algorithm selection.

Vaiciukynas, E., Danenas, P., Kontrimas, V., & Butleris, R. (2021). Two-step meta-learning for time-series forecasting ensemble. *IEEE Access*, *9*, 62687–62696.

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*(11).

Van Greunen, J., Heymans, A., Van Heerden, C., & Van Vuuren, G. (2014). The prominence of stationarity in time series forecasting. *Studies in Economics and Econometrics*, *38*(1), 1–16.

Vanschoren, J. (2019). Meta-learning. In *Automated machine learning* (pp. 35–61). Springer, Cham.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.

3.1.1 Discussion

This study explores the use of meta-learning to predict the performance of time series forecasting algorithms, with a focus on explaining the importance of various time series features. By leveraging feature extraction libraries such as *tsfresh* and *catch22*, and employing diverse machine learning models, we evaluated the impact of different features on forecasting performance. Our results show that although some features are highly correlated, the *tsfresh* features provide more comprehensive information compared to *catch22*, resulting in better performance in forecasting algorithm predictions. Notably, features such as the time series median, standard deviation, and autocorrelation consistently emerge as important predictors across various meta-models. The main contribution of this work is, therefore, an in-depth analysis of the features that are important for predicting the performance of forecasting algorithms.

Moreover, hypothesis **H3**, defined as *Meta-models for selecting time series forecasting algorithms are able to be explained using feature scoring techniques*, is confirmed. We first build meta-models that outperform the proposed baseline meta-models in terms of accuracy when predicting the performance of forecasting algorithms based on a specific set of features. Once this is achieved, feature importance analysis is used to demonstrate the influence of specific time series characteristics on algorithm performance prediction. Additionally, we observe similar feature importance scores across different meta-models and feature scoring techniques, highlighting the consistency of the identified important features, which is a crucial step in confirming the hypothesis.

3.2 Benchmarking the Robustness of Time Series Classification Algorithms Against Common Distortions

When working with time series data, one of the common tasks is TSC. To achieve this, multiple approaches exist, ranging from simpler distance-based methods to complex deep-learning techniques. Due to the large variety of such approaches, selecting the most suitable one is not always straightforward. While there are benchmarks that evaluate the accuracy of TSC techniques, what is often even more relevant is their robustness to various naturally occurring distortions and how these algorithms behave in the presence of such distortions.

This section evaluates existing TSC algorithms to analyze how they perform when distortions are introduced during inference and how this impacts the accuracy of classifiers. This study investigates hypothesis **H4**. We approach this by simulating the most frequent types of distortions and empirically evaluating the robustness of different TSC algorithms.

We demonstrate that existing TSC algorithms exhibit significant variations in their robustness when exposed to distortions. While some state-of-the-art algorithms may perform exceptionally well on undistorted data, their performance can deteriorate rapidly when encountering distortions. Moreover, algorithms with similar accuracy on undistorted data can show vastly different levels of robustness when exposed to distortions. This finding highlights that selecting a TSC algorithm cannot rely solely on accuracy benchmarks for clean data but must also consider performance under realistic conditions where distortions are present. This insight is crucial for practical applications, where robustness to imperfections in the data is often just as important as, if not more important than, raw accuracy.

3.2.1 Introduction

The domain of time series analysis, and more specifically time series classification (TSC), has attracted considerable attention as vast amounts of temporal data are now being gathered across multiple fields. Due to the sequential nature of time series data, it inherently presents unique challenges for classification. Consequently, researchers have investigated and assessed a variety of classification techniques (Middlehurst et al., 2023). With an increasing number of TSC algorithms emerging, it becomes crucial to clearly delineate the advantages and limitations of each method across different datasets and scenarios. This comprehensive evaluation not only reveals which algorithms perform best under specific conditions, but also uncovers areas where further improvements are necessary or where certain approaches particularly excel.

In practical scenarios, TSC algorithms frequently face both anticipated and unforeseen distortions in the input data, such as noise introduction or missing values. These distortions can result from issues like sensor inaccuracies (Q. Liu et al., 2015), system glitches (Ribeiro & de Castro, 2021), or even deliberate data manipulation (Belkhouja & Doppa, 2022; Ismail Fawaz et al., 2019a). Evaluating and understanding how these algorithms cope with such challenges is critical for several reasons. Firstly, insights into their responses can uncover new perspectives on current methodologies, which is valuable for refining existing algorithms and developing novel approaches, as demonstrated in fields like natural language processing (Chang et al., 2021), computer vision (Naseer et al., 2021), time series analysis (Petelin et al., 2023), and optimization (Nomura et al., 2023). Secondly, this understanding is vital for constructing robust systems that depend on various TSC algorithms (Wenninger et al., 2019). In real-world applications, practitioners might need to balance performance with robustness—sometimes accepting a minor reduction in performance to ensure the model remains resilient against data distortions. By examining these interactions, we aim to highlight the robustness and sensitivity of TSC methods.

In this study, we assess the robustness of TSC algorithms when faced with data distortions. Our primary objective is to determine: How effectively do current TSC algorithms handle various distortion types? We conduct an in-depth analysis of both accuracy and resilience across a range of TSC methods using diverse datasets from the UCR repository (Dau et al., 2018). Specifically, we examine how these algorithms perform under different distortions—including noise, time/magnitude warping, and permutations—each applied with escalating intensity. Our investigation serves two purposes: firstly, to explore how these distortions affect algorithm accuracy, and secondly, to quantify the rate at which performance decreases with each distortion. By determining whether some algorithms demonstrate superior robustness against particular distortions across multiple datasets, our findings aim to provide valuable insights into their comparative performance. Ultimately, answering this question can assist practitioners in making more informed algorithm selections, reducing reliance on trial and error.

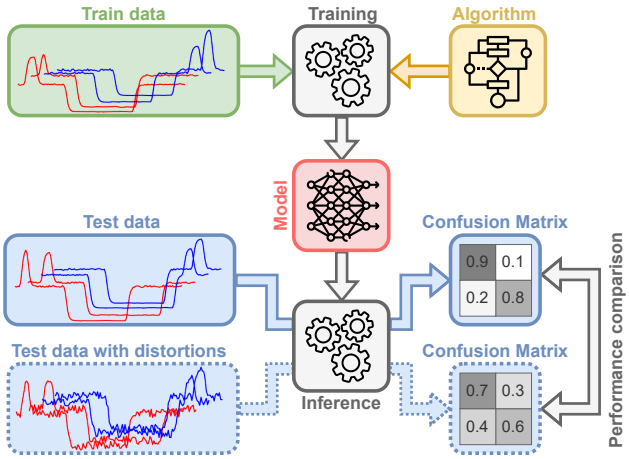


Figure 3.1: Evaluation of the robustness of various time series classification algorithms to different distortions. A trained TSC algorithm is evaluated on both a standard test set and a version of the test set with added distortions to measure the reduction in accuracy resulting from these distortions.

To evaluate how TSC algorithms withstand distortions, we adopt the methodology depicted in Figure 3.1. Initially, the algorithms are trained on an undistorted training set. Subsequently, a designated distortion is introduced into the test set, and the already-trained TSC model is employed to classify the altered data. This process enables us to determine the reduction in classification accuracy attributable to the distortion, thereby pinpointing the threshold at which performance starts to decrease and quantifying the extent of this drop. While recent studies have predominantly examined the robustness of TSC algorithms to adversarial attacks (Ismail Fawaz et al., 2019a; Karim et al., 2020), the degradation in performance due to general data distortions and interferences has received comparatively little attention. This research seeks to address that gap by assessing how TSC algorithms perform when exposed to common data distortions and perturbations.

Our contribution: We offer an extensive study of TSC algorithms, providing a fresh perspective on their resilience to a variety of time series distortions. We assess a range of algorithms across 128 UCR datasets to uncover performance correlations on clean data. Following this, we examine the impact of specific distortions on selected algorithms. In doing so, we identify notable patterns in robustness; algorithms that are typically seen as high-performing often show reduced robustness compared to their simpler counterparts. More-

over, we reveal that algorithms exhibit different sensitivities to local distortions—those affecting only a small segment, like noise—and to global distortions, such as permutations that alter the overall structure of the time series. These findings underscore the need to consider the unique characteristics of both the data and the distortions when evaluating the effectiveness of TSC algorithms, highlighting that an algorithm's superiority is largely dependent on the expected properties of the dataset it is applied.

Outline: This study is organized as follows: Section 3.2.2 describes the existing approaches to perform algorithm classification. In Section 3.2.3, we introduce the methodology for evaluating the existing TSC algorithms, followed by the results in Section 3.2.4 and the discussion in Section 3.2.5. Finally, in Section 3.2.6 we conclude our work and propose future research directions.

3.2.2 Related Work

In this section, we first present a review of time series classification algorithms and associated techniques, followed by an overview of techniques and methodologies exploring the robustness of machine learning algorithms in various domains.

3.2.2.1 Time Series Classification Algorithms

In the field of TSC, researchers have investigated a wide range of methods to address the unique challenges that come with temporal data. The transition from traditional machine learning techniques (Fulcher & Jones, 2014) to advanced deep learning models (Bagnall et al., 2017; Middlehurst et al., 2023) emphasizes the growing importance of understanding the strengths and limitations of these diverse approaches. Although many algorithms exist for classifying time series data, they can generally be organized into a few broad categories, as discussed in studies like (Faouzi, 2022) and (Gupta et al., 2020).

Nearest-neighbor methods (Bagnall & Lines, 2014) serve as one of the cornerstones of TSC. These approaches classify a time series by measuring its similarity to previously observed examples, relying on a similarity metric that quantifies how close data points are. Various metrics have been proposed, each yielding different levels of success (Abanda et al., 2019). Shapelet-based TSC algorithms (Hills et al., 2014; Li et al., 2020) represent another key category. They work by extracting shapelets—subsequences that are highly indicative of a particular class (Ye & Keogh, 2009)—and then comparing these shapelets to segments within a time series to determine similarity, effectively basing classification on the presence or absence of these distinctive patterns. Dictionary-based methods take a different approach by converting time series data into sequences of symbols. These algorithms extract words using a sliding window technique and count the frequency of each word against a predefined dictionary, classifying the series based on the distribution of these symbolic representations (Large et al., 2019). Deep learning-based TSC algorithms, perhaps the most rapidly evolving group, employ neural networks to automatically learn feature representations directly from raw data, thereby eliminating the need for manual feature engineering. A detailed overview of recent advances in this area can be found in (Ismail Fawaz et al., 2019b). In addition to these groups, there are algorithms that do not neatly fit into any single category. For instance, ensemble approaches (Bagnall et al., 2012; Bai et al., 2021; Ismail Fawaz et al., 2019c) combine multiple individual algorithms—sometimes drawing from several of the above categories—with the HIVE-COTE family of algorithms (Lines et al., 2018; Middlehurst et al., 2021) being a notable example.

3.2.2.2 Robustness of Machine Learning Algorithms Across Different Domains

The robustness of machine learning algorithms in handling various distortions and perturbations in data has emerged as a key area of research across many fields (Tocchetti et al., 2022). A robust algorithm is characterized by its ability to deliver accurate predictions while effectively managing outliers, perturbations, and shifts in data distribution without a significant drop in performance. This quality is essential for ensuring consistency and reliability in practical applications, particularly in critical sectors such as healthcare (Qayyum et al., 2020), finance (Zeng & Yan, 2008), and security (Yu et al., 2022), where data variability and unpredictability are common. Nonetheless, it is important to note that there is often a trade-off between robustness and accuracy, as algorithms engineered for greater robustness may sometimes sacrifice precision (Tsipras et al., 2018).

Various techniques can improve the robustness of machine learning algorithms under diverse conditions. For instance, adversarial training incorporates adversarial examples to defend against attacks (Goodfellow et al., 2014), while data augmentation (Cubuk et al., 2019) enlarges the training set with modified samples to boost generalization. Regularization helps prevent overfitting, enhancing resistance to input variations (Ying, 2019), and ensemble methods combine multiple models to reduce individual errors (Dietterich, 2000). These strategies are also widely used in TSC (Alemany & Pissinou, 2020; Ding et al., 2023; Iglesias et al., 2023; Wen et al., 2020), collectively contributing to more robust and effective systems.

Although such techniques are crucial for enhancing the robustness and generalizability of ML algorithms, it is important to understand their limitations. Expertise is often required to determine when to apply specific techniques and when not to, as indiscriminate application can lead to adverse effects. Consequently, practitioners often face challenges in choosing the appropriate ML algorithm and deciding how to modify data or fine-tune models to improve performance and robustness. Additionally, when building models, the techniques' relevance depends on the context of the specific ML algorithm used. For instance, in the domain of time series classification, certain augmentations are only applicable in specific cases (Batista et al., 2014).

Robustness to distortions has been widely studied across various domains, providing insights into algorithm performance and guiding practitioners' expectations. In computer vision, for instance, studies such as (Kamann & Rother, 2021) have examined how deep neural networks cope with common corruptions like blur, noise, contrast, and compression, with extensions to tasks such as segmentation (Kamann & Rother, 2021) and spatiotemporal modeling (Yi et al., 2021). Additionally, works like (Geirhos et al., 2018; Naseer et al., 2021) reveal that convolutional neural networks often show a stronger texture bias than transformers. In natural language processing, research has explored model performance under simple distortions—such as added, dropped, paraphrased, or swapped words, and typographical errors (Belinkov & Bisk, 2017; Jia et al., 2019; Pruthi et al., 2019)—as well as under more complex adversarial attacks (Zhang et al., 2020). These investigations collectively enhance our understanding of how modeling techniques respond to various distortions.

Unfortunately, in the domain of TSC, the area of robustness has received relatively little attention. Some works have investigated the behavior of features and their invariance (Grabocka, 2016; Schäfer, 2015). Additionally, multiple works have investigated the robustness of TSC algorithms to adversarial attacks (Abdu-Aguye et al., 2020; Ding et al., 2023; Ismail Fawaz et al., 2019a). However, to the best of our knowledge, there has been almost no exploration of robustness for TSC algorithms to commonly encountered distortions that occur in real-world applications and are not the product of adversarial

attacks.

3.2.3 Methodology

In the introduction, we outline the primary research question: How robust are TSC algorithms to common distortions? Additionally, we explore performance similarities among TSC algorithms across the UCR dataset. To investigate these issues, we present the methodological approach illustrated in Figure 3.1:

1. Divide the data into train and test sets.
2. Train TSC algorithms on the train set as is without applying any of the distortions before model construction.
3. Apply a single distortion (such as adding noise or spikes) to the test set and then use the pre-trained TSC method to perform classification.
4. Investigate the decrease in accuracy that results from applying distortions.

This methodology allows us, for each dataset and TSC algorithm, to accurately measure the effect of distortions on performance, identifying the threshold at which accuracy begins to decline. By analyzing how various algorithms respond to these transformations, we gain deeper insights into their robustness and adaptability. Applied to all TSC algorithms in Section 3.2.3.3, this approach evaluates performance across all 128 datasets from the UCR repository.

For completeness, we also examine the performance of TSC algorithms on undistorted time series, demonstrating how they perform when trained and tested on data from the same distribution. Note that the comparison of various algorithms has been extensively explored in previous studies (Bagnall et al., 2017; Ismail Fawaz et al., 2019b; Middlehurst et al., 2023), which offer a comprehensive overview of state-of-the-art TSC methods. Our focus here is on identifying correlations between the performances of different TSC algorithms, as this can reveal underlying similarities among them.

3.2.3.1 Distortion

This section outlines the time series distortions examined in our study. Although the literature documents a wide range of distortions, we concentrated on those most commonly applied in practice. We selected these distortions primarily because they frequently occur in real-world TSC applications (Fishburn et al., 2019; Örnhag et al., 2022). For a more comprehensive list of potential distortions, augmentations, and transformations, refer to (Iwana & Uchida, 2021a; Wen et al., 2020). In our research, we evaluated the following distortions, as illustrated in Figure 3.2:

Jitter: The original time series is augmented by introducing random noise drawn from a Gaussian distribution with a mean of 0 and standard deviation σ, as described in (Iwana & Uchida, 2021b; Um et al., 2017). This type of perturbation often appears in practical domains, including digital signal processing (Vasilescu, 2005), exoplanet detection (Shallue & Vanderburg, 2018), and financial modeling (B. Liu & Cheng, 2024).

Random walk: A random walk-based time series is generated by drawing step sizes from a Gaussian distribution with standard deviation σ. This random walk is subsequently combined with the original time series, resulting in a distorted version. Analogous to jitter, this distortion frequently arises in practical scenarios, such as sensor data exhibiting drift (Örnhag et al., 2022; Wu et al., 2020), financial price modeling (Adhikari & Agrawal, 2014), and biomedical signal analysis (Shah & Seghouane, 2014).

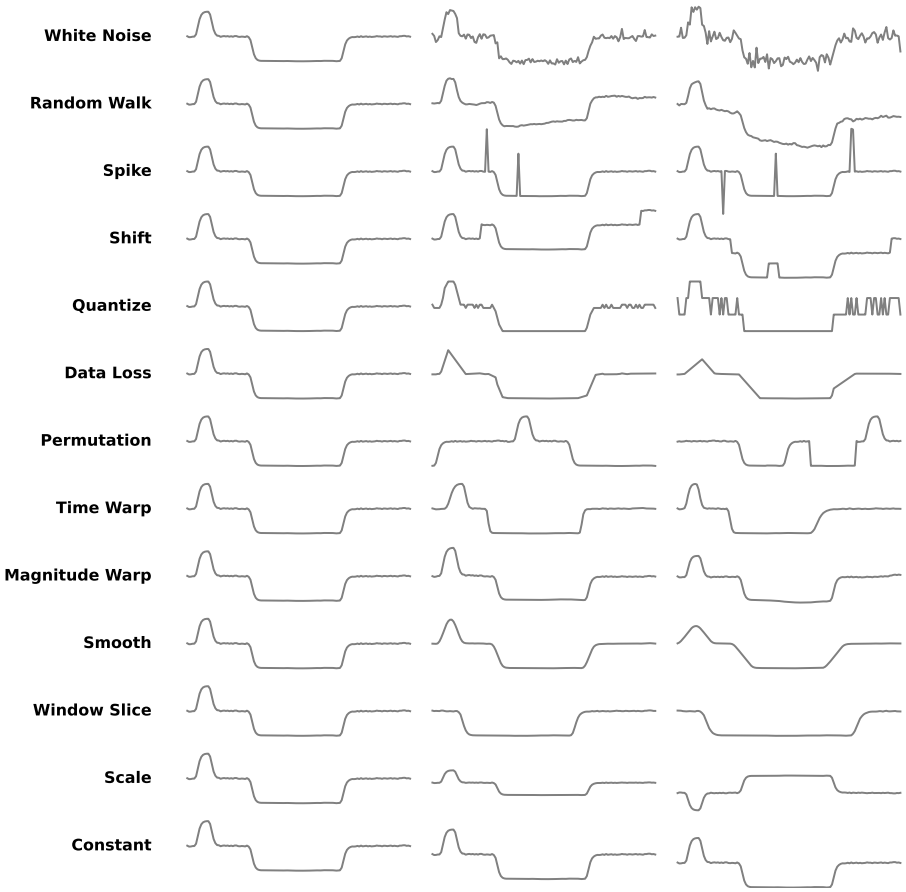


Figure 3.2: A single time series from the BME dataset in the UCR archive is depicted. In the left column, the original, undistorted time series is displayed. The middle column shows the time series with a moderate level of distortion added, while the right column presents the time series with a higher magnitude of distortions.

Shift: The time series is randomly shifted at multiple points, each shift corresponding to the standard deviation computed from the original time series, and repeated n times. Such distortion commonly arises in biomedical applications (Perpetuini et al., 2021), particularly with EEG and fNIRS technologies (Hossain et al., 2022), where motion-induced artifacts alter the signals. Similar distortions can also be observed as sudden shifts in financial markets driven by unexpected events (Au Yeung et al., 2020), or in reliability analysis (Martínez-Arellano et al., 2019), where equipment degradation causes abrupt signal changes.

Spike: Similar to shifting, the original time series is modified by introducing spikes at random positions. Each spike has a magnitude equal to three times the standard deviation of the original time series and lasts exactly one timestamp. The total number of spikes is denoted by n. This form of distortion closely resembles shifts and is therefore prevalent in similar contexts.

Window slicing: The fundamental idea behind slicing involves enhancing the dataset by removing time steps from the edges of the pattern. In our context, a single parameter, denoted as α ($0 < \alpha < 1$), governs the proportion of the time series that is trimmed. When α is set to 0.5, half of the time series (from beginning, end, or both) is removed, whereas an α of 1 preserves the original time series without any alterations. Window slicing often occurs either as a preprocessing step, where long time series are split into shorter ones and

processed individually, or in early time series classification (Faouzi, 2022), where the full window is not always known. In both cases, the discriminative features of a time series might not be fully visible to the model, making classification more challenging.

Scale: This distortion multiplies an entire time series by a constant c. When $c = 1$, the series remains unchanged; $c = -1$ flips it; and $c = 0$ reduces it to zeros, removing all information. Although time series are typically rescaled before classification, this distortion is used to assess how TSC techniques handle improperly scaled data. It commonly occurs in practice, such as in human activity recognition due to orientation variability or device misalignment (Gil-Martín et al., 2023; Golestani & Moghaddam, 2020), and in ECG systems because of electrode-skin impedance variability (Showkat et al., 2023).

Constant: This distortion shifts a time series by adding a constant value to every point. In our study, a distortion of magnitude σ means a random scalar—sampled from a normal distribution with mean zero and standard deviation σ—is added to the series. For instance, with $\sigma = 5$, each series is shifted by a random value drawn from a normal distribution with a standard deviation of 5. Although constant drift is common in applications like fNIRS-related tasks (Hossain et al., 2022) (and such series are usually normalized to have zero mean), we include this distortion to assess the TSC algorithm's ability to mitigate its effects.

Permutations: This distortion reorders segments within a time series to create a new pattern. In our approach, the series is divided into n equal-sized segments that are then rearranged, with the rule that a segment cannot return to its original position. This transformation, as proposed in (Um et al., 2017), simulates errors common in IoT data transmission (Moore et al., 2020; Shukla et al., 2023) and tests how TSC methods cope when the global structure of the time series is disrupted.

Quantization: This distortion rounds each value in the time series from a continuous range to a discrete set. In our scenario, values are divided into n distinct buckets based on the time series' minimum and maximum. This process is particularly relevant when transmitting or storing large volumes of data, as commonly encountered in IoT applications (Chiarot & Silvestri, 2023; de Oliveira et al., 2023).

Data Loss: This distortion randomly removes points from a time series. Prior to classification, missing segments are imputed using the method outlined in Section 3.2.3.4. The proportion of dropped values is represented by α, where $0 \leq \alpha \leq 1$. Data loss commonly occurs during data collection (Silva et al., 2012), transmission (Velasco-Gallego & Lazakis, 2020), or due to sensor failures (H. M. Ahmed et al., 2022).

Magnitude Warp: This distortion, as proposed in (Um et al., 2017), alters the magnitude of time series by introducing small variations that randomly increase or decrease sections. We use an implementation from (Iwana & Uchida, 2021a) where each time series is scaled by a curve generated using cubic spline interpolation with 6 fixed knots and a variable distortion magnitude σ. This distortion is common in sensor data due to factors such as aging and temperature fluctuations (Anik et al., 2021; Teh et al., 2020), and in electrophysiological data analysis (Al-Ayyad et al., 2023), where variability arises from differences in equipment and individual physiological states.

Time Warp: This distortion modifies the temporal dimension of a time series. We use the implementation from (Iwana & Uchida, 2021a) with 6 knots and a temporal distortion magnitude σ. Time warping commonly occurs in human activity recognition, as people perform actions at different speeds (Lara & Labrador, 2012), in manufacturing processes where machinery speeds vary (Pittino et al., 2020), and in speech recognition due to varied speaking tempos (Sood & Jain, 2021).

Smoothing: This operation smooths a time series by convolving it with a kernel of length k, where each element is $1/k$. Padding is applied using the edge values to

facilitate convolution. A larger k results in greater smoothing. While this distortion is rare and primarily used to benchmark TSC algorithms, it can occur in scenarios such as mechanical (Angolkar et al., 1992) and hardware component degradation (Albarbar & Teay, 2017).

It is important to note that not every distortion is applicable to all datasets. For example, in an accelerometer dataset for human activity classification, such as the `CricketX` dataset, distortions like jittering or random walk may occur due to sensor drift and noise variations, as well as differences in sensor orientation (which could lead to a flipped time series). In contrast, flipping the time series of a spectrograph is unlikely in real-world scenarios since it could result in invalid data. Moreover, some distortions like flipping might change the class of the time series, meaning a flipped instance could belong to a different class. Despite these considerations, we have applied each distortion across all UCR datasets to comprehensively assess their impact.

3.2.3.2 Datasets

Historically, TSC algorithms were often assessed on a single—or even an artificially generated—dataset to showcase their capabilities. To overcome this limitation, the University of California, Riverside introduced the UCR time series classification repository (Dau et al., 2018), now containing 128 diverse datasets with varying numbers of classes and time series lengths. However, comparing classification accuracies across these datasets is challenging due to differences in class counts and inherent difficulty levels, where some datasets allow near-perfect accuracy while others pose significant challenges.

3.2.3.3 Time Series Classification Algorithms

This section briefly describes the selected algorithms used in our research. We curated a diverse subset of TSC methods based on their mechanisms (feature-based, deep-learning-based, and distance-based), classification accuracy, training speed, community prevalence, and code availability. The algorithms are as follows:

`hivecotev2` (Middlehurst et al., 2021): An ensemble TSC algorithm that integrates classifiers like shapelet and dictionary methods via a hierarchical voting mechanism. It improves computational efficiency for larger datasets and is widely regarded as one of the top TSC methods.

`rocket` (Dempster et al., 2020): This method transforms time series using numerous random 1D convolutional kernels that capture features such as shape, frequency, and variance, then trains a classifier on the transformed features.

`drcif` (Middlehurst et al., 2021): A component of the `hivecotev2` framework that extends the CIF algorithm by representing randomly selected intervals with `catch22` features and employing an ensemble of decision trees for classification.

`inception` (Ismail Fawaz et al., 2020): Utilizes stacked inception modules with convolutional layers of varying filter sizes and residual connections to capture temporal features at multiple scales while mitigating the vanishing gradient problem.

`tsfresh` (Christ et al., 2018): Automatically extracts a broad range of hand-crafted features (e.g., statistical measures, autocorrelations) from time series data and uses a random forest for classification.

`weasel` (Schäfer & Leser, 2017): Converts time series into a bag-of-patterns representation by extracting sub-sequences from overlapping windows and counting their occurrences to form fixed-length feature vectors for classification.

`tsforest` (Deng et al., 2013): An ensemble of decision trees that builds each tree from random time series segments, extracting features like mean, variance, and slope, with final

predictions determined by majority voting.

`catch22` (Lubba et al., 2019): Provides 22 selected features that capture key time series dynamics (e.g., autocorrelation, distribution). Like `rocket` and `tsfresh`, it requires a separate classifier—in our case, a random forest—to perform classification.

`eknn` (Abanda et al., 2019): A distance-based method that uses the Euclidean version of the 1-NN algorithm to classify time series by comparing them based on a chosen distance metric, offering a faster alternative to methods like DTW.

`cnn` (B. Zhao et al., 2017): Applies convolutional layers to capture local dependencies and pooling layers to reduce dimensionality, followed by fully connected layers for classification. This approach was among the first CNN-based TSC methods and has evolved into more advanced architectures like `inception`.

`baseline`: A simple model that always predicts the majority class, serving as a reference point for evaluating other algorithms.

3.2.3.4 Preprocessing Steps

We apply a standard scaler (z-score normalization) by subtracting the mean and dividing by the standard deviation to achieve a zero mean and unit variance (Bhanja & Das, 2018). Since many UCR datasets contain gaps, we use linear interpolation (DeLang et al., 2022) to impute missing values, estimating them via a straight-line trend between neighboring points.

3.2.3.5 Statistical Comparison

Various techniques exist for comparing classifiers across multiple datasets (Demšar, 2006; Garcia & Herrera, 2008). In our work, we adopt an approach similar to that in (Ismail Fawaz et al., 2019b; Middlehurst et al., 2023). We use the critical difference diagram (Demšar, 2006) to visualize average ranks, grouping classifiers into cliques that indicate no statistically significant differences. Pairwise Wilcoxon signed-rank tests are performed, and cliques are formed using the Holm correction for multiple comparisons (Benavoli et al., 2016).

3.2.3.6 Measuring Robustness to Distortions

To evaluate how much a time series can be altered before a TSC algorithm's performance begins to decline, we introduce the Distortion Error (*DE*) metric, as described in Eq 3.1. A concept akin to *DE* has been explored in computer vision to assess the effects of distortions, as highlighted in (Hendrycks & Dietterich, 2019). The primary difference in our approach is the comprehensive evaluation of multiple TSC algorithms across diverse datasets. Since raw classification accuracies cannot be directly compared across datasets, it is essential to design a metric that accounts for inherent variations in baseline accuracies. To achieve this, we normalize classification accuracies in the computation of *DE*, ensuring that the best-performing TSC algorithm on the original, undistorted dataset attains a *DE* score of 1, while a baseline classifier that always predicts the majority class is assigned a score of 0. The scores of all other classifiers are then scaled within this range. The *DE* is formally defined as follows:

$$DE_{a,l} = \frac{\sum_{d \in \mathcal{D}} DE_{a,l,d}}{|\mathcal{D}|} \tag{3.1}$$

$$DE_{a,l,d} = \frac{CA_{a,l,d} - CA_{baseline,l,d}}{\max_{a \in \mathcal{A}} CA_{a,clean,d} - CA_{baseline,clean,d} + \epsilon} \tag{3.2}$$

$$CA_{a,l,d} = \frac{\sum_{r \in \mathcal{R}} CA_{a,l,r,d}}{|\mathcal{R}|} \tag{3.3}$$

The variable a represents a specific TSC algorithm from the set of all algorithms $\mathcal{A}$ for which the *DE* is computed. The variable l denotes the severity level of a given distortion, while r corresponds to an independent execution of an algorithm from the set of runs $\mathcal{R}$. Additionally, d represents a dataset from the collection $\mathcal{D}$ on which the method was applied. The classification accuracy of algorithm a on dataset d under distortion level l, with a particular initialization seed r, is denoted as $CA_{a,l,r,d}$. These accuracy values are then averaged to obtain $CA_{a,l,d}$, which represents the mean classification accuracy of an algorithm on a given dataset under the specified distortion. Finally, these values are aggregated into $DE_{c,l}$ to quantify the reduction in accuracy for a specific distortion level. It is important to note that, for certain datasets, some algorithms fail to outperform the baseline even on the original, undistorted data. To prevent division by zero in such cases, a small constant ϵ of 0.01 is added to the denominator.

3.2.4 Results

Here, we present the results from our analysis, which are divided into the following sections. We start with the experimental setup in 3.2.4.1 and algorithm comparison in 3.2.4.2, followed by sections that investigate the robustness in 3.2.4.3.

3.2.4.1 Experimental Setup

This section outlines our experimental setup and TSC algorithm configurations. Unless noted otherwise, all algorithms used their default `sktime` settings. Due to the high computational cost of retraining, we avoided extensive hyperparameter tuning, although for resource-intensive algorithms—especially those processing long or numerous time series—we adjusted hyperparameters to reduce computational load. Table 3.1 lists the key hyperparameters, and all reported classification accuracies are averaged over ten independent runs.

To conduct the experiments we used the following libraries: `scikit-learn` (Pedregosa et al., 2011), `sktime` (Löning et al., 2019), `aeon` (Middlehurst, Ismail-Fawaz, et al., 2024), TensorFlow (Abadi et al., 2015) and PyTorch (Paszke et al., 2019). The experiments were conducted on a machine running the Ubuntu operating system, powered by an Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz and equipped with 1 TB of RAM. The associated code is accessible at repository https://gitlab.com/gapi.petelin/ts-rep.

3.2.4.2 TSC Algorithm Comparison

In Section 3.2.3.3, several commonly used TSC algorithms are introduced. Before assessing their robustness, it is essential to first examine how these algorithms compare in terms of accuracy and their relative rankings on the UCR datasets. Figure 3.3 illustrates the average rankings of TSC algorithms across all datasets in the UCR repository, with additional horizontal bars highlighting statistically significant differences. Notably, there are considerable ranking differences among certain algorithms. Using the predefined hyperparameters, `hivecotev2` achieves the highest ranking with an average rank of 2.49, significantly outperforming all other algorithms. This is followed by the `rocket` classifier, which holds a rank of 2.96. Other strong performers include `drcif` and `inception`. On the other hand, feature-based and distance-based algorithms, along with `cnn`, rank lower in terms of performance. As expected, the majority class classifier performs the worst,

Table 3.1: The main hyperparameters of TSC algorithms used in our study.

Model	Hyperparamater	Value
`hivecotev2`	*time_limit_in_minutes*	10
	n_jobs	10
`rocket`	*num_kernels*	10000
`drcif`	*n_estimators*	200
`inception`	*n_filters*	32
	depth	6
`tsfresh (+rf)`	*default_fc_parameters*	efficient
	n_estimators	100
`weasel`	*window_inc*	2
	alphabet_size	2
	anova	True
`tsforest`	*min_interval*	3
	n_estimators	200
`catch22 (+rf)`	*replace_nans*	true
	n_estimators	100
`eknn`	*n_neighbors*	1
	distance	euclidean
`cnn`	*kernel_size*	7
	n_conv_layers	2
`baseline`	*strategy*	prior

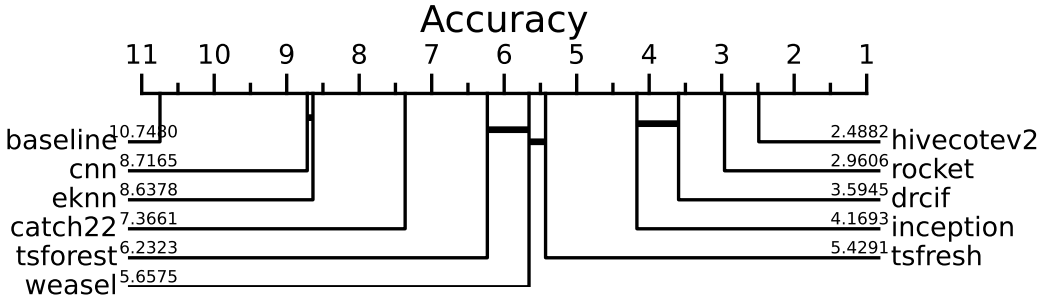


Figure 3.3: Critical difference diagram shows the average ranks of TSC algorithms employed in this study.

with an average rank of 10.74. These results align with the findings of (Middlehurst et al., 2023), where both `hivecotev2` and `rocket` were identified as top-performing classifiers. However, it is important to note that the TSC algorithms examined in this study were used with default hyperparameters rather than optimized settings. This decision was made due to the high computational cost of tuning certain TSC algorithms. It is worth emphasizing that hyperparameter tuning, particularly dataset-specific optimization, can further enhance classification performance.

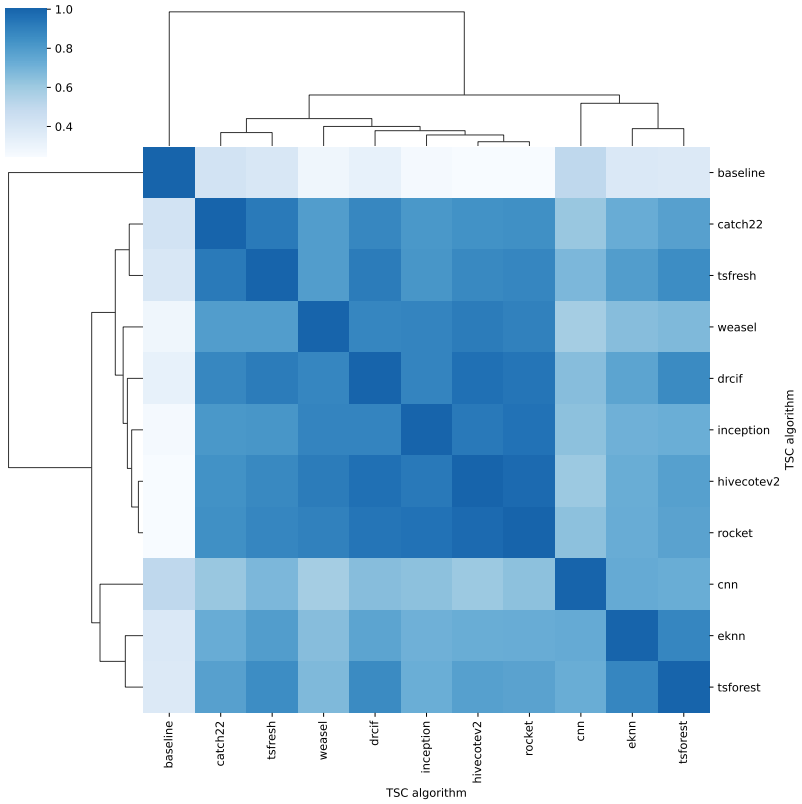


Figure 3.4: Spearman correlation among TSC algorithms based on classification accuracies obtained on UCR datasets.

Comparisons of TSC algorithms can also be conducted based on their performance across UCR datasets. Each method in TSC is represented as a vector of 128 values, capturing the classification accuracy of a given algorithm on a particular dataset. Figure 3.4 presents the Spearman correlation among TSC algorithms across all datasets. Notably, strong correlations exist between certain algorithms. Specifically, when two algorithms exhibit similar rankings, they tend to have a high Spearman correlation, suggesting that they might utilize a common underlying technique for classification. From the figure, it is evident that `rocket` and `hivecotev2` exhibit the highest correlation, which aligns with their relatively close rankings (see Figure 3.3). Interestingly, despite `catch22` and `tsforest` having comparable rankings, their correlation is notably weaker. This highlights that algorithms with similar average performance can still differ in terms of which datasets they excel or struggle with—an important aspect to consider when developing algorithm selection meta-models.

Lastly, Figure 3.5 demonstrates the use of Spearman correlation to examine relationships between datasets. Each dataset is represented by a vector of classification accuracies obtained from the previously mentioned TSC algorithms. While the limited number of algorithms constrains the robustness of this analysis, certain trends remain apparent. In the performance space, clusters of datasets emerge, suggesting a strong degree of similarity among some datasets, as previously explored in (Eftimov et al., 2022). Based on algorithm performance, datasets such as `MiddlePhalanxOutlineAgeGroup`, `Herring`, `PigCVP`, and others form a distinct group, while additional datasets cluster in a similar manner. Notably, some datasets, such as `Earthquakes`, exhibit no correlation with any other dataset. This highlights that no single algorithm can be universally considered the best across all

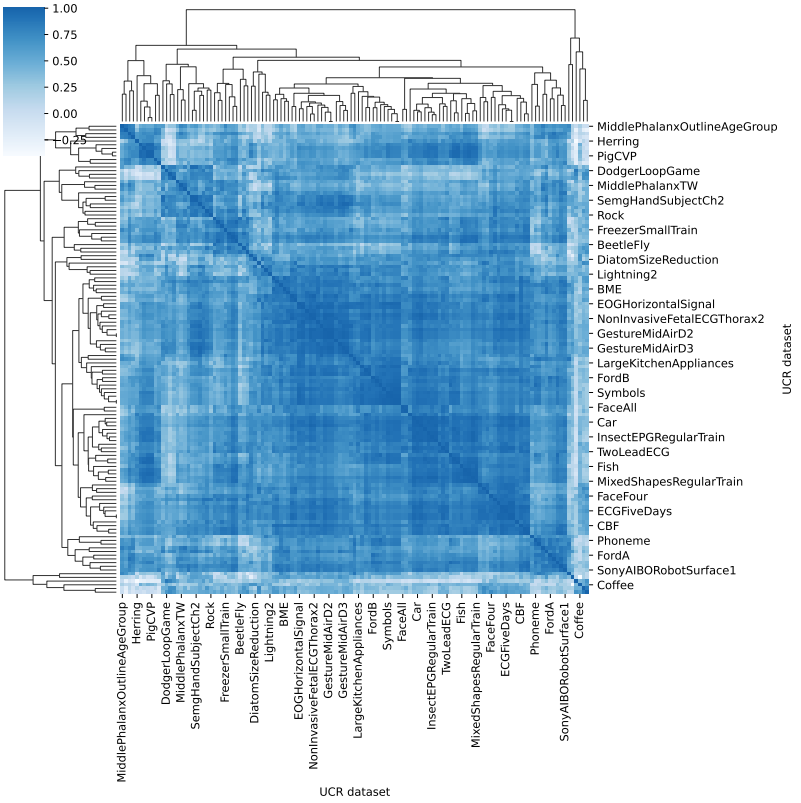


Figure 3.5: Spearman correlation among UCR datasets based on classification accuracies from various TSC algorithms.

datasets, as algorithm rankings fluctuate significantly depending on the dataset in question.

3.2.4.3 Robustness Comparison

Here, we explore how robust various TSC algorithms are when subjected to different transformations. Initially, each TSC algorithm is trained on an undistorted dataset from the UCR archive. After training, the algorithm is evaluated on test data exposed to progressively severe distortions. The aim is to identify the threshold of transformation intensity at which the algorithm's performance starts to decline significantly. Given the impracticality of presenting detailed performance results for each algorithm-dataset pair across all possible distortions, we have concentrated our analysis on the most relevant and intriguing combinations, reporting only aggregated performance metrics.

3.2.4.3.1 Jitter Let us initially consider the simplest and arguably most prevalent distortion, where random noise is introduced into the data. Figure 3.6 demonstrates the classification accuracy of various classifiers as the amount of white noise added to the test data increases. Each line represents an individual UCR dataset, and each subplot corresponds to a different TSC algorithm. Several observations become apparent. Although TSC algorithms differ in terms of performance, they also exhibit varying levels of robustness against noise. To illustrate, consider the dataset `TwoLeadECG`. As anticipated, increasing jitter levels lead to decreased accuracy across all models; however, the extent of performance degradation differs among the TSC algorithms. For the majority of these algorithms, a significant accuracy drop happens between jitter levels σ of 0.01 and 1.0. Even by examining only a select few datasets, notable differences in robustness to jitter among

chosen algorithms emerge. For example, the `inception` model shows strong performance at lower jitter levels but rapidly declines when noise increases, eventually performing worse than simpler methods such as `eknn`. Moreover, the speed at which accuracy decreases under increasing jitter varies between the algorithms. Specifically, on the `TwoLeadECG` dataset, performance degradation with the `weasel` algorithm is relatively swift, whereas the `cnn` algorithm experiences a more gradual decline.

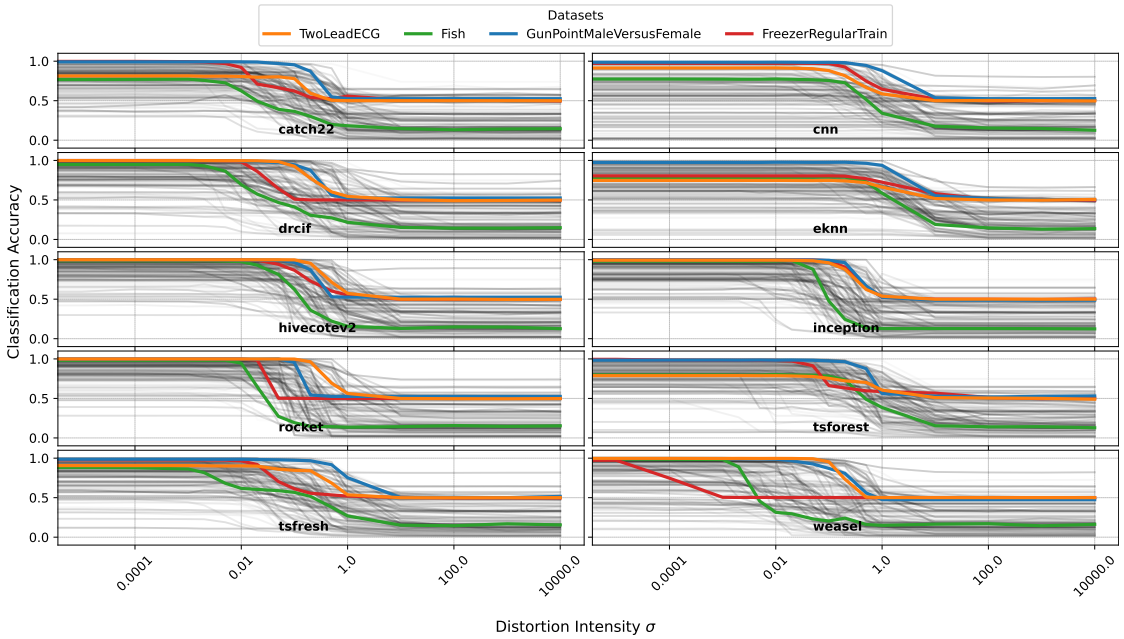


Figure 3.6: Comparative visualization of classification accuracies for various TSC algorithms under increasing jitter levels. Each subplot represents a specific TSC method, and each line within illustrates classification accuracy for a single dataset.

Figure 3.6 provides comprehensive information about the classification accuracy of each TSC algorithm on various datasets at different levels of jitter. However, a more effective approach for comparing algorithms involves evaluating their relative ranks, similar to the method employed in Section 3.2.4.2, but specifically focusing on varying jitter intensities. Figure 3.7 (top) depicts this comparative analysis, illustrating how TSC algorithms perform across different jitter levels. One particularly surprising result is the varying sensitivity of the methods to jitter. Specifically, `hivecotev2`, despite being the highest-ranked algorithm, experiences a relatively sharp decrease in performance as jitter increases. This outcome is unexpected, especially considering that `hivecotev2` is a meta-model composed of multiple algorithms, typically presumed to offer robustness against jitter. On the other hand, `tsforest` and `eknn` exhibit an interesting performance pattern: they rank poorly under conditions of little or no jitter but outperform most other algorithms when exposed to higher levels of jitter.

Another perspective on the observed patterns involves examining the Distortion Error (*DE*), as defined in Section 3.2.3.6 and illustrated in Figure 3.7 (bottom). This measure reflects the relative decline in accuracy for each algorithm across all datasets in the UCR archive. In this visualization, a *DE* value of 1.0 indicates that a particular algorithm consistently outperforms all others across datasets, whereas a value of 0.0 signifies performance comparable to that of the majority class baseline. When averaged over all datasets, noticeable variations emerge regarding jitter sensitivity. For instance, the `weasel` algorithm demonstrates relatively low robustness, experiencing a rapid deterioration in accuracy once

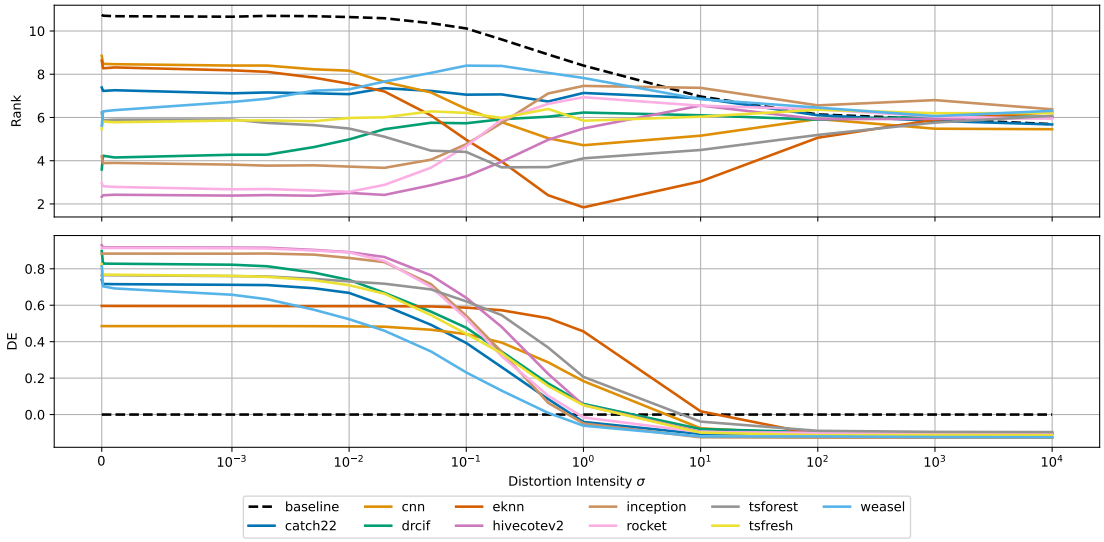


Figure 3.7: Ranks of TSC algorithms (top) and their *DE* scores (bottom) with varying levels of jitter distortion applied.

the jitter magnitude exceeds 0.01. Interestingly, while `eknn` does not lead in classification accuracy, it emerges as the most robust method, tolerating relatively high jitter levels (up to $\sigma = 1.0$) before showing significant performance degradation.

3.2.4.3.2 Random Walk Next, we examine how robust TSC algorithms are to distortions introduced by adding random walks to the time series. Figure 3.8 illustrates the ranks and *DE* values for various TSC methods as the intensity of random walk distortions increases. Although certain differences in robustness among TSC algorithms can be observed, these differences are generally less significant compared to the scenario involving added noise. Here, the ranks of most methods remain relatively consistent, converging toward the baseline as random walk levels range between 1.0 and 10.0. Notably, two algorithms deviate from this trend. The `eknn` classifier performs slightly better than others in terms of ranking at higher random walk levels, whereas the `inception` algorithm shows a noticeable decline in performance already at distortion levels around 0.1. Therefore, we conclude that variations in drift robustness among different algorithms are relatively minor.

3.2.4.3.3 Shift/Spike Here, we examine two distortions: introducing spikes into the time series and applying shifting distortions. Initially, we evaluate the robustness of TSC algorithms to the addition of spikes. Figure 3.9 illustrates the *DE* values and ranks of individual TSC algorithms when spikes are introduced into the time series. We immediately observe significant sensitivity across all algorithms to spike distortions. Typically, even the addition of a single spike causes a performance drop of up to 20%. Consistent with earlier observations, the `tsforest` and `eknn` algorithms again demonstrate the highest robustness against spike-induced distortions. The *DE* plot indicates that the least robust method, `weasel`, approaches baseline-level performance after approximately 20 to 50 spikes, whereas `eknn` continues to outperform the baseline even when subjected to more than 100 spikes.

Next, we evaluate the impact of shift distortion, as illustrated in Figure 3.10. Here, the `drcif` and `hivecotev2` algorithms exhibit the highest robustness, clearly outperforming other methods across all distortion levels. Conversely, `cnn` and `inception` rank among the

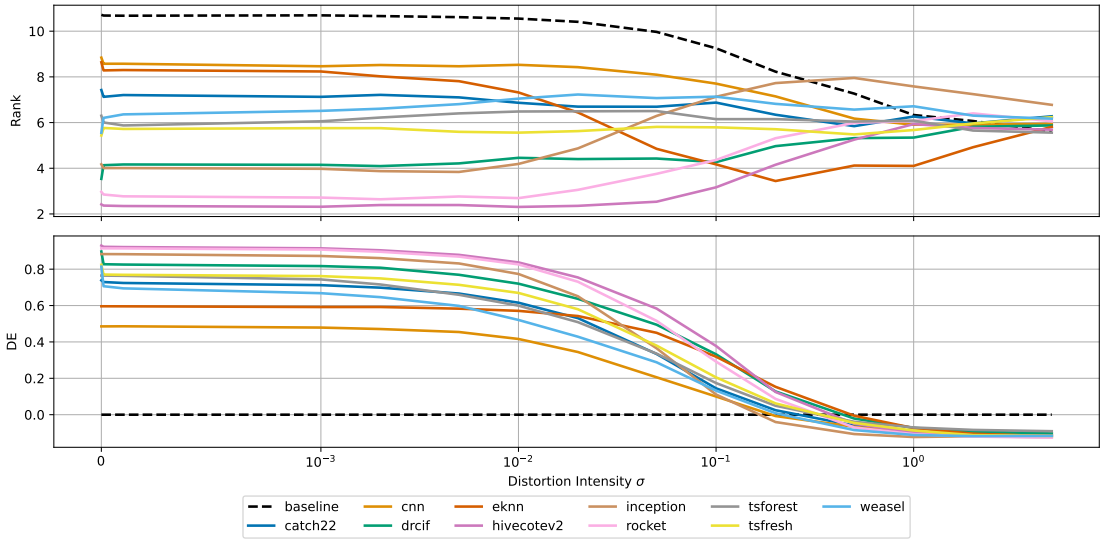


Figure 3.8: Ranks of TSC algorithms (top) and their *DE* scores (bottom) with varying levels of random walk distortion applied.

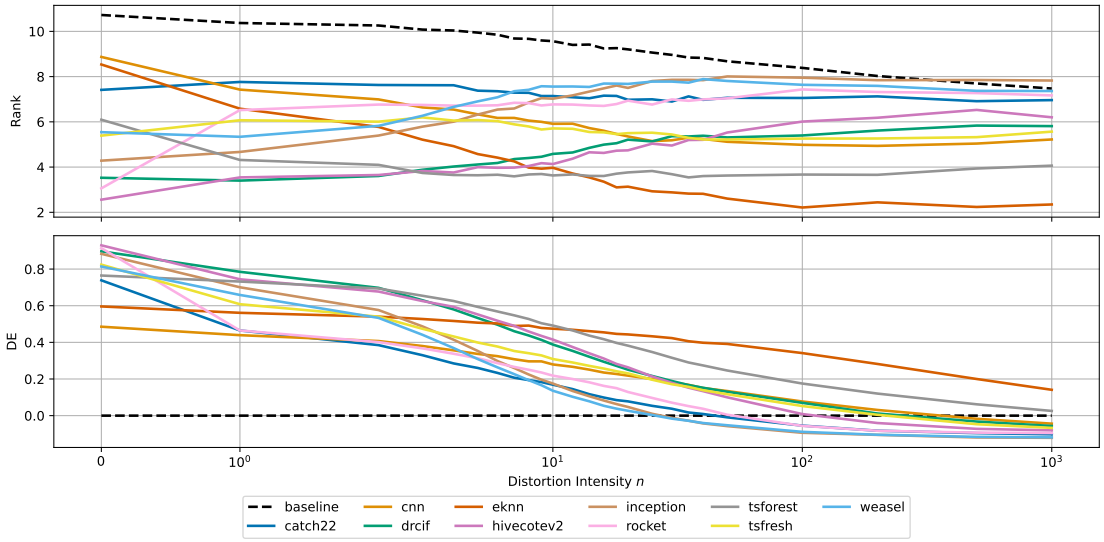


Figure 3.9: Ranks of TSC algorithms (top) and their *DE* scores (bottom) with varying levels of spike distortion applied.

least robust algorithms, rapidly deteriorating in performance as measured by ranks and *DE* scores. At a distortion level of ten shifts, both `cnn` and `inception` fail to surpass the baseline performance, whereas `drcif` maintains approximately half of its original accuracy. Additionally, the response of the `eknn` classifier is noteworthy; while it demonstrates strong robustness against spike distortions, it is significantly less resilient to shifts. This discrepancy arises from the nature of the distortions: spikes affect only local values, whereas shifts alter the overall global structure of the time series. Thus, although spikes and shifts may initially appear conceptually similar, algorithms can respond very differently to these two types of distortions.

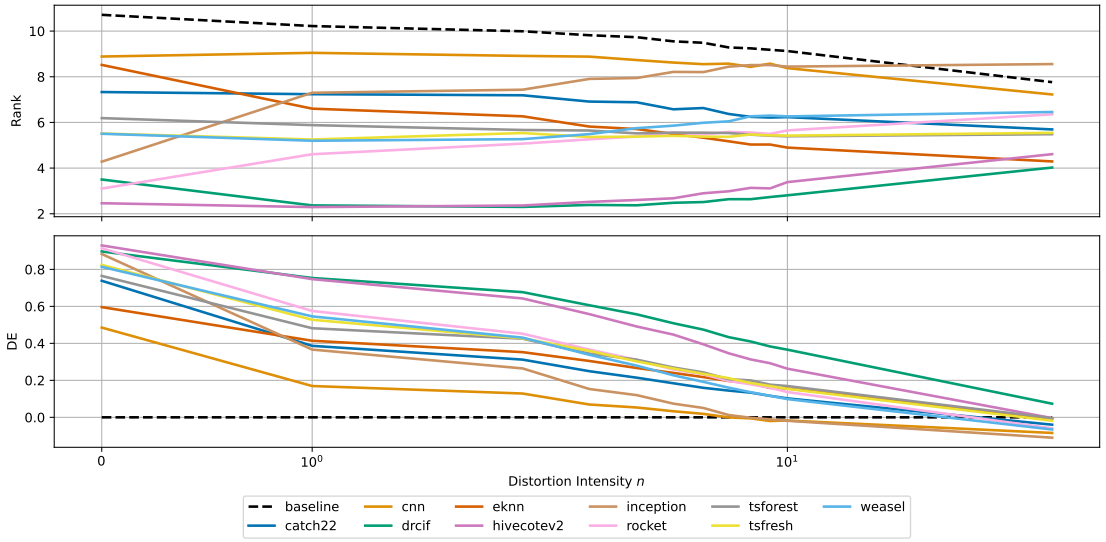


Figure 3.10: Ranks of TSC algorithms (top) and their *DE* scores (bottom) with varying levels of shift distortion applied.

3.2.4.3.4 Quantization/Data Loss Quantization and data loss represent two forms of distortion commonly encountered in signal transmission. Here, we first address quantization, a process where values are discretized into several distinct levels. In the extreme scenario, values are represented using only two levels (1-bit quantization), whereas in the optimal case, they can be represented by 64 levels (6-bit quantization). Starting with the scenario involving 64 quantization levels, it is clear from Figure 3.11 that higher quantization levels minimally disrupt the structure of the time series. At 6-bit quantization, the ranks of TSC algorithms closely match their performance without quantization. However, the effects of quantization become increasingly evident as fewer levels are utilized. At 4-bit quantization, a noticeable decline in performance emerges for all algorithms. With two and three-bit quantization, most algorithms still perform marginally better than the `baseline`. Among these, `eknn` demonstrates the greatest robustness, followed closely by `tsforest` and `hivecotev2`. Conversely, `weasel` proves to be the least robust, experiencing the steepest decline in accuracy under increased quantization. In summary, quantization levels below four bits significantly deteriorate algorithm performance, though performance losses become progressively less pronounced as the quantization precision increases.

Regarding data loss distortion, random values are removed from a time series and subsequently imputed using the preprocessing approach described in Section 3.2.3.4. Naturally, data loss and subsequent imputation introduce errors, thereby complicating classification tasks. Figure 3.12 displays the ranks and *DE* values for various TSC algorithms under different levels of data loss. The results show that up to approximately 50% data loss, the impact on classification accuracy is relatively modest and similar across all algorithms. Notable exceptions, indicated by their *DE* scores, include `inception`, `catch22`, and `weasel`, which experience slightly faster performance declines. However, as data loss surpasses 50%, the performance of all classifiers deteriorates quickly. Notably, `hivecotev2` demonstrates the highest robustness against data loss up to around 90%, whereas `tsforest` is the most resilient algorithm under even more severe data loss conditions. These findings highlight that moderate data loss generally allows most TSC algorithms to maintain adequate performance, but once data loss exceeds approximately 70%, performance degradation becomes pronounced, with varying levels of robustness observed among different algorithms.

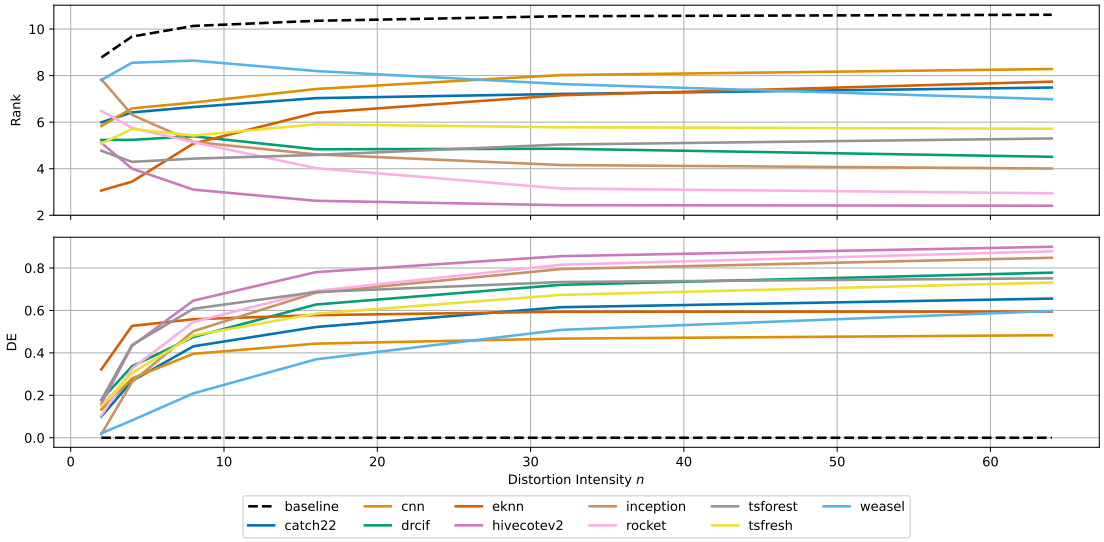


Figure 3.11: Ranks of TSC algorithms (top) and their *DE* scores (bottom) with varying levels of quantization distortion applied.

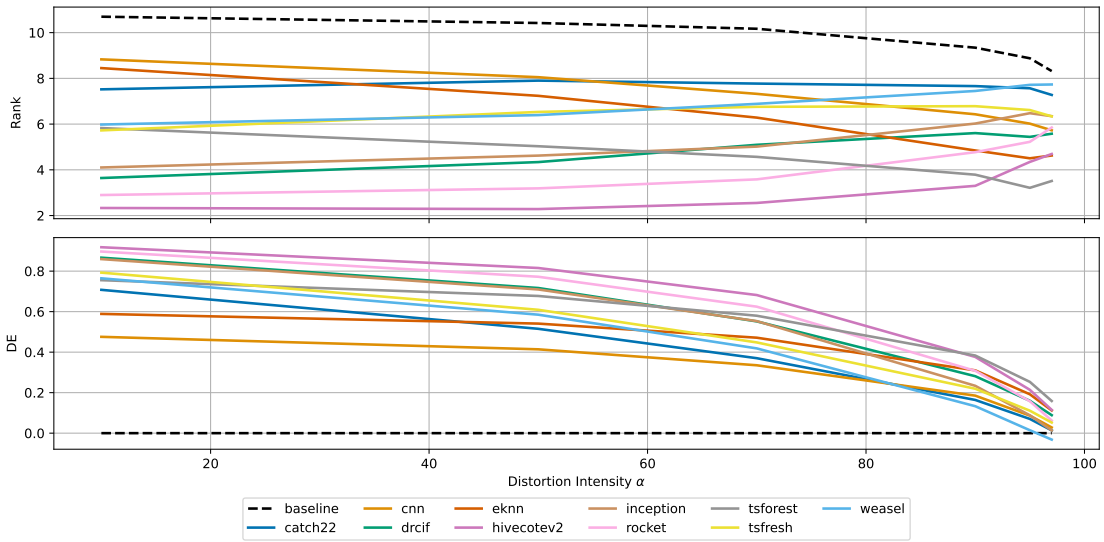


Figure 3.12: Ranks of TSC algorithms (top) and their *DE* scores (bottom) with varying levels of data loss distortion applied.

3.2.4.3.5 Permutation Permutation distortion involves segmenting a time series into equally sized portions and randomly rearranging these segments, disrupting the global structure. Figures 3.13 illustrate the rankings of various TSC algorithms and their corresponding *DE* values when subjected to such permutations. Without permutation, `hivecotev2` emerges as the best-performing algorithm, as anticipated. However, upon permuting the time series, most algorithms exhibit a considerable decline in accuracy, although this decrease varies among methods. For instance, after just one permutation, algorithms like `tsforest`, `eknn`, and `cnn` fall below the `baseline`, highlighting their sensitivity to global structural disruptions. Conversely, other algorithms, such as `inception`, `hivecotev2`, and various feature-based classifiers, display greater robustness, maintaining better accuracy even as the number of permuted segments increases. Although permutation still signif-

icantly impacts performance, these methods consistently outperform the baseline. The varying robustness among algorithms is attributable to their inherent design characteristics. Algorithms such as `eknn` consider the entire time series as indivisible units, making them particularly susceptible to global structural changes. On the other hand, methods like `weasel` segment the time series into smaller intervals before classification, thus partially insulating them from permutation effects. Feature-based approaches similarly maintain robustness since permutations typically affect only specific subsets of features, leaving others intact for accurate classification.

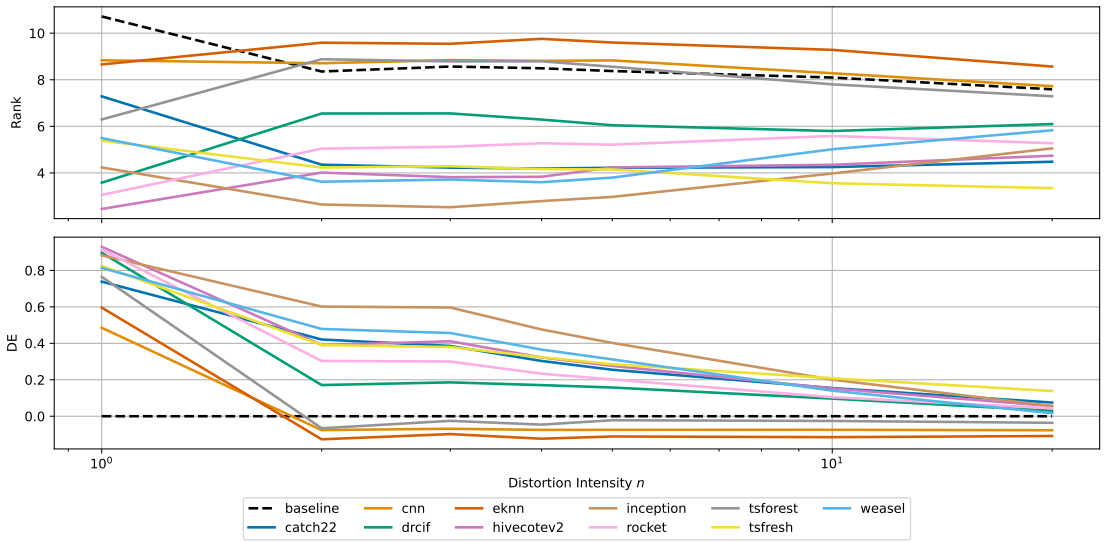


Figure 3.13: Ranks of TSC algorithms (top) and their *DE* scores (bottom) with varying levels of permutation distortion applied.

3.2.4.3.6 Magnitude/Time Warp Here, we investigate how robustness of TCS algorithms is affected by time and magnitude warping. Figure 3.14 shows algorithm rankings and corresponding *DE* scores when magnitude distortions are introduced to a time series. Clearly, even though algorithms differ slightly in robustness to magnitude warping, all methods demonstrate a similar reduction in accuracy when magnitude distortions occur. In particular, with magnitude distortions around $\sigma = 1.0$, each algorithm's performance approximately halves compared to the `baseline`. The only notable deviation from this trend is the `weasel` algorithm, which improves in rank from approximately 6 without distortion to around 4 under magnitude warping. For larger magnitudes of distortion, algorithm performances generally converge toward the baseline. Furthermore, the relative rankings of algorithms remain largely consistent across various distortion magnitudes, indicating that no single method significantly outperforms or underperforms others in terms of robustness.

Figure 3.15 presents the ranks of algorithms and their associated *DE* scores at various levels of time warping. The examined TSC algorithms display relatively limited differences in sensitivity to these distortions. All methods uniformly show a significant and rapid drop in performance as the magnitude of time warping increases. The algorithm rankings remain largely unchanged across all distortion intensities. Notably, the algorithms most sensitive to distortion, namely `cnn` and `eknn`, reach baseline performance earlier as distortions become more pronounced. Overall, although all algorithms are negatively impacted by both magnitude and time-warping distortions, `hivecotev2` maintains superior perfor-

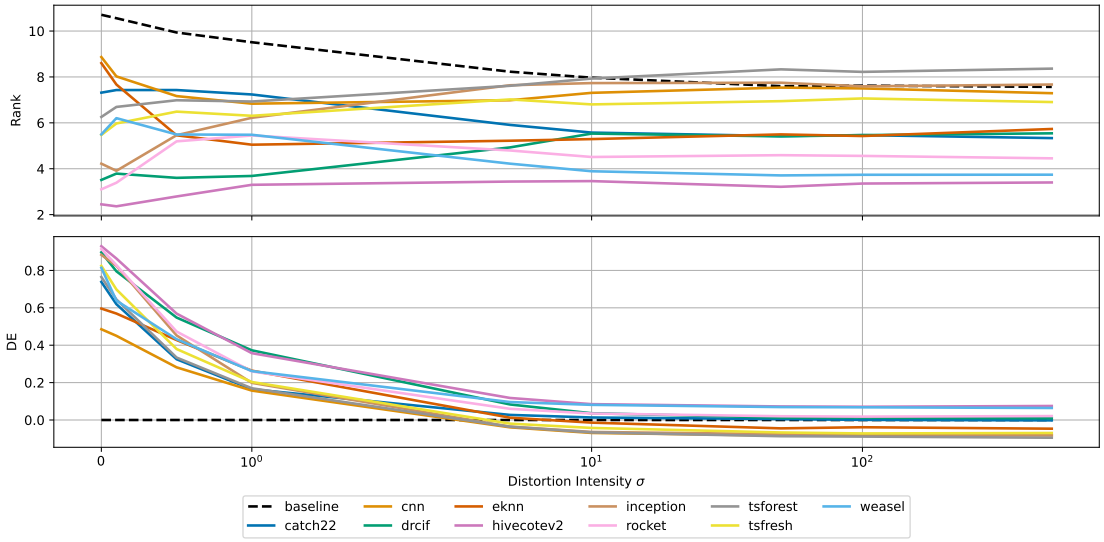


Figure 3.14: Ranks of TSC algorithms (top) and their *DE* scores (bottom) with varying levels of magnitude warp distortion applied.

mance across all tested distortion levels, despite experiencing accuracy reductions when distortions are introduced.

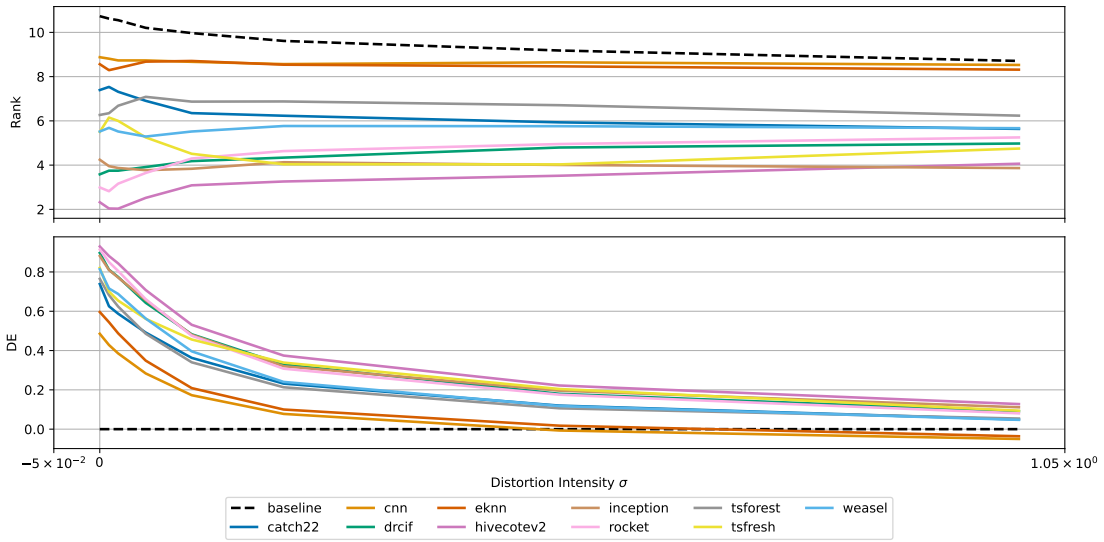


Figure 3.15: Ranks of TSC algorithms (top) and their *DE* scores (bottom) with varying levels of time warp distortion applied.

3.2.4.3.7 Smoothing Moving average smoothing consists of convolving a time series with a filter of specific size k. Figure 3.16 demonstrates how different TSC algorithms respond in terms of robustness to smoothing distortions. Unsurprisingly, all algorithms exhibit reduced performance due to smoothing effects on time series data. Although algorithm rankings shift, these changes are less pronounced compared to other distortion types. The feature-based method `catch22` is particularly vulnerable to smoothing, experiencing a reduction in classification accuracy of over 50% with a smoothing filter size of 10.

Conversely, methods such as `eknn`, `tsforest`, and `hivecotev2` display stronger robustness, maintaining significantly better performance than the `baseline`, even with relatively large smoothing filters (filter size $k > 30$). This finding highlights that certain algorithms can still effectively classify data despite the loss of high-frequency information resulting from smoothing.

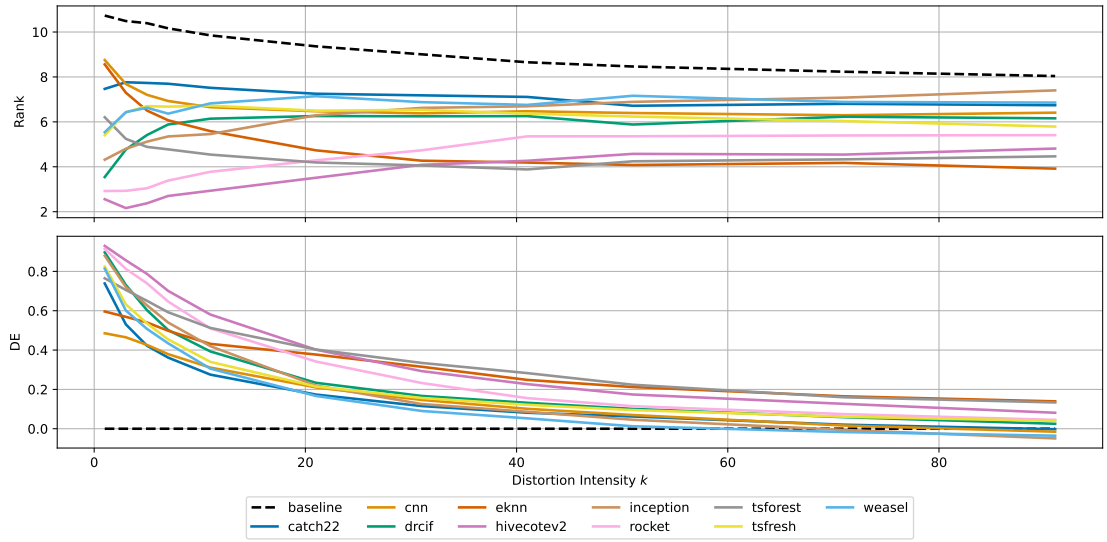


Figure 3.16: Ranks of TSC algorithms (top) and their *DE* scores (bottom) with varying levels of smoothing distortion applied.

3.2.4.3.8 Window Slice Window slicing is a simple distortion technique involving the removal of values from the edges of a time series. This distortion frequently occurs during preprocessing steps, where long time series are segmented into multiple shorter series, and cuts may be selected arbitrarily. Figure 3.17 illustrates algorithm ranks and their respective *DE* scores as an increasing proportion of edge values are discarded. Here, `cnn` and `eknn` demonstrate the lowest robustness, with their classification accuracy dropping close to baseline performance after approximately 30% of edge values are removed. In contrast, `hivecotev2` and `inception` exhibit the greatest robustness to this distortion, consistently outperforming the baseline method at all tested distortion levels. Moreover, these algorithms maintain superior *DE* scores compared to the baseline, even when up to 60% of edge information is removed. This indicates that certain methods retain the capability to accurately identify patterns and classify series despite the loss of boundary information due to improper segmentation.

3.2.4.3.9 Scaling/Constant Here, we examine transformations of time series data through scaling and addition of constants. It is essential to highlight that these transformations—adjusting the scale or shifting the series—are somewhat different from other types of distortions. Usually, issues arising from incorrect scaling or shifting of time series are resolved during preprocessing before classification. Additionally, certain TSC algorithms inherently address such adjustments by dividing a series into segments, individually scaling these segments, and then using them for further analysis. In these cases, preliminary scaling or shifting becomes unnecessary since the classifier internally manages these processes. The test set time series undergo initial scaling as described in Section 3.2.3.4 and are subsequently deliberately rescaled or shifted by a particular scalar. While this

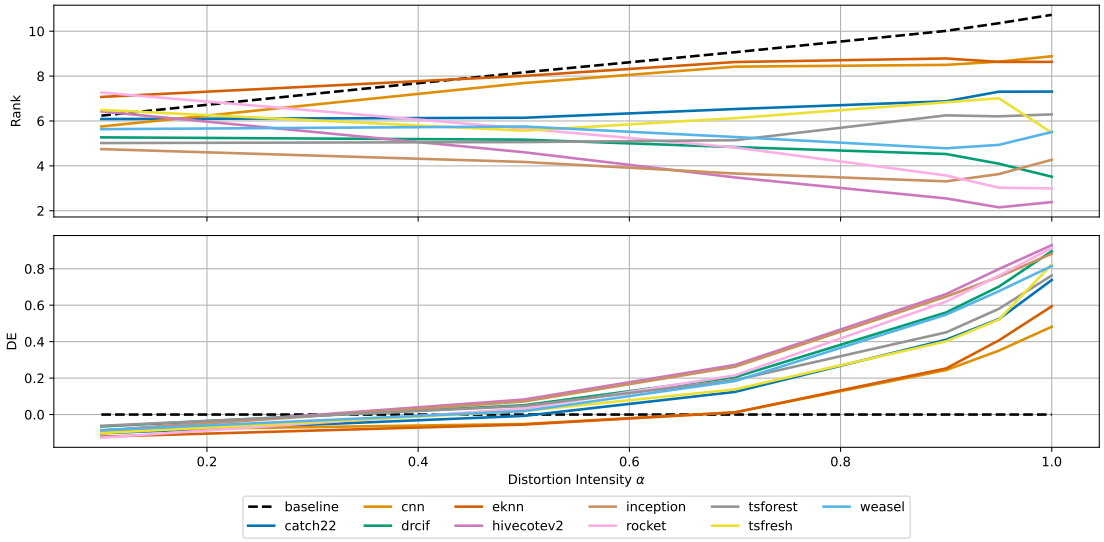


Figure 3.17: Ranks of TSC algorithms (top) and their *DE* scores (bottom) with varying levels of window slice distortion applied.

scenario might not be typical in real-world applications, it provides valuable insights into how widely used TSC algorithms perform when confronted with improperly scaled time series. Despite the common practice of scaling time series prior to classification, we assess classifier performance by deliberately introducing incorrectly scaled series.

Initially, we explore the impact of shifting individual time series by adding constants. These constants are randomly sampled from a normal distribution with zero mean and a specified standard deviation, σ. Figure 3.18 presents the algorithm ranks and associated *DE* scores as a function of the magnitude of the constants added. The findings indicate that not all algorithms are robust against constant addition; indeed, most experience a reduction in accuracy. Only `eknn`, `hivecotev2`, and `rocket` maintain stable performance despite this distortion. Among algorithms negatively affected, `weasel`, `inception`, and `cnn` show the least robustness. These results emphasize the significance of proper normalization of time series data, as introducing constants of larger magnitudes substantially impairs the performance of many algorithms. Although preprocessing generally addresses such issues, assessing the sensitivity of various implementations to improperly normalized data remains beneficial, given that many algorithms still exhibit decreased accuracy under this form of distortion.

The next distortion we investigate involves scaling time series data. Figure 3.19 illustrates the ranks and corresponding *DE* scores for algorithms under various scaling conditions. It is evident that most TSC algorithms evaluated in our study are sensitive to inconsistent scaling. Specifically, algorithms like `catch22`, `drcif`, `tsfresh`, `cnn`, `tsforest`, and `inception` demonstrate significant reductions in accuracy when scaled with positive scalars different from those used during training. Conversely, the implementations of `hivecotev2`, `rocket`, `weasel`, and `eknn` exhibit robustness against scaling changes. An intriguing scenario occurs when using negative scalars, effectively flipping the time series. With a scalar of zero, individual time series lose all information, leading to poor classification performance across all algorithms. However, the most surprising observation arises when the scalar is set to negative one, causing the series to flip. In this scenario, `drcif` and `catch22` perform best, closely followed by `tsfresh`, while other algorithms perform similarly to or slightly better than the baseline. This finding is notable since the top-performing methods utilize

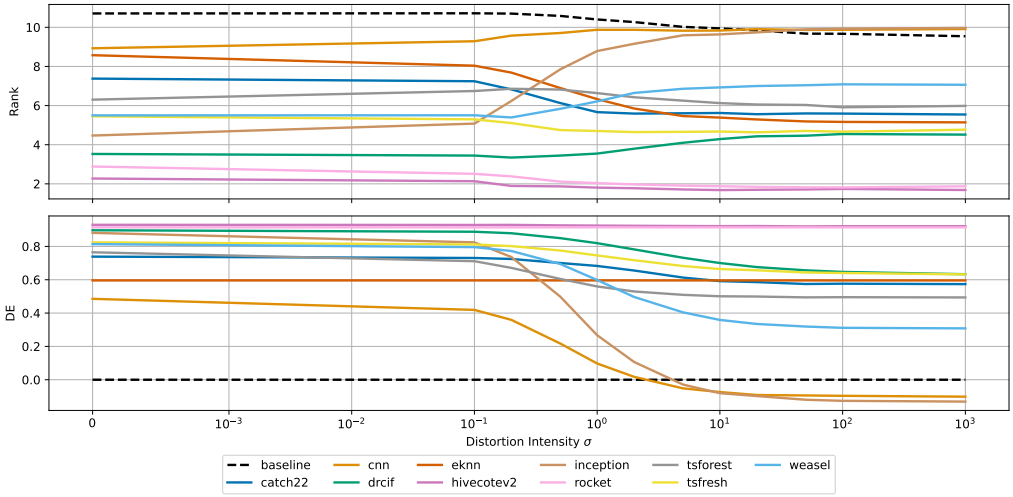


Figure 3.18: Ranks of TSC algorithms (top) and their *DE* scores (bottom) with varying levels of constant distortion applied.

handcrafted features either directly or internally. Although we do not explicitly examine the reason behind the robustness of feature-based methods to flipped series, one plausible explanation is that certain features within `catch22` and `tsfresh` might be invariant to flipping. This could clarify why feature-based methods surpass the baseline despite the observed accuracy decrease, given that not all features would inherently remain invariant to this type of distortion.

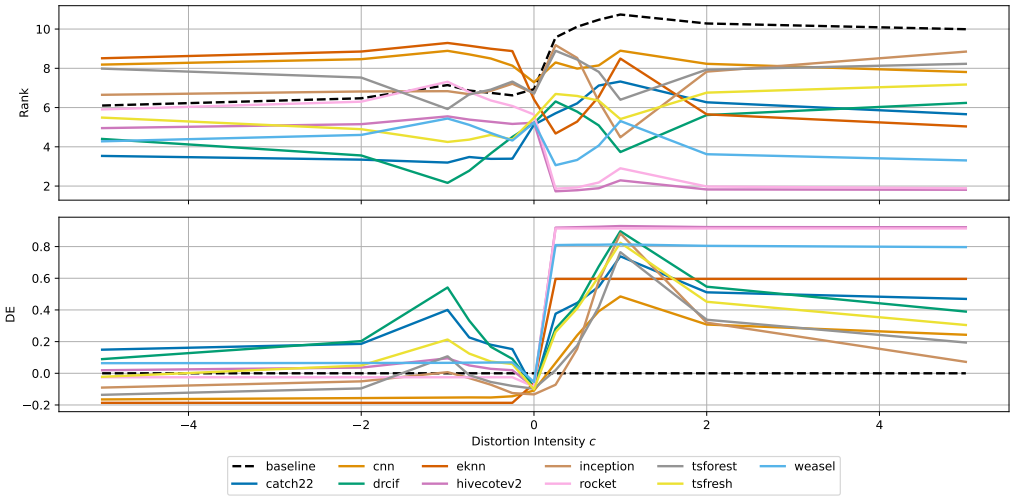


Figure 3.19: Ranks of TSC algorithms (top) and their *DE* scores (bottom) with varying levels of scale distortion applied.

3.2.5 Perspective

Fully understanding why a particular algorithm exhibits greater robustness than another is beyond the scope of this research, as it would necessitate a detailed analysis of each algorithm's individual components—an inherently complex task, especially with deep learning methods and ensemble models. Nevertheless, we can still draw general conclusions re-

garding algorithm robustness. Although certain robustness characteristics might already be recognized within the research community, the primary contribution of this study is to quantify precisely how various distortions impact accuracy. Our findings align with previous research on adversarial robustness, which typically did not emphasize naturally occurring distortions common in real-world sensor data. By specifically addressing these real-world distortions, our study expands existing literature, highlighting the fact that even state-of-the-art algorithms can exhibit reduced performance under non-adversarial, naturally occurring conditions.

Feature-based methods, such as `catch22` and `tsfresh`, operate by extracting handcrafted features from time series data. Because these methods comprise numerous diverse features, they generally demonstrate robustness to distortions affecting only a subset of features, leaving most unaffected or minimally altered. Examples of distortions to which these approaches are robust include permuting a series, flipping it (scaling by -1), or adding constants. Their resilience primarily stems from the invariance or minimal sensitivity of certain features to these changes. For example, the `count_above_mean` feature, which counts values above the mean, remains unaffected by adding a constant or permuting the series. If classifiers heavily rely on such robust features, distortions that modify the series without significantly affecting these specific features result in only slight reductions in performance. In practice, some features naturally exhibit greater robustness than others. During training, a classifier may learn to rely on features that are either robust or sensitive to particular distortions. During inference, classifiers use a mixture of distortion-sensitive and distortion-robust features, causing overall accuracy to decrease based on the proportion and importance of these features. Nevertheless, the `catch22` and `tsfresh` feature sets become less robust when faced with more extensive distortions affecting numerous features. A related but somewhat distinct method is `drcif`, which internally leverages handcrafted features for classification. Under several distortion conditions, `drcif` consistently outperforms both `catch22` and `tsfresh`, indicating that extracting features from subsets of the time series for classification purposes can enhance both performance and robustness. This advantage is particularly noticeable when significant distortions occur at only a few points within the series, such as spikes or abrupt shifts. Additionally, an advantage of handcrafted features is that some possess invariance, or partial invariance, to specific transformations (e.g., incorrect scaling or flipping), a property not commonly found in automatically derived features.

We next consider the `weasel` algorithm, which, similar to `catch22` and `tsfresh`, constructs features from time series data. Although `weasel` automatically generates features rather than relying on predefined handcrafted ones, its behavior under distortions closely mirrors that of `catch22` and `tsfresh`. The primary distinction is evident in its response to scaling and flipping distortions. Given the automatic generation of features, it's unlikely for `weasel` to develop features specifically invariant to flipping a series, resulting in decreased robustness under such conditions. Conversely, when a series is scaled by a positive factor, `weasel` maintains performance stability, which contrasts with `catch22` and `tsfresh`, both of which are sensitive to scaling by positive factors.

The distance-based `eknn` algorithm stands out as one of the most robust among all evaluated algorithms, particularly when handling larger distortions. It demonstrates strong robustness against the addition of jitter, random walk distortions, spikes, quantization, missing data, smoothing, and constant value shifts. However, the algorithm struggles with distortions that alter the global structure of a time series, such as shifts, permutations, warping, and flipping. In this case, understanding why certain distortions significantly reduce performance is relatively straightforward: distortions that change the global structure of the time series degrade performance substantially, whereas those that modify only

the local structure, such as noise, have a smaller impact on accuracy. While noise increases the Euclidean distance between the distorted and original time series, the overall shape and order of the series remain largely consistent, preserving the pattern used for comparison. In contrast, global distortions are more disruptive because Euclidean distance is highly sensitive to the alignment of corresponding time points. Even small distortions can cause significant misalignment, leading to a large increase in Euclidean distance and a corresponding decrease in algorithm performance.

The `rocket` algorithm deserves special consideration regarding robustness. Although generally robust, certain scenarios highlight its vulnerability. Specifically, two primary cases emerge: (i) when numerous minor distortions, such as added noise, collectively modify the kernel activation values slightly, their aggregate effect—particularly through the proportion-of-positive-values function—may sufficiently alter the distribution to trigger misclassification; and (ii) when substantial distortions, like the introduction of spikes, significantly shift the distribution for a few activations, affecting classification accuracy. Despite its relative robustness, these vulnerabilities suggest sensitivity under conditions where distortions alter kernel activations either cumulatively in small amounts or significantly in isolated cases.

Comparing neural network-based algorithms, such as `cnn` and `inception`, poses challenges due to significant architectural differences. Furthermore, neural networks often operate as black boxes, complicating interpretations of classification decisions. The `cnn` method is among the earliest convolutional algorithms applied to time series data but lacks many contemporary enhancements implemented in `inception`. Given the relatively weaker performance of `cnn`, we primarily examine `inception`. Generally, `inception` demonstrates strong performance compared to other approaches but is notably sensitive to severe distortions. The reasons for this sensitivity likely relate to input preprocessing practices for neural networks. Correct scaling and normalization are essential for optimal performance, and distortions significantly deviating from the training data distribution—such as spikes or substantial jitter—can introduce previously unseen values, causing error propagation through network layers and resulting in reduced accuracy. Moreover, both `cnn` and `inception` struggle when distortions include added constants, scaling, or flipping, as these transformations substantially alter the underlying structure of the time series beyond what convolutional filters alone can effectively address. These results underscore the importance of data augmentation during neural network training to ensure the network learns more generalizable features by encountering diverse inputs. Surprisingly, `inception` displays robustness against permutation distortions, an unexpected outcome given its convolutional structure.

Finally, we examine `hivecotev2` alongside two related algorithms, `drcif` and `tsforest`, as their robustness patterns exhibit similarities. Notably, `drcif` constitutes one component of `hivecotev2`, whereas `tsforest` was initially developed as an independent classifier before becoming part of `hivecotev2`'s predecessor. Typically, `drcif` serves primarily to evaluate individual component behaviors within `hivecotev2`, and it is not commonly employed independently. Given the complexity and multi-component nature of `hivecotev2`, pinpointing the exact reasons behind its robustness to certain distortions is challenging. Hence, analyzing and understanding the robustness of its individual components becomes a practical alternative. Although `hivecotev2` typically achieves the highest performance under undistorted conditions, it is not consistently the most robust under all distortions. All three methods share comparable robustness profiles, yet `drcif` and `tsforest` generally lag behind `hivecotev2`. An exception occurs when evaluating flipped time series, where both `drcif` and `tsforest` outperform `hivecotev2`. This observation suggests that ensemble methods generally provide greater robustness than individual algorithms, although

specific distortions, such as time series flipping, can yield greater robustness from single-component methods due to inherent invariances within those components.

3.2.6 Conclusion

This study evaluated the robustness of time series classification algorithms under various common distortions, providing insights into their performance in real-world scenarios. Understanding these aspects is crucial for several reasons. Primarily, it aids in gaining deeper insights into existing time series classification algorithms and developing improved ones. Additionally, it assists practitioners in real-world situations by simplifying the decision-making process regarding selecting the most suitable algorithm and augmentation techniques when distortions are anticipated.

To quantify performance degradation across different algorithms, the study introduces the Distortion Error metric, which captures the effect of distortions on classification accuracy. This metric normalizes results across multiple datasets, facilitating equitable comparisons and offering insights into the anticipated performance decline in the presence of distortions. Additionally, we assess robustness by ranking the algorithms, ensuring that our comparisons are both dependable and reflective of modern as well as traditional evaluation benchmarks.

Our findings reveal notable variability in the robustness of time series classification algorithms. Performance shifted significantly with increasing levels of distortion, highlighting the inherent limitations of certain TSC methods. While state-of-the-art approaches excelled on clean data, they did not always retain their advantage under distorted conditions. In contrast, simpler algorithms frequently exhibited stronger robustness, at times outperforming those traditionally viewed as the best. The study also uncovered distinct patterns in how different algorithms react to various distortions.

Overall, our research offers valuable insights into the resilience of time series classification algorithms, equipping both practitioners and researchers with a deeper understanding of their performance in practical, real-world scenarios. By examining the strengths and limitations of various methods when confronted with common distortions, our findings lay the groundwork for developing more adaptable and robust classification frameworks tailored to a range of time series applications. This enhanced understanding ultimately enables practitioners to make more informed, context-specific decisions when selecting algorithms.

3.2.7 Discussion

This study investigates the robustness of existing time series classification algorithms and their responses to distortions of varying magnitudes. It provides detailed robustness profiles for multiple TSC algorithms, evaluated across a wide range of distortion types. The findings reveal significant differences in robustness between algorithms, with older, simpler methods often outperforming state-of-the-art approaches under certain distortion scenarios. Furthermore, the robustness of these algorithms is shown to depend heavily on the underlying principles driving their design. Understanding these behaviors is crucial, as it enables practitioners to select the most appropriate algorithm for specific applications, ensuring better performance and reliability in real-world scenarios. The main scientific contribution is the additional insight that can guide researchers in refining existing approaches and developing new methods that are inherently more robust to diverse distortions, thereby advancing the field of time series classification.

With this, we confirm the hypothesis **H4**, *By simulating the most frequent types of distortions, it is possible to empirically evaluate the robustness of different time series classification algorithms in terms of their classification accuracy in the presence of various*

data distortions. We demonstrate that it is indeed true that existing algorithms exhibit varying robustness patterns when encountering distortions. For some distortions, simpler models that typically achieve lower classification accuracy may be considered more robust than more complex state-of-the-art models, which can suffer a rapid decline in accuracy. Additionally, we observe substantial differences in sensitivity to global distortions (e.g., permutations) versus local distortions (e.g., added noise), where a model might be extremely sensitive to one distortion but not to others.

Chapter 4

Conclusions

The main focus of this thesis is the exploration of feature construction techniques in time series analysis and single-objective continuous black-box optimization. For single-objective optimization, we propose two novel representations for constructing features tailored to such problems. In the domain of time series analysis, our emphasis lies on the explainability of feature construction approaches for algorithm selection and enhancing the robustness of classification methods when faced with distortions.

The first feature construction technique is based on topological data analysis, a set of techniques designed to extract features that capture the topological structures of point clouds, such as the presence of holes across different scales. These structures are primarily identified through the distances between samples, making the features invariant to rotations and translations. As a result, tools from topology are particularly well-suited for feature construction in the single-objective optimization domain. We adapt this technique to single-objective optimization functions and demonstrate that the proposed features effectively differentiate between COCO problem instances. Furthermore, we show that these features can predict high-level properties of previously unknown functions. Lastly, we explore the use of topological features for algorithm selection, revealing that the proposed features significantly outperform baselines when new problems share some similarities with the problems used to train the meta-model. However, performance decreases when new problems are substantially different.

The proposed topological approach is followed by a method where features are constructed using random functions. Random functions have proven successful in other domains for extracting meaningful features, with their main advantage being the flexibility for users to define how feature extraction functions are created. These functions are then applied to generate feature representations. Similar to the topological approach, we demonstrate that this method can extract features capable of differentiating between COCO problem instances. Additionally, we show that the proposed randomness-based filters can effectively detect high-level properties of objective functions. Lastly, we employ the extracted features for algorithm selection, reaching conclusions similar to those with topological features: the proposed random function-based features are effective for algorithm selection when new objective functions do not differ significantly from the training data.

Exploring new feature construction techniques for optimization problems is essential for several reasons. Firstly, new features can uncover information about objective functions that existing approaches fail to capture. Secondly, these new perspectives on objective functions can aid in designing more representative benchmarks. Lastly, the ultimate goal of feature construction is to enhance our understanding and effectiveness in solving optimization problems.

In time series analysis, we approach feature construction with a focus on explainability and robustness. We begin by examining the use of time series meta-features to predict the performance of various forecasting algorithms, emphasizing the importance of transparency in algorithm selection techniques. Our work demonstrates that existing feature construction methods can accurately predict forecasting algorithm performance on new time series, significantly outperforming baseline models. We highlight key features that play a critical role in predicting algorithm performance and show a strong agreement between different algorithm selection models and explainability techniques in identifying the most relevant features.

We explore the robustness of feature representations and the effects of distortions on classification accuracy. We evaluate existing time series classification algorithms to assess their performance when encountering naturally occurring distortions of varying magnitudes. Detailed robustness profiles are provided for multiple classification algorithms, covering a wide range of distortion types. Our findings reveal significant differences in robustness between algorithms, with older, simpler methods often outperforming state-of-the-art approaches under certain distortion scenarios. The robustness of these algorithms is shown to depend heavily on the design principles underlying their development. These insights can assist practitioners in selecting the most appropriate classification algorithms and contribute to the development of new algorithms by highlighting where current methods fail, enabling the creation of more robust solutions.

In the time series domain, we approach feature construction from the perspectives of robustness and explainability. The main scientific contributions are as follows. For time series forecasting, we demonstrate that explainability techniques can make the algorithm selection process more transparent, offering significant benefits by uncovering patterns that explain why certain forecasting algorithms are more suitable than others. Additionally, we investigate the robustness of feature representation techniques and examine how distortions impact downstream tasks. These insights enhance our understanding of feature construction methods, helping to identify when one technique is more appropriate than another.

4.1 Related Hypotheses and Contributions

In this thesis, we explore hypotheses primarily related to constructing representations in the domains of time series analysis and single-objective continuous optimization. The hypotheses addressed in this thesis are as follows:

H1a: *Using insights from topological data analysis, it is possible to construct rotation and translation invariant features that differentiate between single-objective optimization problems and capture their high-level properties.*

H1b: *Performance of predictive models for optimization algorithm selection based on such topological features is comparable to the performance of models based on existing exploratory landscape analysis features.*

H2a: *Features based on randomly initialized filters calculated using distances between optimization problem samples are able to differentiate between distinct single-objective optimization problems and capture their high-level properties.*

H2b: *Performance of predictive models for optimization algorithm selection based on such features is comparable to the performance of models based on existing exploratory landscape analysis features.*

H3: *Meta-models for selecting time series forecasting algorithms are able to be explained using feature scoring techniques.*

H4: *By simulating the most frequent types of distortions, it is possible to empirically evaluate the robustness of different time series classification algorithms in terms of their classification accuracy in the presence of various data distortions.*

The thesis focuses on feature engineering and representation learning in both time series analysis and single-objective continuous optimization domains. We develop novel methodologies for feature construction and evaluate existing ones for various tasks. Each scientific contribution is closely related to the proposed hypotheses. The main scientific contributions of this thesis are as follows:

SC1: We propose a novel feature extraction approach based on topological data analysis for constructing representations of single-objective continuous optimization functions (Petelin et al., 2024). By capturing the intrinsic structural properties of the optimization landscapes, the proposed topological approach provides a compact yet informative characterization of problems. These representations enable us to perform various tasks, including problem differentiation and high-level property prediction. In the context of problem differentiation, also referred to as problem classification, we show that topological features substantially outperform the baseline and are almost on par with existing state-of-the-art approaches. In terms of capturing high-level properties of objective functions, we demonstrate that topological features are able to detect high-level function properties such as multimodality and global/funnel structure better than the proposed baseline, although they are still somewhat less powerful than some of the existing state-of-the-art approaches. As the proposed methods in both cases outperform the baseline, hypothesis **H1a** is confirmed. Building on that, hypothesis **H1b** concerns the use of the proposed topological features for algorithm selection. We show that in scenarios where objective functions do not differ substantially, the proposed features can be used to recommend the most suitable algorithm, but fall behind in terms of accuracy compared to the state-of-the-art approaches. However, when the problems in the training and test sets differ significantly, the topological features are not effective for algorithm selection, as performance is on par with the baseline, thus leading to the rejection of hypothesis **H1b**. Although not captured in the hypotheses, the proposed topological features also exhibit additional desirable properties, such as rotational and translational invariance. The main contribution to science is, therefore, a set of newly proposed features that capture topological properties and can be used for various downstream tasks in optimization.

SC2: The proposed research introduces a novel feature extraction approach based on randomly initialized functions for constructing representations of single-objective continuous optimization functions (Petelin & Cenikj, 2024b). To evaluate the informativeness of the proposed approach, two hypotheses are considered. Hypothesis **H2a** concerns the use of the proposed feature extraction technique for problem differentiation and high-level property prediction. For both tasks, the proposed approach substantially outperforms the baseline and, in some cases, is comparable to state-of-the-art methods. We therefore conclude that hypothesis **H2a** is confirmed. However, we reject hypothesis **H2b**, as our results show that the proposed approach performs worse than state-of-the-art methods when the training and test sets are similar, and achieves only baseline-level performance on completely unseen objective functions. An additional scientific contribution, not fully captured by the proposed hypotheses, is that the extracted features provide complementary information to existing

representations, thereby enhancing performance in these tasks. These features are therefore an important contribution, as they have shown potential for use in various tasks within the domain of optimization.

SC3: We integrate explainability techniques into the forecasting algorithm selection pipeline to enhance transparency in decision-making and to identify the most influential time series properties affecting algorithm performance (Petelin et al., 2023), as stated in hypothesis **H3**. We first train performance-predicting meta-models that map time series features to the performance of forecasting algorithms on the same time series. The trained meta-models substantially outperform the baseline, indicating that they are indeed capable of identifying the most suitable forecasting algorithms based on the time series characteristics. Following that, we demonstrate that incorporating explainability into a forecasting algorithm selection meta-modeling pipeline provides deeper insights into the decision-making process and highlights the most relevant time series properties. Furthermore, the thesis shows that important features are consistently identified, regardless of the explainability method or the algorithm selection model used. The main scientific contribution related to hypothesis **H3** is thus a better understanding of how meta-models make decisions based on time series features.

SC4: As part of hypothesis **H4**, the thesis investigates the impact of naturally occurring distortions on feature extraction and downstream classification performance. It examines how classifier performance degrades in the presence of distortions, demonstrating that models with similar accuracy on undistorted data can exhibit significantly different levels of robustness. For certain types of distortions, state-of-the-art models may underperform compared to simpler models in terms of classification accuracy, confirming the hypothesis that models indeed exhibit varying levels of robustness to distortions. Additionally, the thesis provides explanations for the underlying factors contributing to these differences in classifier robustness. The main contribution to science is the identification of which specific time-series classification algorithms are more sensitive to particular distortions, as this may help practitioners select the most appropriate models to use.

4.2 Future Work

While previous chapters already cover some aspects of future work for the presented approaches, here we summarize and provide a broader overview of future work and open challenges.

For feature construction in continuous black-box optimization, the major challenges are related to the generalization of features and the efficient computation of those features. By feature generalization, we refer to the ability of features to better capture the underlying properties of the optimization function so that they perform well in a leave-problem-out algorithm selection setting. This is not the case with any existing feature set, including the ones proposed in this thesis. As such, the primary challenge remains the construction of features that can generalize effectively. Another significant challenge is the computational efficiency of the proposed approaches, which may require substantial computational resources, especially for high-dimensional problems. While some potential directions have already been discussed in the respective papers, two key avenues for reducing computational cost include improved parallelization and the use of approximate algorithms wherever possible, particularly in the case of topological approaches.

In the second part of the thesis, we primarily explore constructed representations in the domain of time series, specifically how these representations can be used in forecasting algorithm selection and classification tasks. Future work in this area includes the construction of more comprehensive feature sets that are better suited to predict the performance of forecasting algorithms, the training of meta-models on larger and more diverse datasets, and the mitigation of computational costs associated with the computation of features used for meta-learning. In terms of robustness, future work should focus on the explainability of representations, understanding why some are more robust than others, and the development of methods that yield features potentially invariant to specific transformations.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., ... Zheng, X. (2015). `TensorFlow`: Large-scale machine learning on heterogeneous systems [Software available from tensorflow.org]. https://www.tensorflow.org/

Abanda, A., Mori, U., & Lozano, J. A. (2019). A review on distance based time series classification. *Data Mining and Knowledge Discovery*, *33*(2), 378–412.

Abdu-Aguye, M. G., Gomaa, W., Makihara, Y., & Yagi, Y. (2020). Detecting adversarial attacks in time-series data. *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 3092–3096.

Adhikari, R., & Agrawal, R. (2014). A combination of artificial neural network and random walk models for financial time series forecasting. *Neural Computing and Applications*, *24*, 1441–1449.

Ahmed, H. M., Abdulrazak, B., Blanchet, F. G., Aloulou, H., & Mokhtari, M. (2022). Long gaps missing iot sensors time series data imputation: A bayesian gaussian approach. *IEEE Access*, *10*, 116107–116119.

Ahmed, N. K., Atiya, A. F., Gayar, N. E., & El-Shishiny, H. (2010). An empirical comparison of machine learning models for time series forecasting. *Econometric reviews*, *29*(5-6), 594–621.

Al-Ayyad, M., Owida, H. A., De Fazio, R., Al-Naami, B., & Visconti, P. (2023). Electromyography monitoring systems in rehabilitation: A review of clinical applications, wearable devices and signal acquisition methodologies. *Electronics*, *12*(7), 1520.

Albarbar, A., & Teay, S. (2017). Mems accelerometers: Testing and practical approach for smart sensing and machinery diagnostics. *Advanced mechatronics and MEMS devices II*, 19–40.

Alemany, S., & Pissinou, N. (2020). The dilemma between data transformations and adversarial robustness for time series application systems. *arXiv preprint arXiv:2006.10885*.

Angolkar, P. V., Arnold, J. V., Nanda, R. S., & Duncanson Jr, M. G. (1992). Force degradation of closed coil springs: An in vitro evaluation. *American journal of orthodontics and dentofacial orthopedics*, *102*(2), 127–133.

Anik, M. T. H., Ebrahimabadi, M., Danger, J.-L., Guilley, S., & Karimi, N. (2021). Reducing aging impacts in digital sensors via run-time calibration. *Journal of Electronic Testing*, *37*(5), 653–673.

Au Yeung, J. F. K., Wei, Z. K., Chan, K. Y., Lau, H. Y., & Yiu, K.-F. C. (2020). Jump detection in financial time series using machine learning algorithms. *Soft Computing*, *24*, 1789–1801.

Bäck, T. (2005). Evolution strategies: An alternative evolutionary algorithm. *Artificial Evolution: European Conference, AE 95 Brest, France, September 4–6, 1995 Selected Papers*, 1–20.

Bagnall, A., Davis, L., Hills, J., & Lines, J. (2012). Transformation based ensembles for time series classification. *Proceedings of the 2012 SIAM international conference on data mining*, 307–318.

Bagnall, A., & Lines, J. (2014). An experimental evaluation of nearest neighbour time series classification. *arXiv preprint arXiv:1406.4757*.

Bagnall, A., Lines, J., Bostrom, A., Large, J., & Keogh, E. (2017). The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances. *Data mining and knowledge discovery*, *31*, 606–660.

Bai, B., Li, G., Wang, S., Wu, Z., & Yan, W. (2021). Time series classification based on multi-feature dictionary representation and ensemble learning. *Expert Systems with Applications*, *169*, 114162.

Batista, G. E., Keogh, E. J., Tataw, O. M., & De Souza, V. M. (2014). Cid: An efficient complexity-invariant distance for time series. *Data Mining and Knowledge Discovery*, *28*, 634–669.

Belinkov, Y., & Bisk, Y. (2017). Synthetic and natural noise both break neural machine translation. *arXiv preprint arXiv:1711.02173*.

Belkhouja, T., & Doppa, J. R. (2022). Adversarial framework with certified robustness for time-series domain via statistical features. *Journal of Artificial Intelligence Research*, *73*, 1435–1471.

Benavoli, A., Corani, G., & Mangili, F. (2016). Should we really use post-hoc tests based on mean-ranks? *The Journal of Machine Learning Research*, *17*(1), 152–161.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, *35*(8), 1798–1828.

Bhanja, S., & Das, A. (2018). Impact of data normalization on deep neural network for time series forecasting. *arXiv preprint arXiv:1812.05519*.

Cenikj, G., Petelin, G., & Eftimov, T. (2024). A cross-benchmark examination of feature-based algorithm selector generalization in single-objective numerical optimization. *Swarm and Evolutionary Computation*, *87*, 101534.

Chang, K.-W., He, H., Jia, R., & Singh, S. (2021). Robustness and adversarial examples in natural language processing. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: Tutorial Abstracts*, 22–26.

Chiarot, G., & Silvestri, C. (2023). Time series compression survey. *ACM Computing Surveys*, *55*(10), 1–32.

Christ, M., Braun, N., Neuffer, J., & Kempa-Liehr, A. W. (2018). Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, *307*, 72–77.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., & Le, Q. V. (2019). Autoaugment: Learning augmentation strategies from data. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 113–123.

Dau, H. A., Keogh, E., Kamgar, K., Yeh, C.-C. M., Zhu, Y., Gharghabi, S., Ratanamahatana, C. A., Yanping, Hu, B., Begum, N., Bagnall, A., Mueen, A., & Batista, G. (2018, October). The ucr time series classification archive [https://www.cs.ucr.edu/~eamonn/time_series_data_2018/].

DeLang, M., Taheri, S. A., Hutchison, R., & Hawke, N. (2022). Tackling ucr's missing data problem: A multiple imputation approach. *Journal of criminal justice*, *79*, 101882.

Dempster, A., Petitjean, F., & Webb, G. I. (2020). Rocket: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, *34*(5), 1454–1495.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, *7*, 1–30.

Deng, H., Runger, G., Tuv, E., & Vladimir, M. (2013). A time series forest for classification and feature extraction. *Information Sciences*, *239*, 142–153.

de Oliveira, M. A., da Rocha, A. M., Puntel, F. E., & Cavalheiro, G. G. H. (2023). Time series compression for iot: A systematic literature review. *Wireless Communications and Mobile Computing*, *2023*(1), 5025255.

Dietterich, T. G. (2000). Ensemble methods in machine learning. *International workshop on multiple classifier systems*, 1–15.

Ding, D., Zhang, M., Feng, F., Huang, Y., Jiang, E., & Yang, M. (2023). Black-box adversarial attack on time series classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, *37*(6), 7358–7368.

Eftimov, T., Petelin, G., Cenikj, G., Kostovska, A., Ispirova, G., Korošec, P., & Bogatinovski, J. (2022). Less is more: Selecting the right benchmarking set of data for time series classification. *Expert Systems with Applications*, *198*, 116871.

Faouzi, J. (2022). Time series classification: A review of algorithms and implementations. *Machine Learning (Emerging Trends and Applications)*.

Fishburn, F. A., Ludlum, R. S., Vaidya, C. J., & Medvedev, A. V. (2019). Temporal derivative distribution repair (tddr): A motion correction method for fnirs. *Neuroimage*, *184*, 171–179.

Fulcher, B. D., & Jones, N. S. (2014). Highly comparative feature-based time-series classification. *IEEE Transactions on Knowledge and Data Engineering*, *26*(12), 3026–3037.

Garcia, S., & Herrera, F. (2008). An extension on" statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of machine learning research*, *9*(12).

Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., & Brendel, W. (2018). Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv preprint arXiv:1811.12231*.

Gil-Martín, M., López-Iniesta, J., Fernández-Martínez, F., & San-Segundo, R. (2023). Reducing the impact of sensor orientation variability in human activity recognition using a consistent reference system. *Sensors*, *23*(13), 5845.

Golestani, N., & Moghaddam, M. (2020). Human activity recognition using magnetic induction-based motion signals and deep recurrent neural networks. nat. commun. 11 (1), 1551 (2020).

Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Grabocka, J. (2016). *Invariant features for time-series classification* [Doctoral dissertation, Universität Hildesheim].

Gupta, A., Gupta, H. P., Biswas, B., & Dutta, T. (2020). Approaches and applications of early classification of time series: A review. *IEEE Transactions on Artificial Intelligence*, *1*(1), 47–61.

Hansen, N., & Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, *9*(2), 159–195.

Hendrycks, D., & Dietterich, T. (2019). Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*.

Hills, J., Lines, J., Baranauskas, E., Mapp, J., & Bagnall, A. (2014). Classification of time series by shapelet transformation. *Data mining and knowledge discovery*, *28*, 851–881.

Hossain, M. S., Reaz, M. B. I., Chowdhury, M. E., Ali, S. H., Bakar, A. A. A., Kiranyaz, S., Khandakar, A., Alhatou, M., & Habib, R. (2022). Motion artifacts correction from eeg and fnirs signals using novel multiresolution analysis. *IEEE Access*, *10*, 29760–29777.

Iglesias, G., Talavera, E., González-Prieto, Á., Mozo, A., & Gómez-Canaval, S. (2023). Data augmentation techniques in time series domain: A survey and taxonomy. *Neural Computing and Applications*, *35*(14), 10123–10145.

Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., & Muller, P.-A. (2019a). Adversarial attacks on deep neural networks for time series classification. *2019 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., & Muller, P.-A. (2019b). Deep learning for time series classification: A review. *Data mining and knowledge discovery*, *33*(4), 917–963.

Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., & Muller, P.-A. (2019c). Deep neural network ensembles for time series classification. *2019 International Joint Conference on Neural Networks (IJCNN)*, 1–6.

Ismail Fawaz, H., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D. F., Weber, J., Webb, G. I., Idoumghar, L., Muller, P.-A., & Petitjean, F. (2020). Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, *34*(6), 1936–1962.

Iwana, B. K., & Uchida, S. (2021a). An empirical survey of data augmentation for time series classification with neural networks. *Plos one*, *16*(7), e0254841.

Iwana, B. K., & Uchida, S. (2021b). Time series data augmentation for neural networks by time warping with a discriminative teacher. *2020 25th International Conference on Pattern Recognition (ICPR)*, 3558–3565.

Jia, R., Raghunathan, A., Göksel, K., & Liang, P. (2019). Certified robustness to adversarial word substitutions. *arXiv preprint arXiv:1909.00986*.

Jiang, X. (2009). Feature extraction for image recognition and computer vision. *2009 2nd IEEE international conference on computer science and information technology*, 1–15.

Kamann, C., & Rother, C. (2021). Benchmarking the robustness of semantic segmentation models with respect to common corruptions. *International journal of computer vision*, *129*(2), 462–483.

Karim, F., Majumdar, S., & Darabi, H. (2020). Adversarial attacks on time series. *IEEE transactions on pattern analysis and machine intelligence*, *43*(10), 3309–3320.

Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic algorithm: Past, present, and future. *Multimedia Tools and Applications*, *80*, 8091–8126.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of ICNN'95-international conference on neural networks*, *4*, 1942–1948.

Lara, O. D., & Labrador, M. A. (2012). A survey on human activity recognition using wearable sensors. *IEEE communications surveys & tutorials*, *15*(3), 1192–1209.

Large, J., Bagnall, A., Malinowski, S., & Tavenard, R. (2019). On time series classification with dictionary-based classifiers. *Intelligent Data Analysis*, *23*(5), 1073–1089.

Li, G., Choi, B., Xu, J., Bhowmick, S. S., Chun, K.-P., & Wong, G. L.-H. (2020). Efficient shapelet discovery for time series classification. *IEEE transactions on knowledge and data engineering*, *34*(3), 1149–1163.

Linardatos, P., Papastefanopoulos, V., & Kotsiantis, S. (2020). Explainable ai: A review of machine learning interpretability methods. *Entropy*, *23*(1), 18.

Lines, J., Taylor, S., & Bagnall, A. (2018). Time series classification with hive-cote: The hierarchical vote collective of transformation-based ensembles. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *12*(5), 1–35.

Liu, B., & Cheng, H. (2024). De-noising classification method for financial time series based on iceemdan and wavelet threshold, and its application. *EURASIP Journal on Advances in Signal Processing*, *2024*(1), 19.

Liu, Q., Hu, X., Ye, M., Cheng, X., & Li, F. (2015). Gas recognition under sensor drift by using deep learning. *International Journal of Intelligent Systems*, *30*(8), 907–922.

Liu, Z., Lin, Y., & Sun, M. (2023). *Representation learning for natural language processing*. Springer Nature.

Löning, M., Bagnall, A., Ganesh, S., Kazakov, V., Lines, J., & Király, F. J. (2019). Sktime: A unified interface for machine learning with time series. *arXiv preprint arXiv:1909.07872*.

Lubba, C. H., Sethi, S. S., Knaute, P., Schultz, S. R., Fulcher, B. D., & Jones, N. S. (2019). Catch22: Canonical time-series characteristics: Selected through highly comparative time-series analysis. *Data Mining and Knowledge Discovery*, *33*(6), 1821–1852.

Martínez-Arellano, G., Terrazas, G., & Ratchev, S. (2019). Tool wear classification using time series imaging and deep learning. *The International Journal of Advanced Manufacturing Technology*, *104*, 3647–3662.

Meng, Q., Qian, H., Liu, Y., Xu, Y., Shen, Z., & Cui, L. (2023). Unsupervised representation learning for time series: A review. *arXiv preprint arXiv:2308.01578*.

Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., & Rudolph, G. (2011). Exploratory landscape analysis. *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 829–836.

Middlehurst, M., Ismail-Fawaz, A., Guillaume, A., Holder, C., Rubio, D. G., Bulatova, G., Tsaprounis, L., Mentel, L., Walter, M., Schäfer, P., et al. (2024). Aeon: A python toolkit for learning from time series. *arXiv preprint arXiv:2406.14231*.

Middlehurst, M., Large, J., Flynn, M., Lines, J., Bostrom, A., & Bagnall, A. (2021). Hivecote 2.0: A new meta ensemble for time series classification. *Machine Learning*, *110*(11-12), 3211–3243.

Middlehurst, M., Schäfer, P., & Bagnall, A. (2023). Bake off redux: A review and experimental evaluation of recent time series classification algorithms. *arXiv preprint arXiv:2304.13029*.

Middlehurst, M., Schäfer, P., & Bagnall, A. (2024). Bake off redux: A review and experimental evaluation of recent time series classification algorithms. *Data Mining and Knowledge Discovery*, 1–74.

Moore, S. J., Nugent, C. D., Zhang, S., & Cleland, I. (2020). Iot reliability: A review leading to 5 key research directions. *CCF Transactions on Pervasive Computing and Interaction*, *2*, 147–163.

Muñoz, M. A., Sun, Y., Kirley, M., & Halgamuge, S. K. (2015). Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. *Information Sciences*, *317*, 224–245.

Naseer, M. M., Ranasinghe, K., Khan, S. H., Hayat, M., Shahbaz Khan, F., & Yang, M.-H. (2021). Intriguing properties of vision transformers. *Advances in Neural Information Processing Systems*, *34*, 23296–23308.

Nikolikj, A., Trajanov, R., Cenikj, G., Korošec, P., & Eftimov, T. (2022). Identifying minimal set of exploratory landscape analysis features for reliable algorithm performance prediction. *2022 IEEE Congress on Evolutionary Computation (CEC)*, 1–8.

Nomura, M., Akimoto, Y., & Ono, I. (2023). Cma-es with learning rate adaptation: Can cma-es with default population size solve multimodal and noisy problems? *Proceedings of the Genetic and Evolutionary Computation Conference*, 839–847.

Örnhag, M. V., Persson, P., Wadenbäck, M., Åström, K., & Heyden, A. (2022). Trust your imu: Consequences of ignoring the imu drift. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4468–4477.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Perpetuini, D., Cardone, D., Filippini, C., Chiarelli, A. M., & Merla, A. (2021). A motion artifact correction procedure for fnirs signals based on wavelet transform and infrared thermography video tracking. *Sensors*, *21*(15), 5117.

Petelin, G., & Cenikj, G. (2024a). On generalization of ela feature groups. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 419–422.

Petelin, G., & Cenikj, G. (2024b). Random filter mappings as optimization problem feature extractors. *IEEE Access*.

Petelin, G., Cenikj, G., & Eftimov, T. (2023). Towards understanding the importance of time-series features in automated algorithm performance prediction. *Expert Systems with Applications*, *213*, 119023.

Petelin, G., Cenikj, G., & Eftimov, T. (2024). Tinytla: Topological landscape analysis for optimization problem classification in a limited sample setting. *Swarm and Evolutionary Computation*, *84*, 101448.

Pittino, F., Puggl, M., Moldaschl, T., & Hirschl, C. (2020). Automatic anomaly detection on in-production manufacturing machines using statistical learning methods. *Sensors*, *20*(8), 2344.

Price, K., Storn, R. M., & Lampinen, J. A. (2006). *Differential evolution: A practical approach to global optimization*. Springer Science & Business Media.

Pruthi, D., Dhingra, B., & Lipton, Z. C. (2019). Combating adversarial misspellings with robust word recognition. *arXiv preprint arXiv:1905.11268*.

Qayyum, A., Qadir, J., Bilal, M., & Al-Fuqaha, A. (2020). Secure and robust machine learning for healthcare: A survey. *IEEE Reviews in Biomedical Engineering*, *14*, 156–180.

Ribeiro, S. M., & de Castro, C. L. (2021). Missing data in time series: A review of imputation methods and case study. *Learning and Nonlinear Models-Revista Da Sociedade Brasileira De Redes Neurais-Special Issue: Time Series Analysis and Forecasting Using Computational Intelligence*, *19*(2).

Schäfer, P. (2015). The boss is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, *29*, 1505–1530.

Schäfer, P., & Leser, U. (2017). Fast and accurate time series classification with weasel. *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 637–646.

Seiler, M. V., Prager, R. P., Kerschke, P., & Trautmann, H. (2022). A collection of deep learning-based feature-free approaches for characterizing single-objective continuous fitness landscapes. *Proceedings of the Genetic and Evolutionary Computation Conference*, 657–665.

Shah, A., & Seghouane, A.-K. (2014). An integrated framework for joint hrf and drift estimation and hbo/hbr signal improvement in fnirs data. *IEEE transactions on medical imaging*, *33*(11), 2086–2097.

Shallue, C. J., & Vanderburg, A. (2018). Identifying exoplanets with deep learning: A five-planet resonant chain around kepler-80 and an eighth planet around kepler-90. *The Astronomical Journal*, *155*(2), 94.

Showkat, I., Khanday, F. A., & Beigh, M. R. (2023). A review of bio-impedance devices. *Medical & Biological Engineering & Computing*, *61*(5), 927–950.

Shukla, S., Hassan, M. F., Tran, D. C., Akbar, R., Paputungan, I. V., & Khan, M. K. (2023). Improving latency in internet-of-things and cloud computing for real-time data transmission: A systematic literature review (slr). *Cluster Computing*, 1–24.

Silva, I., Moody, G., Scott, D. J., Celi, L. A., & Mark, R. G. (2012). Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. *2012 computing in cardiology*, 245–248.

Škvorc, U., Eftimov, T., & Korošec, P. (2020). Understanding the problem space in single-objective numerical optimization using exploratory landscape analysis. *Applied Soft Computing*, *90*, 106138.

Sood, M., & Jain, S. (2021). Speech recognition employing mfcc and dynamic time warping algorithm. *Innovations in Information and Communication Technologies (IICT-2020) Proceedings of International Conference on ICRIHE-2020, Delhi, India: IICT-2020*, 235–242.

Tan, C. W., Bergmeir, C., Petitjean, F., & Webb, G. I. (2021). Time series extrinsic regression: Predicting numeric values from time series data. *Data Mining and Knowledge Discovery*, *35*(3), 1032–1060.

Tanabe, R. (2022). Benchmarking feature-based algorithm selection systems for black-box numerical optimization. *IEEE Transactions on Evolutionary Computation*, *26*(6), 1321–1335.

Teh, H. Y., Kempa-Liehr, A. W., & Wang, K. I.-K. (2020). Sensor data quality: A systematic review. *Journal of Big Data*, *7*(1), 11.

Tocchetti, A., Corti, L., Balayn, A., Yurrita, M., Lippmann, P., Brambilla, M., & Yang, J. (2022). Ai robustness: A human-centered perspective on technological challenges and opportunities. *arXiv preprint arXiv:2210.08906*.

Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., & Madry, A. (2018). Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*.

Um, T. T., Pfister, F. M., Pichler, D., Endo, S., Lang, M., Hirche, S., Fietzek, U., & Kulić, D. (2017). Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks. *Proceedings of the 19th ACM international conference on multimodal interaction*, 216–220.

Vanschoren, J. (2019). Meta-learning. In *Automated machine learning* (pp. 35–61). Springer, Cham.

Vasilescu, G. (2005). *Electronic noise and interfering signals: Principles and applications*. Springer Science & Business Media.

Velasco-Gallego, C., & Lazakis, I. (2020). Real-time data-driven missing data imputation for short-term sensor data of marine systems. a comparative study. *Ocean Engineering*, *218*, 108261.

Wen, Q., Sun, L., Yang, F., Song, X., Gao, J., Wang, X., & Xu, H. (2020). Time series data augmentation for deep learning: A survey. *arXiv preprint arXiv:2002.12478*.

Wenninger, M., Bayerl, S. P., Schmidt, J., & Riedhammer, K. (2019). Timage–a robust time series classification pipeline. *Artificial Neural Networks and Machine Learning–ICANN 2019: Text and Time Series: 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17–19, 2019, Proceedings, Part IV 28*, 450–461.

Wu, G., et al. (2020). Sensor drift compensation using robust classification method. *Neural Information Processing: 27th International Conference, ICONIP 2020, Bangkok, Thailand, November 23–27, 2020, Proceedings, Part III*, *27*, 687–698.

Ye, L., & Keogh, E. (2009). Time series shapelets: A new primitive for data mining. *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 947–956.

Yi, C., Yang, S., Li, H., Tan, Y.-p., & Kot, A. (2021). Benchmarking the robustness of spatial-temporal models against corruptions. *arXiv preprint arXiv:2110.06513*.

Ying, X. (2019). An overview of overfitting and its solutions. *Journal of physics: Conference series*, *1168*, 022022.

Yu, Y., Zeng, X., Xue, X., & Ma, J. (2022). Lstm-based intrusion detection system for vanets: A time series classification approach to false message detection. *IEEE Transactions on Intelligent Transportation Systems*, *23*(12), 23906–23918.

Zeng, Z., & Yan, H. (2008). Supervised classification of share price trends. *Information Sciences*, *178*(20), 3943–3956.

Zhang, W. E., Sheng, Q. Z., Alhazmi, A., & Li, C. (2020). Adversarial attacks on deep-learning models in natural language processing: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *11*(3), 1–41.

Zhao, B., Lu, H., Chen, S., Liu, J., & Wu, D. (2017). Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, *28*(1), 162–169.

Zhao, H., Des Combes, R. T., Zhang, K., & Gordon, G. (2019). On learning invariant representations for domain adaptation. *International conference on machine learning*, 7523–7532.

Zhong, G., Wang, L.-N., Ling, X., & Dong, J. (2016). An overview on data representation learning: From traditional feature learning to recent deep learning. *The Journal of Finance and Data Science*, *2*(4), 265–278.

Bibliography

Publications Comprising the Main Body of the Thesis

Journal Articles

Petelin, G., & Cenikj, G. (2024b). Random filter mappings as optimization problem feature extractors. *IEEE Access*.

Petelin, G., Cenikj, G., & Eftimov, T. (2023). Towards understanding the importance of time-series features in automated algorithm performance prediction. *Expert Systems with Applications*, *213*, 119023.

Petelin, G., Cenikj, G., & Eftimov, T. (2024). Tinytla: Topological landscape analysis for optimization problem classification in a limited sample setting. *Swarm and Evolutionary Computation*, *84*, 101448.

Publications Related to the Thesis

Journal Articles

Cenikj, G., Petelin, G., Doerr, C., Korošec, P., & Eftimov, T. (2023). Dynamorep: Trajectory-based population dynamics for classification of black-box optimization problems. *Proceedings of the Genetic and Evolutionary Computation Conference*, 813–821.

Cenikj, G., Petelin, G., & Eftimov, T. (2024a). A cross-benchmark examination of feature-based algorithm selector generalization in single-objective numerical optimization. *Swarm and Evolutionary Computation*, *87*, 101534.

Eftimov, T., Petelin, G., Cenikj, G., Kostovska, A., Ispirova, G., Korošec, P., & Bogatinovski, J. (2022). Less is more: Selecting the right benchmarking set of data for time series classification. *Expert Systems with Applications*, *198*, 116871.

Conference Paper

Cenikj, G., Petelin, G., & Eftimov, T. (2024b). Transoptas: Transformer-based algorithm selection for single-objective optimization. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 403–406.

Petelin, G., & Cenikj, G. (2023). How far out of distribution can we go with ela features and still be able to rank algorithms? *2023 IEEE Symposium Series on Computational Intelligence (SSCI)*, 341–346.

Petelin, G., & Cenikj, G. (2024a). On generalization of ela feature groups. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 419–422.

Petelin, G., Cenikj, G., & Eftimov, T. (2022). Tla: Topological landscape analysis for single-objective continuous optimization problem instances. *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, 1698–1705.

Other Publications

Journal Articles

Eftimov, T., Petelin, G., & Korošec, P. (2020). Dsctool: A web-service-based framework for statistical comparison of stochastic optimization algorithms. *Applied Soft Computing*, *87*, 105977.

Hribar, R., Hrga, T., Papa, G., Petelin, G., Povh, J., Pržulj, N., & Vukašinović, V. (2022). Four algorithms to solve symmetric multi-type non-negative matrix tri-factorization problem. *Journal of Global Optimization*, *82*(2), 283–312.

Petelin, G., Antoniou, M., & Papa, G. (2023). Multi-objective approaches to ground station scheduling for optimization of communication with satellites. *Optimization and Engineering*, *24*(1), 147–184.

Petelin, G., Hribar, R., & Papa, G. (2023). Models for forecasting the traffic flow within the city of ljubljana. *European Transport Research Review*, *15*(1), 30.

Conference Paper

Antoniou, M., Petelin, G., & Papa, G. (2020). On formulating the ground scheduling problem as a multi-objective bilevel problem. *International Conference on Bioinspired Methods and Their Applications*, 177–188.

Antoniou, M., Petelin, G., & Papa, G. (2021). Preferred solutions of the ground station scheduling problem using nsga-iii with weighted reference points selection. *2021 IEEE Congress on Evolutionary Computation (CEC)*, 1840–1847.

Cenikj, G., Petelin, G., Seljak, B. K., & Eftimov, T. (2022). Scifoodner: Food named entity recognition for scientific text. *2022 IEEE International Conference on Big Data (Big Data)*, 4065–4073.

Biography

Gašper Petelin was born in Trbovlje, Slovenia, on December 25, 1995. After completing high school, he enrolled at the University of Ljubljana's Faculty of Computer and Information Science, where he earned both his bachelor's and master's degrees. During his studies, he worked as a web developer and later as a student researcher at the Jožef Stefan Institute. Before pursuing a PhD at the Jožef Stefan International Postgraduate School, Gašper gained valuable experience as a guest scientist at Helmholtz-Zentrum Dresden-Rossendorf. During his PhD, he worked at the Department of Computer Systems at the Jožef Stefan Institute. His primary research interests include time-series analysis, black-box optimization, and representation learning.