ADAPTING SENSOR SETTINGS FOR ENERGY-EFFICIENT CONTEXT RECOGNITION

Vito Janko

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia

Supervisor: Dr. Mitja Luštrek, Jožef Stefan Institute, Ljubljana, Slovenia

Evaluation Board:
Prof. Dr. Matjaž Gams, Chair, Jožef Stefan Institute, Ljubljana, Slovenia
Prof. Dr. Gregor Papa, Member, Jožef Stefan Institute, Ljubljana, Slovenia
Prof. Dr. Juan Antonio Álvarez García, Member, University of Seville, Seville, Spain

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Vito Janko

ADAPTING SENSOR SETTINGS FOR ENERGY-EFFICIENT CONTEXT RECOGNITION

Doctoral Dissertation

PRILAGAJANJE SENZORSKIH NASTAVITEV ZA ENERGIJSKO UČINKOVITO PREPOZNAVANJE KONTEKSTA

Doktorska disertacija

Supervisor: Dr. Mitja Luštrek

Ljubljana, Slovenia, December 2019

Anyone smart enough to finish their PhD
is smart enough to never start it.

Acknowledgments

First and foremost, I wish to thank my supervisor Dr. Mitja Luštrek for his contributions in the form of ideas, constructive critique, discussions, reviews and encouragement. I would also like to thank in advance the members of my evaluation board: Prof. Dr. Matjaž Gams, Prof. Dr. Gregor Papa and Prof. Dr. Juan Antonio Álvarez García, for reading and evaluating my thesis.

There were quite a few people who helped me in this journey. I would first like to thank my family for always supporting me, occasionally feeding me, and for dealing with my frequent absence – especially in the last few years. Special thanks go to my girlfriend Maruška for tolerating me despite the difficulty of the task. I would also like to thank my friends, whose company made the whole journey worthwhile. Your support and encouragement were worth more than I can express on paper and will thus be expressed in drinks instead.

My gratitude also extends to all colleagues at the Department of Intelligent Systems, from whom I learned most of what I know about artificial intelligence and who made my days working at the institute both interesting and entertaining.

Finally, I acknowledge the financial support of the Slovenian Research Agency which funded my Young Researcher program and allowed me to keep postponing hard life choices for the last four years.

Abstract

Widespread accessibility of wearable sensing devices opens many possibilities for tracking the users who wear them. Possible applications range from measuring their exercise patterns and checking on their health, to determining their location. In this work we will use the term *context-recognition* for all these tasks.

A common problem when using such context-recognition systems is their impact on the battery life of the sensing device. It is easy to imagine that an application that monitors users' habits using all the sensors in a smartphone (accelerometer, GPS, Wi-Fi etc.) will quickly drain the phone's battery, making it useless in practice.

While many methods for reducing the energy-consumption of a context-recognition system already exist, most of them are specialized. They work either in a specific domain or can only optimize the energy consumption of specific sensors. Adapting these methods to another domain can be laborious and may require a lot of expert knowledge and experimentation.

We developed three novel methods for generating good energy-efficient solutions that are independent of the domain and can easily be used to optimize a wide range of context-recognition tasks. The first method, *Setting-to-Context Assignment (SCA)* changes the sensing settings – which sensors to use, with what frequency, what duty cycles to use etc. – depending on what context was last detected. To do so, we developed a mathematical model that can predict the performance of any given setting-to-context assignment. The SCA method then uses the NSGA-II algorithm to search the space of possible assignments, finding the best ones. The second method, *Duty-Cycle-Assignment (DCA)* works in a similar way to the SCA method, but is specialized in optimizing only duty-cycling, i.e., periodically turning the sensors on and off. Finally, the *Cost-Sensitive Decision-Tree (CS-DT)* method adapts sensing settings directly to the sensor data. It works by using a cost-sensitive decision tree that was adapted for context-recognition tasks. All three methods were then combined in three different ways in order to join their individual strengths. Two of these combinations adapt to both the current context and to the sensor data.

Our methodology was tested on four different real-life datasets, and on a family of artificial datasets. Doing so, we proved that it works under many different conditions, with different possible settings, sensors and problem domains. We also showed that our methods compare favourably against other state-of-the-art methods.

Instead of returning only one solution, all of our methods can find different trade-offs between the classification quality and the energy consumption. Returning these trade-offs can help the system designer to pick the most suitable for their system. For each of the tested datasets we found many energy-efficient solutions. For example, for the Commodity12 dataset, we reduced the energy consumption from 123 mA to 29 mA in exchange for less than 1 percentage point of accuracy. For another example, we were able to use only 5% of the available data in the SHL dataset (by using a lower frequency, duty-cycling and a subset of sensors) and in exchange sacrifice only 5 percentage points of the accuracy.

Povzetek

V današnjem času narašča uporaba različnih prenosnih senzorskih naprav, kot so pametni telefoni ali pametne zapestnice. Te naprave lahko na različne načine spremljajo uporabnike, ki jih nosijo, kar omogoča, da lahko med drugim spremljajo njihovo zdravje, jih spodbujajo h gibanju ali pomagajo pri navigaciji. Tako zaznavanje in interpretiranje senzorskih podatkov bomo v tem delu poimenovali *prepoznavanje konteksta*.

Pogosta težava pri uporabi sistemov za prepoznavanje konteksta je njihov vpliv na porabo baterije senzorske naprave. Lahko si je predstavljati, da bo aplikacija, ki spremlja navade uporabnika z vsemi senzorji na pametnem telefonu (pospeškomer, GPS, Wi-Fi itd.), hitro spraznila baterijo telefona in bila zato v praksi neuporabna.

Veliko metod za nižanje porabe energije sistemov za prepoznavanje konteksta že obstaja, vendar jih je večina specializiranih. Delujejo samo v specifičnih domenah ali optimizirajo porabo energije specifičnih senzorjev. Prilagoditev teh metod za novo domeno je pogosto časovno potratna in lahko potrebuje veliko strokovnega znanja o tej domeni.

V tem delu smo razvili tri nove algoritme, ki generirajo energijsko učinkovite rešitve, so domensko neodvisni in lahko optimizirajo širok razpon nalog prepoznavanja konteksta. Naša prva metoda, imenovana SCA (*Setting-to-Context Assignment*), spreminja nastavitve sistema glede na to, kateri kontekst je bil nazadnje zaznan. Nastavitve so lahko vse od tega, kateri senzorji so v uporabi, s kako frekvenco zajemajo podatke, do tega, kakšni so njihovi obratovalni cikli (tj. periodično prižiganje in ugašanje senzorjev). Na tak način dobimo prireditev nastavitev glede na kontekst. Razvili smo matematični model, ki lahko napove, kako dobro se obnese poljubna prireditev, nato pa uporabili algoritem NSGA-II, da poišče najboljše prireditve. Naša druga metoda, imenovana DCA (*Duty-cycle Assignment*), dela podobno prireditev, vendar optimizira samo obratovalne cikle senzorjev. Zadnja metoda, imenovana CS-DT (*Cost-Sensitive Decision Tree*), prilagaja nastavitve sistema neposredno senzorskim podatkom. To stori z uporabo odločitvenih dreves, ki upoštevajo energijsko ceno atributov. Vse tri metode smo med seboj tudi kombinirali na tri različne načine. Te kombinirane metode združijo prednosti posameznih metod in lahko nastavitve sistema prilagajajo tako na prepoznan kontekst kot na senzorske podatke.

Naša metodologijo smo testirali na štirih različnih podatkovnih množicah iz resničnega sveta in na eni umetno generirani družini podatkovnih množic. Na ta način smo pokazali, da je metodologija primerna v različnih razmerah, z različnimi nastavitvami in senzorji. Pokazali smo tudi, da so v vseh primerih pridobljene rešitve primerljive s tistimi, ki jih generirajo najsodobnejše metode, ali boljše od njih. Naše metode lahko, namesto da bi izbrale samo eno rešitev, najdejo različne rešitve, vsako z drugačnim kompromisom med klasifikacijsko točnostjo in porabo energije, načrtovalec sistema pa izbere tisto, ki je zanj najprimernejša. Izpostavimo lahko primer, v katerem smo porabo energije pri podatkovni množici Commodity12 zmanjšali s 123 mA na 29 mA, pri čemer smo žrtvovali manj kot 1 odstotno točko klasifikacijske točnosti. V drugem primeru pa smo pri podatkovni množici SHL uporabili le 5% podatkov (z uporabo manj senzorjev in obratovalnimi cikli) in v zameno izgubili le 5 odstotnih točk klasifikacijske točnosti.

Contents

List of Figures **xvii**

List of Tables **xxiii**

List of Algorithms **xxvii**

Abbreviations **xxix**

Symbols **xxxi**

Glossary **xxxiii**

1 Introduction **1**
1.1 Challenges . . . 2
1.2 Approach . . . 3
1.3 Hypotheses and Goals . . . 5
1.4 Scientific Contributions . . . 5
1.5 Thesis Overview . . . 6

2 Background and Related Work **7**
2.1 Markov Chains . . . 7
2.2 Multi-Objective Optimization . . . 9
2.2.1 Multi-Objective Optimization Basic Properties . . . 10
2.2.2 Solving Multi-Objective Optimization Problems . . . 11
2.3 Context Recognition . . . 11
2.3.1 Methodology . . . 12
2.4 Energy-Efficient Context Recognition . . . 13
2.4.1 Optimization Modalities . . . 13
2.4.1.1 Sampling Frequency . . . 13
2.4.1.2 Duty-Cycling . . . 14
2.4.1.3 Sensor Subset Selection . . . 15
2.4.1.4 Reducing Processing Time . . . 15
2.4.1.5 Specialized Hardware . . . 16
2.4.2 Dynamically Adapting Sensor Settings . . . 17
2.4.2.1 Based on Context . . . 17
2.4.2.2 Based on Data . . . 18
2.4.2.3 Cost-Sensitive Learning . . . 19
2.4.3 Comparing Our Methods to the Related Work . . . 19

3 Methodology **21**
3.1 Setting-to-Context Assignment (SCA) . . . 21
3.1.1 Modeling Assignment Performance . . . 22

3.1.1.1 Simple Evaluation Model . . . 22
3.1.1.2 Markov-Chain Evaluation Model . . . 23
3.1.1.3 Different System Schemes . . . 25
3.1.2 Finding Sensible Assignments . . . 26
3.1.2.1 Theoretical Bounds . . . 27
3.1.2.2 Multi-Objective Optimization . . . 28
3.2 Duty-Cycle Assignment (DCA) . . . 29
3.2.1 Modeling Duty-Cycles . . . 29
3.2.1.1 Naive Evaluation . . . 30
3.2.1.2 Duty-Cycle Types . . . 31
3.2.1.3 Distribution of Duty-Cycle Types . . . 32
3.2.1.4 System Evaluation . . . 32
3.2.2 Finding Duty-Cycle Parameters . . . 33
3.2.2.1 Iterative Method . . . 33
3.2.2.2 Multi-Objective Optimization . . . 33
3.3 Cost-Sensitive Decision Trees (CS-DT) . . . 34
3.3.1 Cost-Sensitive Model . . . 35
3.3.2 Using CS-DT for Context Recognition . . . 35
3.3.2.1 Delayed Sensor Activation . . . 35
3.3.2.2 Shared Feature Costs . . . 36
3.3.2.3 Taking Advantage of Shared Feature Costs . . . 37
3.3.2.4 Advanced Use of Shared Feature Costs . . . 37
3.3.3 Generating Different CS-DTs . . . 37
3.3.3.1 Changing Misclassification Cost . . . 37
3.3.3.2 Using Different Settings . . . 37
3.3.3.3 One-vs-all Trees . . . 38
3.3.3.4 Context-Specific Trees . . . 38
3.4 Combining the Methods . . . 38
3.4.1 Combining the SCA and DCA Methods . . . 38
3.4.2 Combining the SCA and CS-DT Methods . . . 40
3.4.3 Combining All Three Methods . . . 40

4 Datasets **43**
4.1 Artificial Datasets . . . 43
4.2 SHL Challenge . . . 45
4.3 Opportunity . . . 47
4.4 Commodity12 . . . 48
4.5 E-Gibalec . . . 50

5 Experiments **53**
5.1 Artificial Datasets . . . 53
5.1.1 SCA Method Experiments . . . 53
5.1.1.1 Evaluating the Accuracy of the Mathematical Model . . . 53
5.1.1.2 Using the Mathematical Model for Finding Good Solutions 54
5.1.1.3 Two-Step Classification . . . 57
5.1.2 DCA Method Experiments . . . 57
5.1.2.1 Evaluating the Accuracy of the Mathematical Models . . . 58
5.1.2.2 Using the Mathematical Model for Finding Good Solutions 61
5.1.3 CS-DT Method Experiments . . . 64
5.1.4 Comparing and Combining methods . . . 66
5.1.4.1 CS-DT and SCA Combination . . . 66

5.1.4.2 SCA and DCA Combination . . . 68
5.1.4.3 SCA, DCA and CS-DT Combination . . . 68
5.2 Real-Life Datasets . . . 72
5.2.1 SHL Experiments . . . 73
5.2.1.1 SCA . . . 73
5.2.1.2 DCA . . . 74
5.2.1.3 SCA and DCA Combination . . . 77
5.2.2 Opportunity Experiments . . . 78
5.2.2.1 SCA . . . 78
5.2.2.2 DCA . . . 81
5.2.2.3 SCA and DCA Combination . . . 82
5.2.3 Commodity12 Experiments . . . 84
5.2.3.1 SCA . . . 84
5.2.3.2 DCA . . . 87
5.2.3.3 SCA and DCA Combination . . . 88
5.2.3.4 CS-DT . . . 90
5.2.3.5 CS-DT and SCA Combination . . . 90
5.2.3.6 SCA, DCA and CS-DT Combination . . . 91
5.2.4 E-Gibalec Experiments . . . 93
5.2.4.1 Dealing with Unknown Transition Probabilities . . . 93
5.3 Comparing to Related Work . . . 97
5.3.1 Simple Approach . . . 97
5.3.2 Settings Based on the Probable Next Context . . . 99
5.3.3 Episodic Sampling . . . 99
5.3.4 Datum-Wise Sampling Frequency . . . 101
5.3.5 Expert Knowledge . . . 102

6 Conclusion **103**
6.1 Performance . . . 104
6.2 Hypotheses, Goals and Scientific Contributions . . . 105
6.3 Limitations and Future Work . . . 106

References **107**

Bibliography **115**

Biography **119**

List of Figures

Figure 2.1: Visual representation of a Markov chain. 7

Figure 2.2: Visual representation of a Pareto front. For each of the solutions (A, B, C, etc.) the values of two objectives are shown, one on each axis. Given the task is the one of minimization, the lower and the more left the points stand, the better. Point A, point B and all the dark blue points are parts on the Pareto front. Point C is dominated by the Point A (as well as Point B) and is thus not on the Pareto front. The same goes for all light blue circles. Do note that comparing any two points on the front, each will be better in one objective than another. We thus cannot say that one is better than another. 9

Figure 2.3: Visual representation of different Pareto front approximations. Only the connecting lines are shown, with the individual solutions implied to stand in each of the corners that point to the coordinate system origin and at line extremes. Each front is labeled by a letter (A–C). The reference point used for hyper-volume calculation is shown as R. The hyper-volume of the Pareto front approximation C is calculated as the area of the shape bounded by front C and gray lines. 10

Figure 3.1: Markov chain where each state represents a (true, predicted) context of a subject. The states are fully connected, including connections to themselves, although most transitions were omitted for clarity. Transition A, for example, models the situation where the subject is continually resting, and their context is recognized as such. For transition B, the subject started walking, which was correctly recognized. Finally, transition C implies an incorrect classification of walking as running. The example comes from the E-Gibalec dataset, which is described in Section 4.5. 24

Figure 3.2: Structure of a $\langle t, p \rangle$ duty-cycle. It starts with the context t classified as p, followed by the sleeping phase (sensors not working) where all contexts are classified as p, and an active phase (sensors working). The last period in the active phase is the head of the next cycle. 31

Figure 3.3: Demonstrating the delayed sensor activation in a CS-DT. We start (A) by having only the accelerometer active. Using it we traverse the first half of the tree, where we need the value of a feature that is derived from the GPS sensor. Not having it, we make the classification as if that node was a leaf node and then turn the GPS on. In the next instance (B) we can traverse the tree as normal and make the correct classification. This continues for n steps, after which (C) we go to the left part of the tree where no GPS feature is required. As a result, in the next instance the GPS is turned off. 36

Figure 4.1: Acceleration and magnetic field data samples (both taken from the x-axis). A four-second sample is shown for each activity. 46

Figure 4.2: All activities in the SHL dataset, connected if the probability of a transition from one to another is more than 30%. 47

Figure 4.3: Activity distribution for the E-Gibalec dataset. 51

Figure 4.4: Classification accuracy in the E-Gibalec dataset using different sampling frequencies of the accelerometer. 52

Figure 5.1: Comparison of assignments where all contexts have the same sensor assigned (Static solutions), to the assignments where active sensors dynamically change based on the last classified context (SCA). This comparison was done on the *Full* dataset. 55

Figure 5.2: Assignments from Figure 5.1 evaluated with both mathematical models and compared to the values achieved by a simulation. This experiment was done for both the *Full* (left) and *Hidden* (right) dataset. On the *Full* dataset the SCA model makes almost perfect predictions while the Simple model only roughly hits the correct area. On the *Hidden* dataset, the SCA predictions were slightly less accurate, but still better than the Simple method predictions, which form curious clusters around a few different classification accuracies. 56

Figure 5.3: Two solutions found using the simple model (blue dots), compared to all solutions found with the combination of SCA model and the NSGA-II algorithm (*Full* dataset was used). The solution A was found by assigning a setting for each context in isolation, while the solution B is the only solution returned by NSGA-II algorithm that uses the simple model for evaluation. The assignment B uses no sensors for the most common context; this results in a system that eventually reaches that context and is then "stuck" using a majority classifier, classifying into that context over and over again. The simple model cannot accurately model this situation and thus over-optimistically predicts the performance of assignment B. 56

Figure 5.4: Solutions found when using two-step classification on the *Full* dataset. Solutions are compared to those that were achieved using the SCA method. 57

Figure 5.5: Predictions of the accuracy of the system when using different duty-cycle lengths on *Set 1*. Predictions are made with different mathematical models, as well as actually running the simulation. The duty-cycle length represents the number of periods spent sleeping. The length of the active phase w on current and similar figures is always 1 period, unless stated otherwise. 58

Figure 5.6: Predictions of the accuracy of the system when using different duty-cycle lengths. Different scenarios are tested: (top left) – duty-cycle lengths are long, (top right) – contexts have different average lengths, (middle left) – more than two contexts, (middle right) – more than two contexts and different average context lengths, (bottom left) – misclassification errors are present, (bottom right) – active phase is long. . . . 59

Figure 5.7: Comparison of the assignments found using DCA method to assignments where all contexts are assigned a duty-cycle of the same length (Static solutions). The resulting set of non-dominated solutions is shown in two different representations: (left) energy gain – the number of periods sleeping for each active one, (right) energy – assuming that an always active sensor has the energy cost of 1. 61

Figure 5.8: Assignments that correspond to the trade-offs in Figure 5.7 are represented here by the assigned duty-cycle length for each of the three different contexts. The order of IDs is the same as in the right part of Figure 5.7 – longer duty-cycles correspond to the solutions where the classification error is high, but energy is low. Flat line at the end of the long context length is a result of the search space being limited to the length of 20. While it is apparent that longer contexts have longer duty-cycles assigned, the ratio between the lengths varies. 62

Figure 5.9: Comparison of the assignments found using DCA method to assignments where all contexts are assigned a duty-cycle of the same length (Static solutions). In this case *Set7* was used and miclassifications were assumed to happen, thus the accuracy is never 100%. 63

Figure 5.10: (left) The structure and the probability of the context transitions. The contexts *Long 1* and *Long 2* have the same average length, while the context *Short 1* is shorter on average. The optimization objective is the recall for the context *Short 1*. (right) Duty-cycle lengths for all three contexts, for all found non-dominated assignments. Context *Short 1* has always the maximal duty-cycle length assigned due to the nature of the optimization criterion. 63

Figure 5.11: The mean of the hypervolume over 30 runs. The gray lines denote the mean +/- the standard deviation over the same 30 runs. 64

Figure 5.12: Cost-sensitive decision trees, built with different costs of sensors. The nodes represent features calculated by the data of the listed sensor. Note that sensor data was artificially generated for this dataset. Only one branch is listed for clarity. 65

Figure 5.13: Trade-offs between energy consumption and classification accuracy achieved by different CS-DTs on the *Hidden* dataset. Alongside are the results of the same decision trees, modified so they do not employ sensor switching. A regular decision tree would perform the same as the most top-left CS tree without sensor switching. 65

Figure 5.14: Comparison of trade-offs generated by different methods on the *Hidden* dataset. 66

Figure 5.15: Comparison of trade-offs generated by different sets of cost-sensitive decision trees on the *Hidden* dataset. 67

Figure 5.16: (Left) Comparison between different (started with different tree set) CS-DT and SCA methods combinations. (Right) Comparison between using CS-DT and SCA methods individually against using their combination. 67

Figure 5.17: Comparison of trade-offs generated by applying the DCA method on the marked trade-offs (blue points) generated by the SCA method. . . . 68

Figure 5.18: Comparison of trade-offs generated by applying the DCA method on the marked (blue points) trade-offs generated by the CS-DT + SCA method. 69

Figure 5.19: Comparison of trade-offs generated by all presented methods on the *Hidden* dataset. 69
Figure 5.20: Comparison of trade-offs generated by all presented methods on the *Hidden* dataset. In this case of the CS-DT + SCA + DCA combination we used limitations on the DCA parameters. 70
Figure 5.21: An example of using a combination of all three methods. For each consecutive time step, we list the sensors used, CS-DT tree used and the predicted activity. 71
Figure 5.22: CS-DT + SCA + DCA predictions on the *Hidden* dataset, compared to the actual performance values achieved in a simulation. 72
Figure 5.23: Comparison of assignments, where all contexts have the same sensor combination assigned (Static solutions), to the assignments where active sensors dynamically change based on the last classified context (SCA). Some points of interest are marked with letters A-D. This comparison was done on the SHL dataset. 73
Figure 5.24: Comparison of assignments on the SHL dataset, where all contexts have the same duty-cycle length assigned (Static solutions), to the assignments where duty-cycles lengths dynamically change based on the last classified context (DCA). 75
Figure 5.25: DCA predictions on the SHL dataset, compared to the actual performance values achieved in a simulation. 76
Figure 5.26: SCA, DCA predictions on the SHL dataset, generated by starting at different SCA solutions (marked with points 1-5). 77
Figure 5.27: Comparison of SCA, DCA and their combined solutions on the SHL dataset. Two solutions are singled out as reasonable trade-offs between accuracy and energy consumption. 78
Figure 5.28: Comparison of assignments on the Opportunity dataset, where all contexts have the same sensor assigned (Static solutions), to the assignments where active sensors dynamically change based on the last classified context (SCA). A point of interest is marked and shown explicitly in Table 5.12 80
Figure 5.29: Comparison of assignments on the Opportunity dataset, where all contexts have the same duty-cycle length assigned (Static solutions), to the assignments where duty-cycles lengths dynamically change based on the last classified context (DCA). Some points of interest are marked with letters A-E. 82
Figure 5.30: Comparison of assignments on the Opportunity dataset, made by three different methods. Some points of interest are marked with letters A-D. 83
Figure 5.31: Every solution in Figure 5.30 evaluated again using a simulation. Simulated values are a close match to the predicted ones. 83
Figure 5.32: Comparison of the assignments, where all contexts have the same sensor assigned (Static solutions), to the assignments where active sensors dynamically change based on the last classified context (SCA). Solutions for two users (in no particular order) are plotted on the left, and the remaining two on the right. Some sample solutions are marked with points labeled A1-B4. Comparison was performed on the Commodity12 dataset. 84
Figure 5.33: Comparison of trade-offs that were generated by using assignments created by using the data from User 1, and applying them to data for each of the four users. 86

Figure 5.34: Comparison of the assignments, where all contexts have the same duty-cycle length assigned (Static solutions), to the assignments found by the DCA method. Solutions for two users are plotted on the left, and the remaining two on the right. The comparison was done on the Commodity12 dataset. 87
Figure 5.35: Comparison of the SCA and DCA methods to their combination on the Commodity12 dataset. Each figure presents the found solutions of one of the users (1–4) in the order: top left, top right, bottom left, bottom right. 89
Figure 5.36: (left) Comparison of the trade-offs, where different CS-DT construction methods are used. (right) Comparison of trade-offs, where different tree sets were used for the CS-DT + SCA method combination. Both cases were taken when optimizing for User 1 on the Commodity12 dataset. . 90
Figure 5.37: Parts of the CS-DTs assigned to the contexts *work* and *out* for User 1. Orange nodes show which sensor must be activated. The left-most blue node indicates for which context the corresponding tree is used, other blue nodes indicate to which context to classify. 91
Figure 5.38: Comparison of the SCA method to different method combinations on the Commodity12 dataset. Each figure presents the found solutions of one of the users (1–4) in the order: top left, top right, bottom left, bottom right. 92
Figure 5.39: Trade-offs generated by each of the methods and their combinations on the E-Gibalec dataset. The baseline (using the maximum sensors frequency and no duty-cycling) is marked for comparison. 94
Figure 5.40: SCA + DCA + CS-DT predictions, compared to the actual performance values achieved in a simulation on the E-Gibalec dataset. 94
Figure 5.41: Different trade-offs between accuracy and energy consumption that occur when the average duration of activities changes if using the assignment from Table 5.26. The orange point represents the trade-off achieved when using the original activity lengths. Other points represent modifying either *running* activity average duration or *other* activity average duration. 96
Figure 5.42: SCA + DCA + CS-DT combination trade-offs, compared to the trade-offs generated by the same assignments, but with a change in the underlying activity distribution. Comparison was done on the E-Gibalec dataset. 96
Figure 5.43: Comparison of our methods to three alternatives. Simple A and Simple B (or just A and B) mark the results from Section 5.3.1. EK1 marks the result from expert knowledge rules from Section 5.3.5, while the EK2 marks the result from the rules after improvement. All tests were performed on four datasets, with the exception of the expert knowledge approach. Top-left: SHL, top-right: Opportunity, bottom-left: Commodity12, bottom-right: E-Gibalec. 98
Figure 5.44: Comparison of our methods to the episodic sampling on four different datasets. Top-left: SHL, top-right: Opportunity, bottom-left: Commodity12, bottom-right: E-Gibalec. 100
Figure 5.45: Comparison of our methods to the Datum-Wise Sampling Frequency method that uses either a decision tree (DW-DT) or a Random forest (DW-RF) classifier. This comparison was done on the E-Gibalec dataset. 101

List of Tables

Table 4.1: The average duration (number of instances) of each context for the artificial datasets 1-9. The number of columns in each row implies the number of contexts in that dataset. As an additional note, in most datasets the contexts have the same transition probability to every other context. Exceptions to this rule are the Set6, where the transition matrix was randomly generated; in Set8 only the second context can transition to the third. In the set9 the transition matrix is crafted so the transition probabilities are as different among themselves as possible. 44

Table 4.2: Average duration of each activity in the SHL dataset. 47

Table 4.3: The average duration [min] of each activity for different users in the Commodity12 dataset. 50

Table 4.4: Energy cost of different sensor combinations measured in mA. Sensors that were active were labeled A – accelerometer (50 Hz), A – accelerometer (5 Hz), S – sound, B – Bluetooth, W – WiFi. 51

Table 5.1: A random assignment made for the *Hidden* dataset. Whenever the context in the top row is detected, the sensors in the bottom row are activated. A-Accelerometer, G-GPS, W-Wi-Fi. 54

Table 5.2: The difference (mean absolute error) between the predicted accuracy/energy and the one calculated in a simulation. The experiment was repeated for both the *Full* and *Hidden* datasets, for both Simple and SCA method and for both random assignments and assignments found by NSGA-II method. The difference in accuracy was measured in percentage points (p.p). The energy was defined in abstract units, with values ranging from 0 to 44 (the maximum value reached when using all sensors). . . . 54

Table 5.3: Time of evaluation of the mathematical models (both the SCA and Simple model) compared to the time of the simulation. 54

Table 5.4: MAE (measured in percentage points) between the predicted accuracy and one calculated in a simulation. DCA method clearly outperforms the simple approximations. 60

Table 5.5: MAE (measured in percentage points) for the DCA method between predicted accuracy/energy gain and one calculated in a simulation. Two sets of experiments were conducted – in the first the same duty-cycle length was assigned to all contexts, in the second the duty-cycle lengths varied based on the classified context. 60

Table 5.6: The time of evaluation of the DCA model compared to the time of the simulation of duty-cycling on the whole dataset. The test was performed with different number of contexts (*Set1–Set4*), using both base and optimized implementation. 60

Table 5.7: The sensors used when the corresponding activity is detected for each of the four solutions marked in Figure 5.23. Sensors are abbreviated: acc-accelerometer, mag-magnetometer, ori-orientation sensor, gyr-gyroscope, bar-barometer. Both accuracy and energy costs (measured in the percentage of data used) are provided as well. 74
Table 5.8: Time of evaluation of the mathematical models compared the time of the simulation. Test was repeated and averaged over 100 random assignments. 74
Table 5.9: Difference (mean absolute error) between the predicted accuracy/energy and the one calculated in a simulation for the SHL dataset. The experiment was repeated for both Simple, SCA and DCA method and for both 100 random assignments, and the assignments that form the corresponding Pareto front approximation. The difference was measured in percentage points (p.p). 75
Table 5.10: Sample solutions returned by the DCA methods for the SHL dataset. For each activity the assigned duty-cycle length is given (the length of the sleeping phase of the cycle). In addition, both the accuracy and energy costs are listed. Duty-cycle length of 0 implies that duty-cycling should not be used. 76
Table 5.11: Time of evaluation of the DCA model compared to the time of the simulation. The test was repeated and averaged over 100 random assignments. 76
Table 5.12: For each object we list sensor locations to be activated if the assignment in Figure 5.28 is selected. Some locations of interest are bolded. This assignment uses 5 sensor locations on average and has an *F-score* of 0.42. 79
Table 5.13: Time of evaluation of the mathematical models compared to the time of the simulation. The test was repeated and averaged over 100 random assignments. 80
Table 5.14: Difference (mean absolute error) between the predicted F-score/average number of active sensors and the one calculated in a simulation for the Opportunity dataset. The experiment was repeated for both Simple and SCA method and for both 100 random assignments, and the assignments that form the corresponding Pareto front approximation. 80
Table 5.15: For each object we list the length of the duty-cycle assigned in addition to the recall for that object in case no duty-cycling is used. 81
Table 5.16: Time of evaluation of the DCA model compared to the time of the simulation. The test was repeated and averaged over 100 random assignments. 81
Table 5.17: Difference (mean absolute error) between the predicted F-score/average number of active sensors and the one calculated in a simulation for the Opportunity dataset. The experiment was repeated for both 100 random assignments, and the assignments that form the corresponding Pareto front approximation. 82
Table 5.18: The sensors used when the corresponding context is detected for each of the eight solutions marked in Figure 5.32. Some sensors are abbreviated: sqr - FourSquare webservice, wifi - Wi-Fi, acc - (accelerometer, pressure, light), hr - (hear rate, respiration rate). Multiple sensors are listed in brackets as they all share a cost and are thus either all included or excluded. Some contexts/user combinations have no sensor assigned as that user was never in that context in the training part of the dataset. . 85
Table 5.19: Time of evaluation of the mathematical models compared to the time of the simulation. The test was repeated and averaged over 100 random assignments on the Commodity12 dataset. 86

Table 5.20: The difference (mean absolute error) between the predicted accuracy/energy and the one calculated in a simulation for the Commodity12 dataset. The experiment was repeated for each user, for both the Simple and SCA method and for both 100 random assignments and assignments that form the corresponding Pareto front approximation. Difference in accuracy was measured in percentage points, while the difference in energy is expressed in mA. 86
Table 5.21: Time of evaluation of the DCA model compared to the time of the simulation. The test was repeated and averaged over 100 random assignments. 87
Table 5.22: Difference (MAE) between the predicted accuracy/energy and the one calculated in a simulation. The experiment was repeated for each user. . 87
Table 5.23: Sample solutions returned by the DCA methods for the Commodity12 dataset. For each activity the assigned duty-cycle length is given. In addition, both the accuracy [%] and energy costs [mA] are listed. 88
Table 5.24: Comparison between energy consumption for each of the users before and after the optimization using the SCA + DCA combination. Note that using no sensors has a baseline cost of 20 mA. 88
Table 5.25: Time of the evaluation [ms] of the mathematical models compared to the time of the simulation. The test was repeated and averaged over 100 random assignments on the E-Gibalec dataset. 93
Table 5.26: One assignment found using the SCA + DCA method. The sampling frequency and the duty-cycle lengths are shown for each activity. In such a system, after the activity is determined, one instance is classified using the listed frequency, then the system sleeps for the listed duration. . . . 95

List of Algorithms

Algorithm 3.1: CS-DT + SCA + DCA (Create assignments) 41
Algorithm 3.2: CS-DT + SCA + DCA (Use assignments) 42

Abbreviations

CoH	...	Creatures of Habit
CS-DT	...	Cost-Sensitive Decision-Tree method (only used when referring to the adapted decision tree proposed in our thesis)
DCA	...	Duty-Cycle Assignment method
FFT	...	Fast Fourier Transform
GPS	...	Global Positioning System
LPC	...	Linear Predictive-Coding
mA	...	Milliampere
MAE	...	Mean Absolute Error
MFCC	...	Mel-Frequency-Cepstral-Coefficient
NSGA-II	...	Non-dominated Sorting Genetic Algorithm II
NP	...	Nondeterministic Polynomial time
p.p.	...	Percentage Point
PSD	...	Power Spectral Density
SCA	...	Setting-to-Context Assignment method
SCA + DCA	...	A combination of SCA and DCA methods (same goes for other methods separated with +)
SHL	...	Sussex-Huawei Locomotion
std	...	Standard Deviation
SVM	...	Support Vector Machine

Symbols

a	... the assignment of settings to contexts. $a(c)$ represents the setting s assigned to context c
$\mathrm{Acc}(c\|s)$	... the accuracy for recognizing context c, given the setting s is active
avg	... the average context length
C	... the set of contexts
c	... a specific context
$\langle c_1, c_2 \rangle$	... Markov-chain state that represents the true context is c_1 that is predicted as c_2
CM	... the confusion matrix. $CM_{i,j}$ represents the probability that the context i is classified as context j
$CM_s(c_1\|c_2)$	... the probability that the classifier that works with setting s will classify an instance to c_1, if the true context is c_2. Note that the indices have a slightly different meaning than in the previous case
D	... the distribution of the duty-cycle types $D_{i,j}$ is the proportion of $\langle i, j \rangle$ cycles
ea	... energy consumption during the active phase of duty-cycling
$EA_{t,p}$	... the average number of contexts that appear while in the active duty-cycle phase, given that this cycle started with the true context c_t that was classified as c_p
e_i	... the i-th base vector
$e(s)$	... the energy requirement of the setting s per time unit
es	... energy consumption during the sleeping phase of duty-cycling
$ES_{t,p}$	... the average number of contexts that appear while in the sleeping duty-cycle phase, given that this sleeping period started with the true context c_t that was classified as c_p
f_i	... the i-th objective function
$gain$	... the proportion of sleeping phase length compared to the active phase length
len	... the length of the sleeping period. $len(c)$ represents the length of the sleeping period that is assigned to context c
M	... the set of all states in the Markov chain
$P_{t,i}$	... the probability distribution of contexts after i steps, given the current context c_t
$P(mc_1 \to mc_2)$	... the probability of transition from a Markov-chain state mc_1 to the state mc_2
$p(c)$	... the probability of context c, estimated by its proportion in the dataset
r_i	... the i-th feature
S	... the set of settings
s	... a specific setting

T	...	the transition matrix. $T(c_1, c_2)$ represents the probability in the dataset that the next context will be c_1 given that the current one is c_2
$t(\langle c, c' \rangle)$	...	the predicted proportion of time spent in state $\langle c, c' \rangle$
w	...	the length of the active period
X	...	the set of feature vectors
Y	...	the set of classes

Glossary

Context recognition	...	the task of transforming wearable-sensor data into something semantically relevant about the user who wears the sensor (see Section 2.3)
Context	...	the target of a context-recognition task. In a context-recognition dataset, context corresponds to the label of that dataset. E.g., user's activity, location, stress level
Context-specific trees	...	a set of cost-sensitive decision trees, where each tree is only trained on data from one context and data that is subsequent to that context (see Section 3.3.3.4)
Cost-sensitive decision tree	...	a decision tree that tries to optimize both the classification accuracy and the cost of the features used. In our thesis the cost was the systems energy consumption (see Section 3.3)
Duty-cycling	...	the process of periodically switching sensors on and off again (see Section 2.4.1.2)
Duty-cycle assignment	...	the same as the setting-to-context assignment (also in glossary), but the settings are limited to the duty-cycle settings (see Section 3.2)
Markov chain	...	a system that consists of states and the probabilities to transition between them. These probabilities depend only on the current state and not on any other factor, including the previous history of transitions (see Section 2.1)
Misclassification trees	...	a set of cost-sensitive decision trees created by varying the ratio between the feature and misclassification costs (see Section 3.3.3.1)
One-vs-all trees	...	a set of cost-sensitive decision trees, each trying to differentiate one context from others (see Section 3.3.3.3)
Pareto front	...	best solutions in a multi-objective optimization problem, presented in the objective space. Best is defined as not being dominated by any other solution (see Section 2.2)
Non-dominated set	...	a subset of solutions that are not dominated by any other solutions in the given set. One solution dominates another if its better in one objective and better or equal in all others (see Section 2.2)
Setting	...	a parameter of the context-recognition system that can be modified in order to increase its energy-efficiency. E.g., sensors used, sensor sampling frequency, type of features calculated
Setting-to-context assignment	...	an assignment where each context is assigned a setting. A system that uses such an assignment then switches between settings in runtime. When a context is detected, the corresponding setting is used (see Section 3.1)

Setting trees	...	a set of cost-sensitive decision trees, each created by starting with a different system setting, e.g., different sensor subset. (see Section 3.3.3.2)
Simulation	...	the process of going through the dataset and classifying each instance one by one. After each classification, the system settings are changed based on the classified context, and are used for the next classification. The word "Experiment" is also used for the same process
Steady state	...	the probability distribution of Markov-chain states, to which every other probability distribution converges given infinite time steps (see Section 2.1)

Chapter 1

Introduction

In recent years, we have experienced a rise of popularity of smart wearable devices. Practically everyone owns a smartphone, with other devices like smart watches slowly following this trend [1]. All these devices can use sensors to track the users who wear them, detecting everything from their heart rate and movement to their location.

Collected sensor data can then be processed and interpreted using artificial-intelligence methods. We will call this task of transforming sensor time-series data into something semantically relevant about the user, *context recognition*. For example, we could recognize if the user is walking, running, resting or performing similar activities using accelerometer data. Sensing with multiple sensors, possibly at once, opens additional options for context recognition, for example recognizing contexts such as shopping, traveling or work.

Context-recognition applications can be very convenient for the user, e.g., motivating them to exercise, helping them navigate, changing their smartphone settings based on their surroundings, etc. In some cases, they may also be used to assist their health management by monitoring their physiological signals, analyzing their activity patterns or even detecting if they fell and injured themselves. As a result, they are widely adopted [2][3][4] and often come pre-installed on wearable devices.

However, a major limitation of such continuous sensing and context recognition is its heavy toll on the sensing device's battery life. This is especially relevant for smartphones, which have a very limited battery that must be shared between many applications, but the same limitation applies to basically any wearable device. This issue is often neglected when discussing the design of a context-recognition system, however, it is an essential component if such systems are to be used in practice. It is easy to see that if an application halves the battery life of a smartphone, it will probably not be adopted by many users, regardless of its context-recognition ability.

There is an inherent trade-off between a system's energy consumption and its recognition quality. Increasing energy savings decreases the recognition quality and vice versa. An example of this would be decreasing the sampling frequency of a sensor, affecting both the energy required to run it and the quality of the data it provides.

In this thesis we explore this problem of creating energy-efficient solutions that present good trade-offs between a system's energy consumption and its recognition quality. We aim to do so in a way that is as general as possible, without depending on expert knowledge, a specific domain, or an exact type of settings to be optimized.

1.1 Challenges

Some solutions for conserving the battery life of a device try to reduce the energy consumption by finding the ideal sampling rate of sensors or by turning them on and off periodically (duty-cycling). Finding the best such sensor setting for a particular context-recognition task is already a challenging task, but such a solution might be suboptimal. We might, for example, want to have the GPS active at a high sampling rate while the user is driving, but at a very low rate when they are working in the office. This calls for dynamic changes in the settings for both the sensors and for the subsequent processing.

Many adaptive approaches already exist and a comprehensive review of them can be found in Section 2.4. Most of them rely on expert knowledge of the domain. For example, one of them [5] uses a hand-made flowchart of possible states and lists sensors necessary for detecting transitions between them. A problem with these and similar knowledge-based solutions is that they are designed for a specific context-recognition problem. If we were to recognize different contexts using different sensors, we would have to adapt these methods, requiring either a lot of experimentation or expert knowledge.

Other proposed approaches are specialized and can only optimize a particular type of settings (e.g., sampling frequency). Such approaches then obviously cannot be re-used to optimize a different type of settings (e.g., which sensors to use). It would be useful to have a general method that can be provided with a context-recognition dataset and an indication of what parameters can be optimized, and from it generate dynamic energy-efficient solutions.

A relatively simple and general solution of the above problem would be to select the best sensor setting for each context, and then use it when such a context is detected. However, since such selection is made for each context in isolation, the effect on the whole system could be unpredictable. To illustrate: an accelerometer is very good at recognizing walking and resting, while a GPS is very good at recognizing driving. However, if we only have the accelerometer active while walking, driving might never be recognized and the sensor switch might never occur.

To truly take into account all such possible interactions, we would have to run an experiment where we classify all instances in our dataset, using a specific setting for each context and switch between them in runtime. Since there are many (context, setting) combinations that can be tried, this process might be prohibitively time consuming, possibly taking months on large datasets with many possible settings and contexts. An alternative would be to use a mathematical model to predict the effect of a (context, setting) combination, but creating an accurate model is difficult and rarely done in related work.

The problem becomes even harder if we want to adapt system settings not only to the current context, but also to the underlying data used for classification. If, for example, the current user of the system is connected to a Wi-Fi access point, their location could be inferred from the access point ID. If not, other sensors must be used for the task. This opens up an enormous search space of possible energy-efficient solutions, as the data (or the features calculated from it) can have many possible values. Traversing this search space and finding sensible values to adapt to – especially without using expert knowledge – is thus a very challenging task.

Regardless of the method used, different energy-efficient solutions will present different trade-offs between energy used and the classification quality. It is often hard to decide in advance what kind of trade is the system designer willing to make (e.g., how many percentage points of accuracy to lose in exchange for every percentage of energy saved). Different trade-offs can thus be hard to directly compare to each other. Ideally, we would like to create many such trade-offs, painting the whole landscape of possible solutions.

1.2 Approach

We propose three different methods for generating energy-efficient solutions and show how to combine them to get further energy savings. These methods optimize the energy efficiency in different ways and are designed to complement each other. They all attempt to create good solutions for an arbitrary context-recognition problem, without any expert knowledge of the domain, and two of them have no limitations on what settings are being optimized.

For the first two methods – named Setting-to-Context Assignment (SCA) and Duty-Cycle Assignment (DCA) – we assume that we have a system that works as follows: the system is in one of the predefined contexts (e.g., the user's activity), and the goal of the system is to recognize it. Each context is associated with some setting. Settings can be which sensors are in use, or what sampling frequency or duty-cycle policy is active, or which feature set is used for classification, and so on. A setting is used as long as the system believes that the user is in a context that corresponds to that setting. When a context change is detected by the system, the setting is changed accordingly. For example: if we are sitting, we might use a low sensor sampling frequency, but when walking is detected, a higher sensor sampling frequency can be turned on. Such problem formulation is very general and independent on the choice of the settings. Note that the set of contexts corresponds to the set of classes that the context recognizer classifies into.

Given the possible settings and contexts, the main issue is to efficiently find an assignment of settings to contexts that generates good trade-offs between the system energy consumption and system quality (for example: the accuracy of the sensing system). What trade-off is considered best is of course up to the system's designer, but we should strive to present them Pareto-optimal solutions to choose from.

The most straightforward way to determine the optimal assignment of settings to contexts for such a system is to simply try out every assignment. As explained in Section 1.1, this approach has two obvious problems that make it infeasible for a large number of settings or contexts. The first is the exponential number of possible assignments – if we have $|C|$ contexts and $|S|$ settings, we have $|S|^{|C|}$ different assignments. Given that a sensing system can have many sensors, each having many ways to be configured, we can expect the set of system settings to frequently be large. The second problem is the time of evaluation of a single assignment: one must create appropriate machine learning models and then run through all the data in the dataset, classifying instances while simulating different settings and switching between them accordingly.

The first problem was solved using multi-objective optimization. The two objectives of an assignment being its energy-efficiency and its classification quality. In particular, we used the NSGA-II [6] multi-objective genetic algorithm to find good trade-offs between these two objectives.

The second problem was solved by mathematically modelling a setting-to-context assignment instead of simulating it on the dataset. This significantly speeds-up the evaluation of a single assignment and subsequently the number of assignments the optimization algorithm can process. The model was made by assuming the context sequence has the Markov property (Section 2.1) and then constructing a Markov chain that models both the transitions between contexts as well as probabilities of correct classification given a context. Transition probabilities of the Markov chain are created using the transition probabilities of contexts in combination with the confusion matrices of different classifiers. While real-life datasets do not fully conform to the Markov property, the model still returns very accurate predictions.

Both the first (SCA) and the second (DCA) method are similar in spirit; the SCA method can work with any type of settings, while the DCA method is specialized in op-

timizing duty-cycling only. Duty-cycling is a process of periodically turning the sensors on (active) and off (sleeping), with the active and sleeping length being the target of optimization. This specialization allows for more powerful modeling – an entirely different mathematical model is built, still based on the Markov property – which allows for a more efficient search of this particular optimization modality. Duty-cycles were chosen for this special treatment as they can be used in virtually any context-recognition environment and often contribute the most to the energy savings.

The drawback of the previous two discussed methods is that they can only create systems that adapt to the current context (classification class of the system). The last method (CS-DT) aims to adapt the settings based on the feature values. This adaptation can be much more fine-grained and we expect it to make our system even more energy-efficient. In order to do so, we attempted to use the methodology from the field of cost-sensitive learning and tried to adapt it to the energy-efficient context recognition. We used cost-sensitive decision trees, which work in a similar way as the regular ones – the main difference being that the feature cost is explicitly used both when constructing the tree and when classifying using it. As an example, the decision on whether to expand any particular tree node is based on both the increased accuracy due to the split and the cost of the splitting feature. While the method name – CS-DT – derives from "Cost-Sensitive Decision Tree" we use this abbreviation only when referring to our adapted decision tree.

Our version of the cost-sensitive decision tree uses the energy requirements as the feature cost. It is adapted so that if some sensors are turned off, only a partial subtree is used. Afterwards, a different system setting (e.g., sensor subset) is used based on the current feature values. Such tree can be interpreted in an almost "flowchart" manner, where the system exists in a number of states with defined conditions for transition between them (e.g., if acceleration value $> x$ turn the GPS on). Further optimizations to the tree construction method were performed in order to make it more suitable to the context-recognition domain.

Finally, we attempted to adapt to both the underlying data (expressed by features) and the context at the same time by combining all three methods together. Doing so combines the advantages of each method and usually generates the most energy-efficient solutions.

In summary, our methodology uses mathematical models to provide an accurate estimate of the system behavior under different configurations, and then uses multi-objective optimization to find good energy-efficient solutions. It aims to generate the whole Pareto-optimal set of different trade-offs, giving the system designer the possibility to make an educated choice between them and choose one that most suits the application needs. Furthermore, it requires no hand-picked parameters, and works with an almost arbitrary function for evaluating either the system quality or its energy consumption.

1.3 Hypotheses and Goals

Related work shows that context recognition can be made more energy-efficient without losing much classification quality, if different system settings are assigned to different contexts. In this work we investigate the following two hypotheses:

Hypothesis 1: The performance of a setting-to-context assignment can be mathematically modeled in a way that is both fast and accurate. This model can in turn be used to automatically find good assignments that are comparable or better than those found by methods in related work.
Hypothesis 2: Energy efficiency can be further improved by not only adapting to the current context, but also to the underlying data that was used to detect it.

In addition to proving the hypotheses, the goals of the thesis are the following:

Goal 1: Develop a general method to adapt sensor settings to the current classified context. This adaptation aims to generate good trade-offs between energy consumption and classification quality.
Goal 2: Develop a method that specializes in determining duty-cycle parameters for each context. The specialization into the duty-cycling is justified by the fact that this optimization modality is the most effective in reducing the energy consumption in most context-recognition domains we tested.
Goal 3: Develop a general method to adapt sensor settings to the underlying sensor data – more specifically to the values of the features used for classification.
Goal 4: As part of these methods, create a mathematical model that can predict the quality of different assignments (their energy efficiency and classification accuracy) and find an effective way to search the space of these assignments.
Goal 5: Evaluate the methods independently, and find a way to combine them.

1.4 Scientific Contributions

Scientific contributions of this work can be summarized as follows:

Contribution 1: Creating an accurate mathematical model for assessing the quality of a setting-to-context assignment. In addition, creating an accurate mathematical model for assessing the quality of a duty-cycle-to-context assignment. Such assessments were previously done by either a simulation or very simple models.
Contribution 2: Applying the multi-objective optimization methodology to the domain of energy-efficient context recognition. We used the previously listed models as surrogate models for the optimization. This combination of fields was not yet explored. We implemented this combination in the form of two novel methods – SCA and DCA.
Contribution 3: Introducing the cost-sensitive methodology to the field of energy-efficient context recognition. We created a novel method (CS-DT) that adapts a cost-sensitive decision tree to this field.

1.5 Thesis Overview

This thesis is composed of six chapters. Chapter 2 presents the background and the related work for the thesis. It starts with a brief introduction to the research areas and techniques used in our approach, namely the Markov chains, multi-objective optimization and context recognition. Then it explores the related work on energy-efficient context recognition in detail, outlining where the contribution of our work lies compared to it.

Chapter 3 describes the design of our three proposed methods (SCA, DCA, CS-DT) in detail. It aims to explain both the intuition and the mathematical reasoning behind each of them. In the end, we propose how to combine the three methods for a joint, more energy-efficient use. Three different method combinations (SCA + DCA, CS-DT + SCA, CS-DT + SCA + DCA) are explored.

In Chapter 4 we describe all datasets on which the method was tested. Four real-life datasets used were the Commodity12 dataset, Opportunity dataset, E-Gibalec dataset and SHL dataset. These particular datasets were chosen because they were very different from one another, having different contexts and recorded using different sensors. In addition, we describe the artificially generated datasets that aimed to demonstrate some specific properties of our methods.

Our approach was tested in Chapter 5. On each of the datasets, real-life and artificial alike, we tested all three methods and their three different combinations. This allowed us to test their individual and joint contribution to the energy-efficiency of the system. Additionally, we tested the speed of our models and their accuracy. Our methods were also compared against various baseline metrics and against approaches proposed in the related work.

Finally, in Chapter 6, we conclude the thesis by making an overview of the results and presenting the discovered strength and weaknesses of our approach. Using these findings we show that these results can confirm both of our hypotheses.

Chapter 2

Background and Related Work

In this chapter we briefly present the field of Markov chains and multi-objective optimization as both are used by our methods. Understanding their basics can thus ease the understanding of the proposed approach. Additionally, we describe the field of context recognition, especially energy-efficient context recognition, in depth. Doing so presents the related work for the thesis and helps place our work in it.

2.1 Markov Chains

At its core, a Markov chain is a mathematical model that simply assumes that something can exist in a number of states, with a given probability of transition from one state to another. This probability depends only on the two states in question and not on any other factors, including the history of previous transitions. If a system can be accurately modeled in this way, it is said to have the *Markov property*.

Such a system can be visually represented with connected states as shown in Figure 2.1. In that example, if starting in state 2 we have a 50% chance to transition into state 3 in the next time step and 50% to remain in the same state. Alternatively, it can be represented using a matrix of transitions as seen in Equation 2.1. Both representations are equivalent and fully define the system.

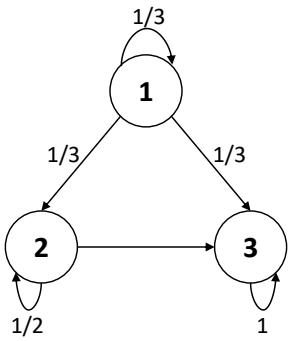

Figure 2.1: Visual representation of a Markov chain.

$$T = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

The relative simplicity of the system allows us to make powerful deductions on the system's behaviour. For example, given the system is in a state s_1, what is the probability it will be in state s_2 after n steps. More generally, given the probability distribution, x_0, of which state the system is currently in, what is the probability distribution of states after n steps. The distribution after one time step, x_1, is shown in Equation 2.2 and by induction, the distribution after n steps x_n is given by Equation 2.3.

$$x_0 T = x_1 \tag{2.2}$$

$$x_0 T^n = x_n \tag{2.3}$$

If we know the system starts in state s_1 we simply have to fix the distribution x so that this state has the starting probability of 1 and every other has the probability of 0. Such vectors are referred to as base vectors and are marked as e_i, where i marks the chosen state (Equation 2.4). The x_n in Equation 2.5 thus represents the state distribution after n steps after starting in state 1. The probability of landing in state 2 is the second component of vector x_n.

$$e_i = [0, .., 0, \underbrace{1}_{i} 0, ..., 0] \tag{2.4}$$

$$e_1 T^n = x_n \tag{2.5}$$

As a concrete example using Markov chain from Figure 2.1, if we start in state 1, after two time steps we get into the state distribution shown in Equation 2.6. From that expression it can be observed that with the likelihood of 61% we end up in state 3.

$$[1, 0, 0] \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}^2 = [0.11, 0.28, 0.61] \tag{2.6}$$

Importantly, we can also deduce the long-term properties of such a system. For example, what proportion of time does the system spend in each of the states, given enough time. To do so, let us suppose we have a distribution p for which Equation 2.7 holds. We will call the distribution p the *steady-state* of the system.

$$pT = p \tag{2.7}$$

If the current distribution of states is p, then it will remain p after any number of time steps. We can thus reason that the average proportion of time spent in any of the states is identical to the probability distribution p for that state. For example, if you have 10% to end in state s_i after each time step, then in the long run you will be in state s_i 10% of the time.

It is provable that if T^n has no 0 entries for at least one n, then the distribution p exists and any other starting distribution will converge to p as the number of time steps goes to infinity. This implies that after a certain amount of steps, the proportion of the time spent in any state corresponds to the steady-state of the Markov chain (defined by matrix T) regardless of the state in which the Markov chain starts. To determine such a steady-state, one can simply solve Equation 2.7 – the most straightforward way is to expand it into a system of linear equations (which has the same time complexity as the matrix inversion).

To complete the example, the Markov chain defined by the matrix in Equation 2.1 has the steady state of $[0, 0, 1]$ which means that after some time, the system will constantly be in state 3.

2.2 Multi-Objective Optimization

Multi-objective optimization is a field of optimization where we have more than one objective that has to be optimized simultaneously. The objectives usually (although not necessarily) conflict with each other – a classical example from this thesis would be the classification quality conflicting with the energy savings of the system. This goal can be formally expressed by finding all x that satisfy Equation 2.8, where we assume the optimization goal is the minimization of objective functions f_1, f_2,..., f_n.

$$min(f_1(x), f_2(x), \ldots, f_n(x)) \tag{2.8}$$

Given the potentially conflicting objectives, it is not immediately obvious how to compare two solutions x_1 and x_2 as one may be better in one objective and other in another. We define that x_1 dominates x_2 if both Equation 2.9 and Equation 2.10 hold.

$$\forall i : f_i(x_1) \leq f_i(x_2) \tag{2.9}$$

$$\exists i : f_i(x_1) < f_i(x_2) \tag{2.10}$$

Solutions that are not dominated by any other solution are called Pareto optimal and finding them is the goal of a multi-objective algorithm. Such solutions visualized in objective space (Figure 2.2) are called *Pareto front*. They are often visually depicted as connected to their neighbors. The connections in this figure were made in a step-wise fashion as is conventional in the field. This way, every point that would be located to the right and above of the Pareto front, would be dominated by at least one of the points on the front.

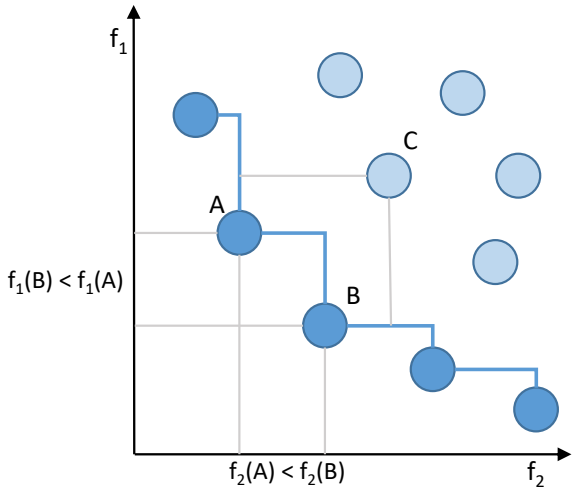

Figure 2.2: Visual representation of a Pareto front. For each of the solutions (A, B, C, etc.) the values of two objectives are shown, one on each axis. Given the task is the one of minimization, the lower and the more left the points stand, the better. Point A, point B and all the dark blue points are parts on the Pareto front. Point C is dominated by the Point A (as well as Point B) and is thus not on the Pareto front. The same goes for all light blue circles. Do note that comparing any two points on the front, each will be better in one objective than another. We thus cannot say that one is better than another.

2.2.1 Multi-Objective Optimization Basic Properties

Given a small set of possible solutions, evaluating them all and constructing the corresponding Pareto front can be considered trivial using the definitions from the previous section. However, the set of possible solutions is usually large and often infinite. One must thus resort to evaluating only a manageable subset of them. A possible strategy for the subset selection is given in Section 2.2.2.

An algorithm that only evaluates a subset of solutions cannot in general guarantee to find all (or any) of the points on the Pareto front. It can, however, find a non-dominated set of solutions out of all evaluated. We will call this set the Pareto front approximation.

Two different algorithms – or if the algorithm is stochastic, two runs of the same algorithm – may find different Pareto front approximations. One such case is depicted in Figure 2.3. Sometimes, it is clear that one set of solutions is better than another, as is the case with the sets A and C. Sometimes, the superiority of one set is less clear, as is the case with the sets A and B. One can observe from the figure that the set A is to be preferred if we are looking for solutions with a low value of f_2 and vice versa if we want the solutions with a low value of f_1.

An alternative to the "naked eye" observation is to evaluate the quality of the solution set using a mathematical indicator. The indicator chosen for use in this thesis is the hyper-volume indicator [7], which measures the area from the Pareto front approximation to some reference point that is placed in a position dominated by every other point. The greater the area, the better (according to the hyper-volume indicator) is the set of solutions that compose it. The hyper-volume measures two important properties of the solutions set – the quality of individual solutions and their spread.

In this thesis different Pareto front approximations are sometimes generated by not only different algorithms, but also different solution types – one approach may optimize the sampling frequencies, another which sensors to use. Nonetheless, they can still be compared using the same objective functions (the classification quality and energy consumption).

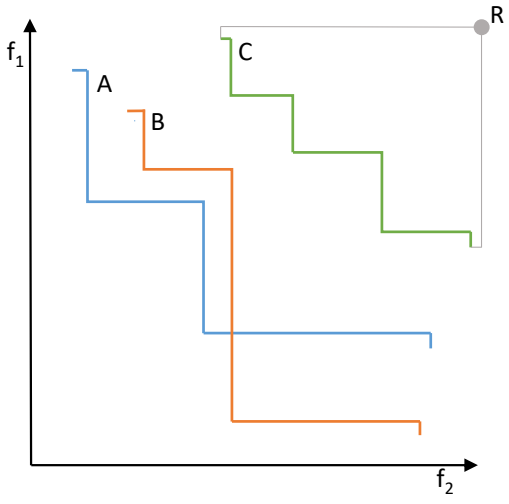

Figure 2.3: Visual representation of different Pareto front approximations. Only the connecting lines are shown, with the individual solutions implied to stand in each of the corners that point to the coordinate system origin and at line extremes. Each front is labeled by a letter (A–C). The reference point used for hyper-volume calculation is shown as R. The hyper-volume of the Pareto front approximation C is calculated as the area of the shape bounded by front C and gray lines.

Sometimes even the evaluation of a small solution subset is too computationally ex-

pensive. In such cases it is possible to create approximations of the f_1, f_2,...,f_n functions and use those instead in order to speed up the search. Such approximations are called surrogate models and will be used extensively in this thesis – where instead of determining the energy consumption of a system using a simulation, we use a mathematical model instead.

2.2.2 Solving Multi-Objective Optimization Problems

The most popular approach used for searching the solution space for better and better Pareto front approximations are genetic algorithms. These algorithms got their names by borrowing some ideas from the real-life evolution. The outline goes as follows: it starts with random solutions, they are evaluated and the best ones "survive" while the worse ones are discarded. The next generation is created by a variation and combination of the survivors and the process is repeated.

In order to evaluate and modify a solution, it must have some pre-defined structure. As an example, an energy-efficient solution consisting of instruction: "use sampling frequency of 50 when context 1 is detected and the sampling frequency of 100 when context 2 is detected" could be formalized as vector (50, 100).

In this thesis we use a well known genetic algorithm called NSGA-II [6]. One of its distinguishing features is how it decides which solutions are selected to become part of the next generation. It uses two criteria, the first is called non-dominant sorting that ranks all solutions in the following manner: solutions that are not dominated by any other have the rank of one. Those that are only dominated by solutions of rank one, have rank two, and so on. The second criterion, crowding distance, rewards the solutions that are different from others, encouraging a wide spread of solutions on the resulting Pareto front approximation.

2.3 Context Recognition

We will define context recognition[1] in a very general way:

Definition 2.1. Context recognition is a term encompassing a wide array of tasks where (usually wearable) sensors are used for monitoring a user and trying to infer something about their state, activity or surroundings (*context*). This is done by using machine learning to analyze time-series of sensor readings.

It is a mature research area [8] that is still actively researched and is already enjoying a widespread use in practice. The most typical and well researched task is the *activity recognition* where the goal is to determine the user's current activity [9] – e.g., *walking*, *running*, *resting* etc. This can be used in a variety of ways ranging from monitoring ones lifestyle [2][3], improving ones health [10][11] or simply tracking ones progress in a sport of interest [12].

Another common task is determining the user's location. This information can be directly useful (e.g., in a navigation application), but it can also be used for location-based recommendations [13], adapting phone's settings [14], getting and contributing to the traffic congestion status [15], annotating user's pictures [16], etc.

The many other applications include using sound sensors to determine if the user is having a conversation or the type of ambient they are currently in (e.g., in a meeting, walking on street) or using a wristband to monitor user's blood pressure [17].

Various low-level context information can be combined to determine more sophisticated features of the user. For example, one could determine their mental state (e.g., stressed

[1] Also referred to as "Context Awareness" in other literature.

[18], bored, erratic), or their "high-level" activities (e.g., working, traveling, shopping, resting at home) [11].

2.3.1 Methodology

The context recognition is typically performed in a *supervised learning* setting. The task of supervised machine learning is to learn a function from the space of feature vectors X into the space of classes Y. To do so, it uses a dataset of training examples, each being a $(x \in X, y \in Y)$ pair. This pair is often called *instance*, while the y is referred to as class, label or in the case of context recognition – context. When Y contains a finite set of discrete values, the task is called *classification*, and when Y contains continuous numeric values, the task is called *regression*.

The first step of this process is the data collection. This is most frequently done using body-worn sensors. The most convenient devices for the task are currently the smartphones. They are equipped with a variety of sensors and a processing unit that can analyze the incoming data, and most importantly are universally carried. Other devices, such as smart wristbands and watches, are also gaining on popularity [1] and are useful for similar reasons. A variety of other wearable devices exist, ranging from pedometers worn in shoes to chest-worn heart-rate monitors. In the future, other types of wearable sensors may gain on popularity (such as smart eyeglasses or smart clothing). External sensors can be used just as well, camera and motion sensors being the prominent examples.

Any such sensor typically produces a continuous stream of data. The stream is then split into windows of a fixed length – each window becoming one training feature vector. To complete the training instance, the correct label (usually hand collected, during the data collection) is appended to each data window. Windows can be partially overlapping, although in the datasets presented in this thesis they are all strictly sequential. The window length is dependant on how much data is needed in each window for a meaningful feature extraction and on the required frequency of reported context.

In the next step, features are calculated from each window. Typical features may be the mean and variance of the containing data, its maximum, minimum or periodicity, but they may be substantially more complex. If the data is composed of several different sensors (each contributing one data stream) they are usually each processed individually, but may be combined in some circumstances.

Many algorithms for inferring the $X \mapsto Y$ function exist. For processing the datasets presented in this thesis we primarily used the Decision Tree classifier [19], Random Forest [20], k-Nearest Neighbors [21] and XGBoost [22]. Recently, deep learning approaches [23] are gaining prominence and their performance closely matches the performance of the classical machine learning algorithms on context-recognition tasks.

The learned function is called a *classifier* in the discrete case and a *regressor* in the continuous case. Alternatively, the name *model* is used for both cases. To use it in practice, we continue collecting the data, windowing it and transforming it into features, but omitting the collection of labels. The labels are automatically inferred using the learned model.

Finally, we need to evaluate the quality of the trained model. The most simple approach for the task is to split the collected and labeled set into two parts, training and testing. The first is used to train the model, and then the model is used to predict the labels of the second. Given that the correct labels for the testing set are known in advance, they can be compared to the predictions. Accuracy is the most commonly used metric, and it simply reports the percentage of times the predicted and actual labels matched. A slightly more sophisticated metric is the *confusion matrix* which counts the number of times the context c_i was correct while the context c_j was predicted, for every i and j.

2.4 Energy-Efficient Context Recognition

While continuous sensing with potentially many sensors can be highly beneficial for different context-recognition tasks, it also places a strain on the sensing device's energy consumption. This can quickly drain the battery of the device, making the context-recognition application undesirable in practice. While this issue is often ignored, especially in academic papers, it it crucial in practice – Murnane et al. [24] even state it is one of the biggest contributors to the abandonment of medical monitoring applications.

This section explores the current state of the art on the topic and discusses a variety of different approaches to the problem. First (Section 2.4.1), we present different optimization modalities, i.e., different parameters that can be changed in order to make the context-recognition methods more energy efficient. Second (Section 2.4.2), we discuss mechanisms for dynamically adapting these parameters during a system's runtime. Last (Section 2.4.3), we list what we consider as weaknesses of the current field of research and explain how our proposed approach tries to amend them.

2.4.1 Optimization Modalities

Different parts of a context-recognition system can be adapted in order to increase its energy efficiency. A quick survey can be found in the work of Rault et al. [25]. For convenience we will call all such system's parts that are being adapted *system's settings*.

The first category of possible system's settings are the sensor settings. The most frequent is changing the sampling frequency of the sensors. This not only reduces the energy consumption of the sensor itself, but it also indirectly reduces the cost of sending and processing the data. Another popular approach is periodically switching sensors on and off – *duty-cycling*. One can also choose not to use all sensors at their disposal or replace power hungry sensors with others that may also be suitable for the task. Finally, one can use specialized hardware that is made for a specific task, or simply better for energy efficiency in general (e.g., bigger battery).

The second category of settings consists of approaches that reduce the amount of processing needed. This can be achieved by not processing all the data, or using only a part of the processing pipeline if the data is deemed uninteresting. Another idea would be to reduce the complexity of the computation itself, by either using simple features or simple classification models. Finally, one can consider sending the data to another device, where it is processed remotely.

2.4.1.1 Sampling Frequency

Ideally, the sampling frequency should be high enough to capture the relevant aspect of the data, but not any higher. It was shown in several domains [26][27] that the energy required increases almost linearly with the increasing frequency, but the classification accuracy usually increases rapidly up to a certain point, and then reaches a plateau and stops improving.

This "ideal" point can be determined in several different ways. It has been empirically established that the frequency of 20 Hz is required and sufficient for many context-recognition tasks [28][29] and can be used as a first approximation when choosing a suitable frequency. This is, however, not always the case – for example, Khan et al. [30] analyzed 7 activity recognition datasets and for each determined a different required sampling frequency, ranging from 12 Hz to 63 Hz. Walton et al. [31] did activity recognition on sheep and required the sampling frequency of 32 Hz for the task. The problem must thus be solved separately for each domain.

One solution is collecting high-frequency data and then iteratively down-sampling it, testing the classification accuracy at each step [27][32]. In the example provided by Kraus et al. [26], the classification accuracy for activity recognition using a wristband was 70% at 1 Hz, 80% at 5Hz and 95% when using 10 Hz or higher. Such analysis can then be used to justify the selection of the 10 Hz sampling frequency.

Another way for determining a good sampling frequency is to use the well-known Shannon-Nyquist theorem [33], which states that the sampling frequency should be at least double the frequency of motion the sensor tries to capture. This justifies the 20 Hz estimate given earlier, as most human motion exhibits a frequency of less than 10 Hz. When trying to capture non-human movement – such as vehicle vibration in an attempt to recognize the mode of transportation [34] – higher sampling frequencies might be required.

A statistical approach that is not dependent on either the context-recognition task or the classification model is presented in the work of Khan et al. [30]. It iteratively down-samples the data and compares it to the original data using Kolmogorov-Smirnov statistical test. This tests to which frequency can one down-sample before losing information present in the data.

It is also of note that different features may be more or less dependant on the sampling frequency. For example, in the work of Qi et al. [35], they analyzed the performance of several features and selected only those that perform well using a low sampling frequency. A similar argument can be made for different classifiers – in the work of Zheng et al. [36] they adapted a SVM classifier in order for it to work well with low-frequency data.

2.4.1.2 Duty-Cycling

The majority of contexts that we may be interested in change only infrequently. For example, user activities such as walking or resting usually persist for a few minutes before changing. If we are interested if the user is at home or at work, it may be hours before a change occurs.

This naturally leads to the idea of determining the context and then turning sensors and subsequent processing off for a given time before turning them on again. During the *off* (also called *sleeping*) cycle, an application usually assumes that the context has not changed – negatively affecting both the classification accuracy and latency of context-detection, if it did. If the context did change in the sleeping phase, one can also try to determine the most probable time period of when it happened. In the work of Abdesslem et al. [37], they did so by modeling context switching with a Hidden Markov Model.

Assuming that turning sensors on has no time or energy cost attached to it, it is easy to deduce that the energy consumption is inversely proportional to the length of the *sleep* time. E.g., a system that sleeps $\frac{1}{10}$ of the time consumes roughly 10 times as little energy (both due to reduced sensor and processing cost). A precise mathematical model of how the sleeping time length affects the accuracy of the system is given in Section 3.2.1. For a first "rule of thumb", however, one can assume that the accuracy falls linearly with the length of the sleeping phase (especially for short phases). If turning the sensor on is not free (GPS is a notable example with a long set-up time), the time and energy cost of turning the sensors on must be accounted for. In this case longer sleep cycles may have a better ratio between the performance and energy-consumption. As a concrete example of the benefits of duty-cyling, Bosch et al. [38] were measuring the level of a person's activity, and using them managed to preserve 60% of battery life in exchange for 5 percentage points accuracy loss.

The main intuition for setting appropriate duty-cycle settings goes as follows: in domain with slower changing contexts, long duty-cycles can be used, while in domains with fast changing contexts short duty-cycles are required. Note that, if only one of the contexts of

interest is short, one might still be inclined to use short duty cycles in order not to miss that context entirely. A good example of the latter is fall detection, where falling is such a short "activity" that long sleep cycles should be avoided. The effect of duty-cycling can be experimentally determined by discarding the appropriate parts of the data of a recorded dataset. They can also be modeled directly as described in Section 3.2.1.

Note that in some cases the duty-cycling is employed only on a sensor subset, e.g., in the work of Cvetković et al. [39] they used it only for the sound recording (which is energy hungry) but not on the other sensors. Additionally note that instead of constantly using only a fixed duty-cycle length, many approaches [40][41] opt for changing the duty-cycle lengths based on a variety of factors (the last classified context, the frequency of previous context transitions, etc.) as explained in Section 2.4.2.

2.4.1.3 Sensor Subset Selection

Another common approach for decreasing the energy consumption is to change the sensors used for the recognition task. It is frequently the case that different sensors could be used for the same context-recognition task. For example, location can be accurately determined by using the GPS, but it can also be determined (perhaps with a slightly lesser accuracy) using the Wi-Fi signal [42] – consuming four times less energy [43][34]. Alternatively, location could be determined by location fingerprinting using ambient sound, light and color [44] or by getting the location information from another device that may be connected to it [45]. The choice of sensors can thus affect both the energy consumption and classification accuracy.

Many context-recognition tasks operate using multiple different sensors at the same time [46][47]. In some cases the information of some sensors is redundant and the system can be just as accurate if using only a subset of them. This was done in the work of Gao et al. [48] for a body-sensor network, where they used convex optimization to determine the ideal subset of connected acceleration sensors in order to reduce the energy cost of data transmission. Similarly in the work by Noshadiet et al. [49], multiple pressure sensors were placed into a single shoe. By trying different sensor subsets and testing if it is possible to predict the output of the remaining sensors, a sensible subset was selected. A somewhat unique approach was used by Zappi et al. [50] who tried to find an energy-efficient sensor subset that was robust to the failure of any one individual sensor. Note that in the cases where each sensor comes with their own battery, reducing their number does not necessarily result in a more energy-efficient system.

In general there is no mathematical (domain-knowledge-free) method to determine which sensors can be replaced with which, without first recording a dataset using that sensor in addition to analyzing its energy requirements. This approach to energy optimization is thus most reliant on extensive experimentation. It also requires the original dataset to be captured with many sensors in order to make such experimentation even possible.

Given that in many context-recognition problems the available sensors are often predetermined or limited in number, this optimization modality can be sometimes hard to use statically. However, it yields greater benefits when the selected sensor subset is dynamic as described in Section 2.4.2, as it is easy to see that different sensor subsets may be useful in different circumstances.

2.4.1.4 Reducing Processing Time

Many devices that collect data are also able to process it on the spot. This can be very convenient for many context-recognition tasks, but it also places an additional strain on the battery life of the device. It is thus beneficial to limit the processing time required by

the context-recognition methods. While the data collection is usually more power hungry than its processing [51][52], the ratio between the two can change depending on the device used.

The first part of the data processing is usually the feature extraction process. It can be reduced by either choosing simpler features or by calculating fewer of them. It has been shown [32][53], for example, that frequency-domain features (using Fourier transform [54] as the first step) are usually more computational demanding than time-domain features. Using fewer features can be achieved by employing one of the standard feature selection methods [55]. Cilla et al. [56] used multi-objective optimization, where the first criterion is the number of features, while the second is classification accuracy. Their method is one of the few in this survey to present different trade-offs (in the form of a Pareto front approximation) to the system designer.

Different classifiers can also have a great effect on the processing power required. Some classifiers (e.g., Naive Bayes) are much simpler than others (e.g., Deep learning). In addition, classifiers can sometimes be adapted in order to increase their energy-efficiency – in the work of Anguita et al. [57] they accomplished this task by changing the SVM classifier so that it used fixed point arithmetic, reducing its energy consumption by 25%.

Increasing the window size (and potentially decreasing the sampling rate) [27], can also serve in order to reduce the number of times the features and subsequent classifications need to be calculated. Note that features are not always cheaper to calculate if split into more windows as larger windows contain more data which can be harder to process – especially if the features have a time complexity $> O(n)$.

If the processing pipeline is structured in a hierarchical manner [58][59], one can also partially process the data, and then discard it if it is deemed "uninteresting". For example in the work of Srinivasan et al. [59] the system first checks a small window of acceleration data. If the values are close to zero, the user is deemed "idle" and other more computationally complex steps are not performed. A hierarchical system that tries to solve different context-recognition tasks at the same time opens additional doors to optimization [60][61][62]. If the "vehicle classifier" decides that the user is in a car, then perhaps the "high-level activity classifier" can infer that the user is traveling, without the use of extra processing time or additional sensors. For example, in the work of Nath et al. [60] they achieved this by creating logical rules about the correlation between different contexts.

Alternatives to processing the data on the device itself are storing it internally or sending it to another device in order for it to be processed remotely. The former approach can be unpractical in many domains (as the storage space is limited and it places a burden on the user to move the data), while the latter introduces a new energy cost – the cost of sending the data (thus needing Wi-Fi, Bluetooth or similar connection) [62]. Nonetheless it is used in many domains. Some papers [63] explicitly model both the cost of sending the data as well as the cost of processing it, and dynamically choose which is more efficient. Some [64][65] process easier tasks locally, while letting harder ones to be done remotely.

2.4.1.5 Specialized Hardware

The details of creating hardware with a longer battery lifetime (either in general or for a specific context-recognition task) are beyond the scope of this thesis. We list this category simply to acknowledge that this is a common approach for commercial products that have a fixed software running.

A special and note-worthy case of hardware solutions is having an extra low-battery-consumption processor that specializes in a simple task and only wakes the main processor when needed [66][67]. Prominent example of this method are voice-activated devices/applications (e.g., Amazon Alexa [68], OK Google [69]) or many context-recognition

systems that are already built into the smartphones by their manufacturers.

2.4.2 Dynamically Adapting Sensor Settings

While discussing the optimization modalities in Section 2.4.1, we assumed that the described settings constantly remain the same when the context-recognition system is working. This may of course be sub-optimal. We might want to be using the GPS when the user is traveling, but not using it when he is indoors. We may want to have a high sampling frequency when the user is running, but a very low sampling frequency when they are resting – just high enough to detect the start of their movement. One can easily imagine many other scenarios when this is the case.

Adapting the system settings depending on the circumstances is the core of this research area, as there are many different factors the settings can adapt to, and many methods for deciding on how to adapt.

In some domains expert knowledge suffices to design a useful sensor adaptation scheme. An example can be found in the work of Wang et al. [5], where they used an xml scheme of instructions to determine when to use a specific sensor. Such a scheme can be very flexible and able to adapt to different types of factors (properties of the data, feature values, classified context, etc.). Many domain-specific approaches [70] would fall into this category. The obvious downside of such an approach is that it cannot be reused on another domain, and applying it on each new domain often requires a lot of expert knowledge and tedious experimentation.

Other approaches are more structured, and usually try to optimize based on one factor in a systematic way. In most cases, however, they are still partially dependant on expert knowledge. In the following subsections we provide a quick survey of these approaches, split based on whatever they adapt to the current context or to the current data.

2.4.2.1 Based on Context

These approaches are based on the last recognized context. Each context has a setting assigned and that setting is used whenever the corresponding context is recognized. The previously listed case with running and resting activities is a good example of this approach.

Deciding which settings to use for which context, however, is non-trivial. Even if both the set of contexts and the set of settings are fixed, the problem exhibits a combinatorial explosion of possibilities. In fact, it can be proven (Section 3.1.2.1) that finding the optimal assignment is at least NP hard. Approximate methods are thus required.

A simple example of such approximation is to decide for each activity in isolation which setting is best for that activity [32][71]. This is done by testing each activity against each possible setting, and determining, for example, that sampling frequency does not affect the recognition of *resting*, but it needs to be high to detect *running*. In this example, a high sampling frequency would be assigned to *running* and low to *resting*. This greatly reduces the complexity of the search, but may be a poor approximation for the performance of the system. What often happens in practice, but is not modeled by this method, is that a misclassification causes the switch to setting inappropriate for the current context, which in turn leads to further misclassifications.

A way to partially mitigate the described problem is to modify the system in a way that whenever a context change is detected, a "maximum" (all sensors, max frequency, etc.) setting is used to determine the correct context [53]. After that, an appropriate setting for that context is used as usual. This helps to mitigate misclassification chains, but in turn has the downside of using the "maximum" setting often, especially in a domain where either frequent misclassifications or frequent context changes are expected.

Another variation is to base the sensor settings on the context that system believes would happen next, instead of the current one. In the work of Gordon et al. [72], the next most probable contexts were predicted using a Markov chain model, and then a sensor subset that can best distinguish between them was used. The best subset is again determined in isolation and thus has all the above-described problems. Nonetheless, they reduced the energy consumption when working with the publicly available Opportunity dataset [73] by 90% in exchange for 3 percentage points of accuracy.

An altogether different way of using context for adapting the system settings is tracking the context change. If the context is "stable" and has remained unchanged for a while, sensing capabilities could be lowered. This approach is particularly useful in combination with duty-cycling. When the context is rapidly changing, a short duty-cycle is used; when the context is stable, an increasingly longer duty-cycle is used instead [40][41][74][75]. In the work of Au et al. [75] the duty-cycle was lengthened linearly while the context is stable, but its length was divided by a constant otherwise. Rachuri et al. proposed a similar scheme, but they were multiplying the duty-cycle length in the case of the stable context. Sarker et al. [74] also proposed multiplying, but with a relatively complex formula. In all cases the minimum and maximum duty-cycle length was bounded.

A combination of this and the previous approach can be found in the work of Lee et al. [27], where the duty-cycling is employed only if certain stationary activities are detected, while the window size is increasingly longer and the sampling frequency is decreasing if the activity is stable.

2.4.2.2 Based on Data

Context-based approaches are convenient as the number of possible contexts is usually limited. On the other hand, it restricts the adaptation to the context itself. Imagine having an application that recognizes the words people are speaking around it and wanting to avoid using a complex processing pipeline when the surroundings are totally silent. Silence is an attribute of the data and not a recognized class of the system, hence no approach in the previous section could model this behaviour.

This example in fact represents the most frequent use of this adaptation method. System sensing capabilities lower when the incoming data is in a way "uninteresting". Typical indicators are sound sensors hearing silence [76], movement sensors detecting no movement [77] or location sensor detecting no location change [42]. A different type of "uninteresting" is presented in the work of Lu et al. [76], where the data is discarded if it is too similar to the previously acquired (implying that the context has not changed). The last listed approach in fact used several different techniques (different for each sensor) in a complex pipeline. To make the system more robust, the setting is sometimes [77] only changed if the "uninteresting" data persists for a longer time, or alternatively the systems settings are gradually lowered [76] in that case. Such systems can be simple to implement and usually work well in practice, but are inflexible in the sense they can only adapt to some predefined notion of "uninteresting" data.

Another similar category of approaches uses a specific data pattern as a trigger. For example, one may turn the GPS on only when the accelerometer detects movement [78]. Similarly, Park et al. [79] used gyroscope in their work only when accelerometer detects data that could be a part of a hand gesture. In the work of Rahmati et al. [80] a combination of acceleration data, time and available cellular towers was fed to a mathematical model to determine the probability of Wi-Fi being available, without turning on its interface.

The RAPS [81] system decides which location sensors to use based on which were needed the last time the user was around this location at a similar point of day. Doing so,

they managed to reduce the battery consumption by a factor of four, while retaining their localisation accuracy. This system is interesting as it is one of the few that can gradually adapt to the user habits while the application is in use.

Approaches so far presented in this section rely heavily on expert knowledge. Adapting to the data using a domain-knowledge-free method is done less frequently. The most popular way of doing so is using a Markov-decision process [82][83][84] that models which setting to use given the current state. States are usually represented by discretizing different features, while possible actions are defined as the possible settings to use for the next instance. In the work of Au et al. [83] the states were defined with the help of expert knowledge, while in the work of Cheng et al. [82] all features and the last predicted context were taken as a possible state. This last approach thus implicitly combines the adaptation to data and context.

An alternative is to use cost-sensitive learning presented in the next subsection, which is used for similar tasks in other domains, but has previously not been adapted for context-recognition.

2.4.2.3 Cost-Sensitive Learning

Cost-sensitive learning is in fact a data-based approach to adapting the sensor settings. However, given it is a wide research field on its own and given it is used heavily in one of our proposed methods, we dedicated this section to it.

The field of cost-sensitive learning is interested in developing machine-learning methods that acknowledge that different features have different costs. To illustrate on a medical diagnosis problem, we can imagine that performing many tests on a patient increases the odds of correct diagnosis. On the hand, these tests have a cost attached to them (time, comfort, money), and they should be avoided if possible. This creates a trade-off between the diagnosis cost and the quality of the diagnosis. Similar trade-offs appear in energy-efficient context recognition and this methodology can thus be used for the task.

While the research area of cost-sensitive learning is also interested in different misclassification costs (e.g., diagnosing a healthy patient as ill might be less damaging than vice-versa) we will focus primarily on the feature-cost aspect of the field and assume that all misclassifications are equally undesirable.

Many different regular machine learning methods have already been adapted to the cost-sensitive approach (e.g., Naive Bayes [85], Neural Networks [86]). The most classical example of the cost-sensitive learning algorithm, however, is the cost-sensitive decision tree. There are many cost-sensitive decision trees described in literature [87]; the one we used as the base of our method in Section 3.3 is presented in the work of Ling et al. [88]. To explain it briefly: at each tree node – instead of checking each feature's information gain or a similar metric – we compare the expected cost of misclassification if this node becomes a leaf, with the expected cost of the feature and misclassification if the attribute is used to further divide the instances. The feature that reduces the expected cost the most, if any, then expands this node.

Using any of these methods in the context-recognition setting requires some adaptations. To the best of our knowledge, they have not yet been made for this domain.

2.4.3 Comparing Our Methods to the Related Work

Previous sections presented the landscape of the related work, which makes it easier to both identify weaknesses in the presented state-of-the-art and to compare our methods (outlined in Section 1.2 and described in Section 3) with it.

Most of the presented methods heavily depend on either the domain (e.g., optimization of localization) or the type of settings (e.g., optimization of duty-cycles). Our method, in contrast, is independent of both: it uses no expert knowledge, and can be used with most context-recognition tasks.

The majority of the presented methods also offer only one solution instead of returning a spread of different trade-offs. While some methods are parameterized and can return different solutions depending on the chosen parameters, it is seldom specified how to search the space of possible values. Our methods return a Pareto front approximation of possible solutions which can be greatly beneficial to the system's designer. To do so, it uses multi-objective optimization, a field that is rarely used in context-recognition (we only found one other work [56], and its approach is not directly comparable to ours).

Another potential area for improvement of the current state-of-the-art is the modeling of the solutions. The existing approaches rely either on very simple models (which are inaccurate) or on simulating those solutions on the recorded dataset (which is slow). To remand this issue we developed accurate mathematical models for context-to-setting assignments. This allows us to quickly evaluate a wide array of different solutions and speeds up the search for them.

In addition to our methods' generality and speed, we will show (Section 5.3) that we can match and surpass some of the presented state-of-the-art methods. Ideally, we would want to compare our methods to the best methods from the related work, but finding the *best* proved to be difficult. Different methods optimize different settings in different domains and as such they very rarely provide a comparison of one to another. Instead, we selected the methods from related work that are most directly comparable to ours (similar level of generality, settings, etc.) and were presented with enough information to be fully reproduced.

The SCA method assigns a sensor setting to each context. We started by comparing it to using a single setting in all contexts [30]. Then we showed that our accuracy modeling is more accurate than the one presented in the work of Yan et al. [53] or Vo et al. [32], and that this allows us to find better solutions. Lastly, we compared these solutions to the solutions achieved when assigning the setting depending on the predicted subsequent context (instead of the current one) [72]. The last was chosen because it was comparable to our SCA method in both the generality and the search space of possible solutions.

The DCA method assigns a duty-cycle setting to each context. As a baseline, it was compared against using a fixed duty cycle for all contexts. Then we tested it against an approach that changes duty-cycle length depending on the frequency of context change [75]. These two methods adapt to different factors (current context vs frequency of context change), but we deemed them similar enough to make the comparison. This also lets us observe the advantages and disadvantages of adapting to these two different factors.

The CS-DT method uses the cost-sensitive methodology to reduce the energy consumption. We compared this method to one of the methods that use Markov decision process, as they both adapt to the sensor data. In particular, we chose the work of Cheng et al. [82] as it was the newest and because it claims to be better than other similar methods.

Finally, we compared our solutions against a hand-crafted solution generated with expert knowledge to show that despite the generality of the method it can still compete against very specialized solutions. While such a comparison is not conclusive and is much reliant on the "expert", it serves to demonstrate how hard the problem is to solve with expert knowledge alone.

It is of note that many of the presented methods do not directly compete against ours, but instead complement each other. For example, using an energy-efficient classifier or data transmission method can be considered a system's setting that our methods can optimize.

Chapter 3

Methodology

In this section we present three different methods for generating energy-efficient solutions to a context-recognition problem. These methods optimize the energy efficiency in different ways and are designed to complement each other.

The first method, *Setting-to-Context Assignment* (SCA), is described in Section 3.1. It tries to assign one of the predefined settings to each context, and use it when that context is recognized by the system. Intuitively, a system might need different sensors when the user is driving compared to when he is resting at home.

The second method, *Duty-Cycle Assignment* (DCA), is described in Section 3.2 and it creates the same type of assignments, but only for the duty-cycle settings (how long does the system sleep after each classification). This specialization was made in order to refine these settings, given that they can be used in virtually any domain and usually contribute more to the energy savings than any other setting.

The last method, *Cost-Sensitive Decision-Trees* (CS-DT) is described in Section 3.3. It approaches the problem from a different angle – by employing cost-sensitive learning. While the previous two methods adapt to the recognized context, this one adapts to the feature values, which can be considered a more fine-grained adaptation. E.g., even if the context is determined as "traveling", different sensors or duty-cycle policies can be active depending on the current speed of the user.

To simultaneously benefit from all three methods we combined them in different ways (Section 3.4): the first two methods can be run one after another, generating a combined, more energy-efficient solution. The third method can be modified so it adapts not only to the raw data but to contexts as well – using the methodology from the first two. Lastly, we propose how to use them all in unison.

3.1 Setting-to-Context Assignment (SCA)

Definition 3.1. Setting is a parameter of the context-recognition system that can be modified in order to change its energy consumption and/or its classification quality. For example, settings can be which sensors are in use, or what duty-cycle policy is active, or which feature set is used, etc.

Suppose we have a sensing system that works as follows: the user is in one of the predefined contexts $c \in C$ (e.g. the user's activity), and the goal of the system is to recognize it. Each context is associated with some setting $s \in S$. A setting is used as long as the system believes that the user is in a context that corresponds to that setting. When a context change is detected by the system, the setting is changed accordingly. For

example: if we are sitting, we might use a low sensor sampling frequency, but when walking is detected, a higher sensor sampling frequency can be turned on.

Definition 3.2. Setting-to-context assignment is a mapping that assigns one setting to each of the possible contexts ($a : C \mapsto S$). The same name is also used for the method that generates such assignments.

Given possible settings and contexts, the main task is to efficiently find a setting-to-context assignment that generates good trade-offs between the system energy consumption and its classification quality (for example: the accuracy of the sensing system). Note that these two objectives are usually inversely related: increasing the energy-efficiency of the system usually reduces its classification quality and vice versa. What trade-off between the objectives is considered best is of course up to the system's designer, but we should strive to present him Pareto-optimal solutions to choose from.

The most straightforward way to determine the optimal assignment of settings to contexts for such a system is to simply try out every possible assignment. This approach has two obvious problems that make it infeasible for a large number of settings or contexts. First is the exponential number of assignments: if we have $|C|$ contexts and $|S|$ settings, we have $|S|^{|C|}$ different assignments. Given that a sensing system can have many sensors, each having many ways to be configured, we can expect the set of system settings to frequently be large. As an example from the Commodity12 dataset (Section 4.4): having 10 sensors that can be either on or off gives us $|S| = 2^{10} = 1024$ different settings. With 7 different contexts the total number of assignments becomes $1024^7 \approx 10^{21}$. The second problem is the time of evaluation of one assignment: one must create appropriate machine learning models and then run through all the data in the dataset, classifying instances while simulating different settings and switching between them accordingly – a process that can be prohibitively slow if we plan on evaluating a large number of assignments. In the Commodity12 example the evaluation of one assignment lasts a couple of seconds (model training time not included) on standard hardware, which still results in the optimization process that can last several days even if only small subsets of all possible assignments are tried.

A common solution is to use expert knowledge to determine sensible assignments and test only those. However, such knowledge is often either not available or not good enough to isolate all good potential assignments, which calls for an alternative, automatic method for solving the task. In Section 3.1.1 we show how to reduce the evaluation time of a single assignment and then in Section 3.1.2 how to select a subset of assignments to evaluate.

3.1.1 Modeling Assignment Performance

To evaluate as many assignments of settings to contexts as possible, we strive to reduce the time of each evaluation by mathematically estimating the energy consumption of the system and its classification quality and avoid actual experiments. One experiment is considered to be classifying all instances in the dataset using appropriate settings, and creating new classification models if so needed for the task. The following two sections describe a naive and a more accurate evaluation model.

3.1.1.1 Simple Evaluation Model

We start with a simple approach, similar to related work [53][32] in the sense that it models every context in isolation. To use it, we first determine the accuracy of the system using each setting, by experimental evaluation. In these experiments the same setting is kept throughout, without switching depending on the context. This requires $|S|$ experiments, exponentially fewer than $|S|^{|C|}$. Then, to evaluate the performance of the system that uses

a specific assignment a, the accuracies of each context c using the corresponding setting $a(c)$ are summed up, weighted by the proportion of that context in the dataset.

$$Accuracy(a) = \sum_{c \in C} p(c) Acc(c|a(c)) \tag{3.1}$$

$a(c)$ – the setting assigned to context c.
$p(c)$ – the probability of context c, estimated by its proportion in the dataset.
$Acc(c|s)$ – the accuracy for recognizing context c, given the setting s is active.

Let us take the example of having two contexts c_1, c_2 and two corresponding settings s_1, s_2. Additionally we suppose that using s_1, we can predict c_1 with the accuracy of 60%, and using s_2, we predict c_2 with the accuracy of 100%, and that both contexts appear an equal amount of time. In this case we can make a prediction that this assignment will produce 80% accurate results.

We can make a similar prediction about the energy consumption, given the information about the energy requirement of each setting.

This evaluation, while easy to compute, models only how the system behaves when the correct setting is active. However, it does not model what happens when a classifier makes a classification mistake and the system subsequently switches to a setting inappropriate for the current context. Additionally, it does not directly take into account what context is likely to appear next and how good is the current setting at classifying it. The naive evaluation is presented both because it is intuitive and as a benchmark to show how modeling the system as a whole reduces its shortcomings.

3.1.1.2 Markov-Chain Evaluation Model

To improve upon the naive model, we propose [52][89][90] a Markov-chain model for making the performance trade-off estimation. A Markov chain is a stochastic process that moves through predefined states, with the probability of every transition being dependent only on the start and end state (Section 2.1). To use this model we have to make two assumptions about our problem. First, that the probability of a transition between contexts depends only on the current context (*Markov property*), and second, that the probability of classifying an instance to a context depends only on the current context and the current setting. Both assumptions should hold, at least to a large degree, in most cases of use. For an example of a case where the Markov assumption does not hold, imagine a context that has a fixed duration, e.g., *work* – the probability of a transition out of *work* is heavily dependant on whenever *work* started eight hours ago. As an additional example, two sequential instances are more likely to be classified to the same context than the classification matrix would suggest due to them usually having similar underlying data. Nonetheless, even if the assumptions do not always hold perfectly, the model gives reasonable predictions as shown by our tests on real-life datasets (Section 5.2).

The key insight in creating the Markov-chain evaluation model is to correctly define Markov-chain states, so they 1) contain enough information to calculate transition probabilities between them, and 2) allow us to infer assignment properties from them. To do so we create a Markov chain that has $|C|^2$ states. Each state represents a pair (current context, context the system believes we are in), marked $\langle c, c' \rangle$ in short. An example of such a Markov chain can be found in Figure 3.1. The current context (the first part of the pair) is necessary to model how frequent that context is and which contexts are likely to come next. Knowing which context the system believes we are in, or equivalently which context did the system classify the current context to (the second part of the pair), is necessary

for modeling which sensor settings are active and thus the system performance using that setting.

The resulting Markov chain is never traversed directly – our goal is to calculate the *steady state* of this chain, which gives us the proportion of the time we would spend in each of those states, given infinite (or at least large enough) time. Note that the structure of the Markov-chain model is the same for each assignment, but different assignments have different transition probabilities and subsequently different steady states. Using this steady state, we can then make various predictions about the system performance for that assignment that are much better than the ones using the naive approach described in Section 3.1.1.1.

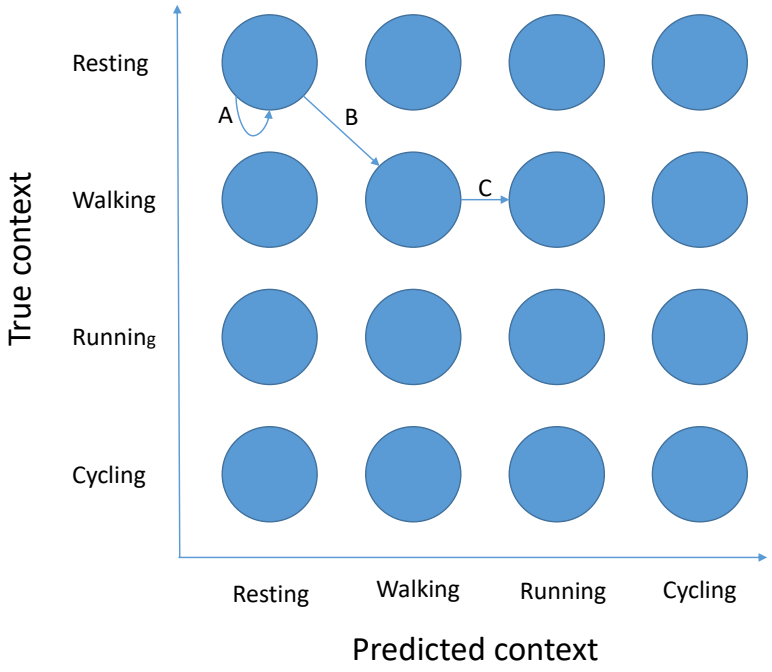

Figure 3.1: Markov chain where each state represents a (true, predicted) context of a subject. The states are fully connected, including connections to themselves, although most transitions were omitted for clarity. Transition A, for example, models the situation where the subject is continually resting, and their context is recognized as such. For transition B, the subject started walking, which was correctly recognized. Finally, transition C implies an incorrect classification of walking as running. The example comes from the E-Gibalec dataset, which is described in Section 4.5.

Similarly to the naive evaluation model, we begin by making experiments where for each setting a confusion matrix is calculated. This is again done without sensor switching and requires $|S|$ experiments. Additionally, we need the transition probability $T(c_i, c_j)$ from each context to each other, which can easily be inferred from the dataset. Finally, we must make energy consumption estimation per time unit for each setting $e(s)$.

To evaluate a particular assignment, we have to calculate the transition probability from one Markov state to another. They can be calculated from the transition probabilities of the contexts and data from the previously computed confusion matrices. Intuitively: we get in a state $\langle c, c' \rangle$ if the context really changes to c and if the system classifies this instance into c'.

$$P(\langle c_1, c_2\rangle \rightarrow \langle c_3, c_4\rangle) = T(c_1, c_3)CM_{a(c_2)}(c_4|c_3) \tag{3.2}$$

$P(\langle c_1, c_2\rangle \rightarrow \langle c_3, c_4\rangle)$ – the probability of a transition from state $\langle c_1, c_2\rangle$ to state $\langle c_3, c_4\rangle$.
$T(c_1, c_3)$ – the probability in the dataset that the next context will be c_1 given that the current one is c_3.
$a(c)$ – the setting assigned to context c.
$CM_s(c_4|c_3)$ – the probability that the classifier that works with setting s will classify an instance to c_4, if the true context is c_3.

Having the transition probabilities, we can use the basic Markov chain calculus to calculate the steady state of the Markov chain. This gives us the amount of time the system will be in each of the Markov states. Since we know how much time we spend in every Markov state, which setting is used in each state, and how much energy this setting consumes per time unit, the energy consumption of the whole system can be estimated. Additionally, since the previously calculated confusion matrices give us the accuracy for each setting, we can calculate the accuracy of the whole system in the same manner.

$$Energy(a) = \sum_{\langle c,c'\rangle \in M} t(\langle c, c'\rangle)e(a(c')) \tag{3.3}$$

$$Accuracy(a) = \sum_{\langle c,c'\rangle \in M} t(\langle c, c'\rangle)Acc(c|a(c')) \tag{3.4}$$

M – the set of all states in the Markov chain.
$a(c)$ – the setting assigned to context c.
$e(s)$ – the energy requirement of the setting s per time unit.
$Acc(c|s)$ – the accuracy of the classifier that works with setting s, if the true context is c.
$t(\langle c, c'\rangle)$ – the predicted proportion of time spent in state $\langle c, c'\rangle$, or equivalently, the probability of landing in that state.

It should be noted that many other metrics can be determined from such a model. Example: precision, recall, F-score or the latency of activity-change detection. They can be used instead of the presented accuracy or energy estimations when evaluating the performance of the system using a chosen assignment. Most of those metrics can be calculated directly from the confusion matrix for the whole system, which in turn is trivially calculable from the steady state. In fact, if one places the values of the steady state in a table that has the same structure as the states in Figure 3.1, they would get exactly the confusion matrix of the system that switches settings according to the given assignment.

$$\mathrm{CM}_{i,j} = t(\langle i, j\rangle) \tag{3.5}$$

$\mathrm{CM}_{i,j}$ – the value of the confusion matrix, for correct context i and predicted context j. Note that all the values are normalized to sum to 1; this can easily be remanded by multiplying them by the number of instances, if so needed.

3.1.1.3 Different System Schemes

Previous section considered a scheme where each context is assigned a setting. Those settings then change as different contexts are recognized. However, by adapting the Markov-chain model, similar system schemes can be constructed and evaluated. This section provides an example:

Consider a system that uses a two-step classification in the following way. It uses most of the sensor data to recognize the user's context, but then having recognized it, it uses limited data to determine if the context remains unchanged. Example: after recognizing that the user is sitting, only a low-sampling accelerometer is required to determine the persistence of the activity. After movement is detected, a general classifier starts working again to recognize the next context. Again the problem lies in assigning a sensible setting to each context-specific classifier and to the general classifier.

To model this system variation, "Transition" context is added to the list of contexts, and represents the state after the system detects a context change and starts the general classifier. "Transition" context is marked with 't' in all cases, while all other letters for states imply non-transitional context. A typical sequence of (true, predicted) contexts would be as follows: $\langle c_1, c_1\rangle \rightarrow \langle c_2, t\rangle \rightarrow \langle c_2, c_2\rangle$.

Transition probabilities are slightly more complicated as we now have two types of contexts, but the general concept and intuition behind them remains the same. The following equations list the transition probabilities and replace Equation 3.2 in Section 3.1.1.2. The first and second equation show the transition probability of 0, since the transitional context is always followed by a "real" one, and transition from one context to another always goes through the transitional context.

$$P(\langle c_1, t\rangle \rightarrow \langle c_3, t\rangle) = 0$$

$$P(\langle c_1, c_2\rangle \rightarrow \langle c_3, \neg c_2\rangle) = 0$$

$$P(\langle c_1, c_2\rangle \rightarrow \langle c_3, t\rangle) = T(c_1, c_3) CM_{a(c_2)}(\neg c_2 | c_3)$$

$$P(\langle c_3, t\rangle \rightarrow \langle c_4, c_5\rangle) = T(c_3, c_4) CM_t(c_5 | c_4)$$

$$P(\langle c_1, c_2\rangle \rightarrow \langle c_3, c_2\rangle) = T(c_1, c_3) CM_{a(c_2)}(c_2 | c_3)$$

$\neg c_2$ – the context different than c_2 or t.
$a(c)$ – the setting assigned to context c.
$T(c_1, c_3)$ – the probability in the dataset that the next context will be c_3 given that the current one is c_1.
$CM_s(c_4|c_3)$ – the probability that a classifier that works with setting s will classify an instance to c_4, if the true context is c_3.
Note: when c_i and c_j are listed in the same equation, i can equal j, unless otherwise stated.

This should give the reader an idea how to approach and model similar schemes. It also demonstrates the flexibility of using Markov chains for modeling system performance in cases where settings dynamically change.

3.1.2 Finding Sensible Assignments

In Section 3.1.1.2 we showed how to estimate the performance of any individual assignment, and Section 5 demonstrates that the estimation is both quick and accurate. Each assignment evaluates into a different trade-off between energy consumption and classification quality. As long as the number of assignments is reasonable, we can estimate all of them and then present the Pareto optimal set. This approach scales badly with the increasing number of settings and contexts, as the formula $|S|^{|C|}$ implies. Additionally, in Section 3.1.2.1 we prove that in general, Pareto-optimal assignments cannot be found in polynomial time. For large $|S|$ or $|C|$ a more efficient search for good (not necessarily ideal) trade-offs is required. However, since we are essentially solving a problem of multi-objective optimization, we can use the methodology from that research field (Section 3.1.2.2).

3.1.2.1 Theoretical Bounds

In this section we show that the problem of assigning settings to contexts cannot be solved in polynomial time, not even if we substantially simplify the problem. One could assume that improving the setting (by adding sensors, increasing sensor frequency etc.) will increase the system quality (e.g., accuracy), but decrease its energy efficiency, and that therefore a systematic and efficient way for constructing the Pareto front exists. This, however, is not the case.

First, the assumption does not hold: sometimes giving a method more data decreases its quality, especially if the new data is not relevant for the classification problem at hand. Second, even if it held, it would not be enough – we will show this by constructing a specific subproblem that is well behaved, and we will show that the subproblem can be translated to the well-known knapsack problem. Thus we will prove that the specific subproblem is NP hard, and by consequence that the general version of the problem is NP hard.

Consider the system *SimplifiedSystem* with the following properties.

1. The sensing system is composed from different sensors that are either on or off ($R = \{r_1, r_2, \ldots, r_n\}$)
2. Each setting s is defined as a set of indices of the currently active sensors (example: $s = \{r_1, r_3, r_6\}$)
3. Each sensor adds a fixed amount to the system quality and to the system energy consumption, regardless of what other sensors are currently active, depending only on which context $c \in C$ it was added to (adding a sensor is more beneficial for recognizing some contexts than others).

Example: If the accelerometer is activated when walking, the accuracy of the whole system goes up by 5%; if it is added to running, the accuracy goes up by 10%, if it is added to both, the accuracy goes up by their sum – 15%. This third property greatly simplifies the problem and allows us to calculate the quality and energy for a single assignment in the following manner.

$$Quality = \sum_{c \in C} \sum_{r \in \mathrm{s(c)}} q(c, r) \tag{3.6}$$

$$Energy = \sum_{c \in C} \sum_{r \in \mathrm{s(c)}} e(c, r) \tag{3.7}$$

$s(c)$ – the setting (set of indices) corresponding to context c.
$q(c, r), e(c, r)$ – quality and energy gains, given context c and sensor r.

Theorem 3.1. *In the SimplifiedSystem there are $2^{|R|}$ settings. Exhaustive search requires to check $(2^{|R|})^{|C|} = 2^{|R||C|}$ different assignments.*

Proof 3.1. Every subset of R defines its own setting and there are $2^{|R|}$ subsets. Since we can independently assign one of the $2^{|R|}$ settings to each of the $|C|$ contexts, we get a total of $(2^{|R|})^{|C|}$ different assignments.

Theorem 3.2. *Any method for finding Pareto-optimal assignments for the SimplifiedSystem has the time complexity of $O(2^{|R||C|})$ in the worst case. This is no better than the exhaustive search.*

Proof 3.2. We can interpret every (context, sensor) combination as an item with value $q(c, r)$ and weight $e(c, r)$. This way we get $|R|\,|C|$ different items. Mimicking the knapsack problem, we can put some of the items in a knapsack. Every assignment of settings to contexts corresponds to placing all the corresponding items in the knapsack. This correspondence is bijective: for each knapsack filling, there is an appropriate unique assignment. In the original knapsack problem, we have to find the best value given a weight limit. Let us define *ParetoKnapsack* as a problem where we have to find the best value for each possible weight limit (a Pareto front of value against weight). This is a strictly harder version of the problem and is therefore also NP hard.

The problem of assigning settings to contexts in *SimplifiedSystem* can therefore be directly translated to the *ParetoKnapsack* problem. The worst case for the time complexity of the knapsack problem is $O(2^{\text{number of items}=|R||C|})$.

We have shown that even the simplified version of the problem cannot be efficiently solved if we desire an optimal solution, which implies that neither can be the general version. This justifies our use of heuristic methods to find near-optimal solutions.

3.1.2.2 Multi-Objective Optimization

To perform an efficient search we used multi-objective optimization – the two objectives being the energy efficiency and the classification quality. We used the NSGA-II [6], a genetic multi-objective optimization algorithm, for the task. This choice was made due to the wide-spread use of the algorithm for problems with few objectives. The choice was confirmed by gaining sensible results in tests with real-life datasets (Section 5.2). We assume, however, that some other similar algorithm could be used in its place.

The evaluation of any input into energy/quality estimates can be done on the fly, when a specific assignment has to be evaluated. This would be excessively time-consuming if we used actual simulation instead of evaluations with the Markov-chain model as hundreds of evaluations have to be performed for each generation in the genetic algorithm.

Assignments of settings to contexts are used as the inputs to be optimized. If the settings themselves have a sensible structure – and if we have a huge amount of settings, they are almost bound to have it – we can use this structure to encode the inputs, to make algorithm operations (crossover, mutations) more natural. When working with different datasets, we used different encoding structures.

For example, for the Commodity12 dataset (Section 4.4) each setting represented which sensors are turned on or off. This can be encoded by a binary string of length s, where s is the total number of sensors, e.g., "110000" would represent that only the first two sensors are working. Each context can have a different setting assigned, thus the total length of the encoding is cs, where c is the number of contexts, e.g., "110000-100000-00001-00001-00001-00001-00001" represents having only the first two sensors active when the first context is recognized, only the first sensor when the second context is recognized and only the last one otherwise. We used bit-flip mutations and the classical crossover. The same type of representation (with a different number of sensors) was used for the SHL (Section 4.2) dataset.

Another example comes from the E-Gibalec (Section 4.5) and Opportunity (Section 4.3) datasets. There we worked with a small number of settings, which we enumerated. We then used integer vector encoding; e.g., (2,4,35,1) would signify to use the setting that was enumerated as "2" for the first context, setting "4" for the second one and so on. Mutations in this case are the transformation of one vector component into a random value between 0 and the number of settings. Again, the classical crossover was used.

We implemented an additional optimization that sped up the search process: adding

two solutions to the starting population – one that always uses the most accurate setting for all contexts (e.g., all sensors with maximal frequency) and another that uses the most energy-efficient setting for all contexts. These two solutions are known in advance to lie at the two extremes of the Pareto front, and including them substantially increased the performance of the NSGA-II algorithm.

In all cases we used the population of size 100 and let the algorithm run for 500 generations. Experimental tests showed that these parameters were adequate for all of our problem domains. As always when using NSGA-II or any similar algorithm, the parameters have to be adapted to each particular problem – and one is encouraged to do so when implementing this methodology on their own domain.

3.2 Duty-Cycle Assignment (DCA)

Suppose we have the same context recognition system as in the previous case – it classifies sequential time periods into contexts $c \in C$. The quality of this classification is given by the confusion matrix CM.

We then add the duty-cycling – with parameters w and len – to this system in the following way: sensors are working for w periods, classifying that many contexts. After w classifications, the last classified context c defines the length of the sleeping phase. The system then turns off the sensors for the next $len(c)$ periods, and classifies all contexts in this time as c; i.e., it assumes that the context has not changed.

Our goal is two-fold. First, to create a mathematical model that will quickly and accurately predict the behavior of this system – the proportion of the time the sensors are sleeping and the confusion matrix of the newly classified contexts (Section 3.2.1). Second, to propose a way of selecting appropriate parameters w and $len(c)$ for each c (Section 3.2.2).

This can be considered a similar problem as the one being solved by the SCA method (Section 3.1), with the exception that all settings are simply the lengths of the duty-cyle phases (w and len). Using the SCA, we would make an experiment for every (w, len) combination in a given range, which can be time consuming given sufficient range. This method leverages the fact that the performance of a setting can be deduced without an experiment. Instead of doing $|S|$ experiments (one experiment for each setting), only one – the case with no duty-cycling – is required. This allows us to evaluate many different duty-cycle parameters and thus increases the chances of finding a good solution.

3.2.1 Modeling Duty-Cycles

When building the mathematical models for estimating the performance of the system that employs duty-cycling we make the same assumptions as in Section 3.1.1.2. I.e. that the context sequence has the Markov property – the probability of the next context depends only on the current one and that the classification errors happen randomly, with the distribution given by the confusion matrix.

Given the assumptions, the evaluation of a system with given duty-cycle parameters can be performed with the subsequently defined steps. This evaluation needs only a few matrix (of size $|C|$) multiplications and summations and can therefore be done almost instantly; in any case much more quickly than running a complete experiment (simulating duty-cycling on a dataset, classifying all the instances). The speed can be additionally increased due to the fact that most calculations can be stored from the evaluation of one assignment to the next.

3.2.1.1 Naive Evaluation

Before describing the relatively complex mathematical model, we list three simple ways of estimating the system performance. These approximations are not used by our method, but may serve as a shortcut for system designers to get an intuition about how duty-cycling affects the performance. Here we assume that all contexts use the same duty-cycle length which is smaller than the average length (avg) of contexts. In addition we assume that all contexts are correctly classified during the active phase.

a) Suppose a context is recognized, the sleeping phase begins, and when the system resumes with the active phase, the context remains unchanged. If len is much less than avg, this repeats several times before the context changes. It is reasonable to assume that the context change on average happened in the middle of the sleeping phase. So on average, the system misclassified $len/2$ instances. The accuracy of such a system can thus be represented with Equation 3.8. Note that the length of the active phase is never explicitly taken into account.

$$Accuracy = \frac{avg - \frac{len}{2}}{avg} \tag{3.8}$$

b) Similarly, we could calculate the probability of context change happening, and assume that if it did not, the system classified all instances correctly, otherwise only half of them. If the probability of context change from one instance to another is p (in Markov models, $p = \frac{1}{avg}$), then the probability that the context did not change during the sleeping phase is $(1-p)^{len}$ or q^{len} if we define $q = 1-p$. This assumes that during the sleeping phase, the context cannot change and then change back to the original one. First w instances are always correct due to the active sensors. The resulting approximation is shown with Equation 3.9. Note that in Equation 3.8, we were approximating the accuracy by looking at the average proportion of a context that is classified correctly. This and the following equation are approximating accuracy by looking at the average proportion of an active+sleep (which has the length $w + len$) cycle that is classified correctly.

$$Accuracy = \frac{w + q^{len} len + (1 - q^{len})\frac{len}{2}}{w + len} \tag{3.9}$$

c) The last two approximations assume that the context change happens in the middle of the sleeping phase. This may not be the case when len starts to approach avg. However, it is simple to calculate the probability of transition at each step. After the sleeping phase starts, the chance of the first context remaining the same (context transition did not happen) is $1-p$. The chance of the second one remaining the same is $(1-p)^2$ as the transition could happen at two different points. The third one is $(1-p)^3$ and so on. This yields a finite geometric sum with len terms that resolves as shown in Equation 3.10. Again we took into account that the first w instances are always correct.

$$Accuracy = \frac{w + \frac{1-(1-p)^{len}}{1-p}}{w + len} \tag{3.10}$$

As shown in Figure 5.5 and Figure 5.6, all three equations – labeled as "Approximation A", "Approximation B" and "Approximation C" accordingly – can serve as a loose approximate if their assumptions hold, but their performance quickly deteriorates if one of them (e.g., the sleeping phase is too long) breaks.

3.2.1.2 Duty-Cycle Types

The first step of the "real" evaluation model [91] is to break the data into duty-cycles of different types that start with the last active period ("head") as shown in Figure 3.2. The cycle type depends on the true context of that period, in addition to the context that period is classified as – indices of these contexts are subsequently named t and p, respectively. If the last active period before sleeping is *walk* that is classified as *run* we have a $\langle walk, run\rangle$ cycle. The first component of the cycle allows us to calculate the likely context transitions, while the second one determines the length of the sleeping phase.

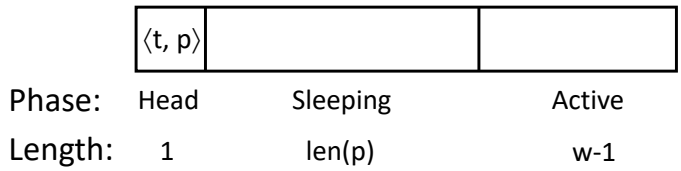

Figure 3.2: Structure of a $\langle t, p\rangle$ duty-cycle. It starts with the context t classified as p, followed by the sleeping phase (sensors not working) where all contexts are classified as p, and an active phase (sensors working). The last period in the active phase is the head of the next cycle.

For each of those cycle types, we can use the following equations to estimate how many of each contexts appear in both the sleeping (3.13) and active (3.14) phase. We start by the state distribution in the *head* of the $\langle t, p\rangle$ cycle – by definition the state there is known and thus the state distribution is e_t. For each next time period we can estimate the state distribution by multiplying the previous distribution with the transition matrix T. For more information on state distributions see Section 2.1. Having the context probability at each time step, their total expected number is simply the sum of their probabilities at each step.

$$e_i = [0, .., 0, \underbrace{1}_{i} 0, ..., 0]^T \tag{3.11}$$

$$P_{t,i} = e_t T^i \tag{3.12}$$

$$ES_{t,p} = \sum_{i=0}^{len(p)} P_{t,i} \tag{3.13}$$

$$EA_{t,p} = \sum_{i=len(p)+1}^{len(p)+a-1} P_{t,i} \tag{3.14}$$

e_i – the i-th base vector. It represents the context distribution in the first period of a cycle.
T – the transition matrix, $T_{i,j}$ represents the probability of the next context c_j, given the current context c_i. Easily obtained from the dataset.
$P_{t,i}$ – the probability distribution of contexts after i steps, given the current context c_t. Vector of size $|C|$.
$ES_{t,p}$ – the average number of contexts that appear while sleeping, given that this sleeping phase started with the true context c_t that was classified as c_p. Vector of size $|C|$.
$EA_{t,p}$ – the average number of contexts that appear while active, given that this cycle started with the true context c_t that was classified as c_p. Vector of size $|C|$.

3.2.1.3 Distribution of Duty-Cycle Types

In the previous section we showed how a specific $\langle t, p \rangle$ cycle behaves, given we know its type. The next point of interest is determining the expected proportion of each cycle type. To do so we first calculate the probability that a cycle of type $\langle t_1, p_1 \rangle$ follows a cycle of type $\langle t_2, p_2 \rangle$ for each t and p. Intuitively, this happens if t_1 transitions to t_2 after $len(p_1) + w$ steps (length of the cycle) and if the t_2 is classified as c_2. The probability of that happening is expressed in Equation 3.15.

$$P(\langle t_1, p_1 \rangle \rightarrow \langle t_2, p_2 \rangle) = (P_{t_1, len(p_1)+a})_{t_2} CM_{t_2, p_2} \tag{3.15}$$

$(P_{i,j})_k$ – the k-th component of the vector $P_{i,j}$
$P(\langle t_1, p_1 \rangle \rightarrow \langle t_2, p_2 \rangle)$ – the probability of the next cycle type being $\langle t_2, p_2 \rangle$, given the current cycle type $\langle t_1, p_1 \rangle$. This happens if the last context of the previous cycle is t_2, classified as p_2.

Having the transition probabilities, we can use the basic Markov-chain calculus to calculate the steady state in the similar way as in Section 3.1.1.2. The steady state corresponds to the desired cycle-frequency proportions D.

3.2.1.4 System Evaluation

Combining the results from the previous two sections, we can estimate the performance of the system in the form of its confusion matrix (3.16) and its energy gain (3.17). Energy gain is defined as the average number of sleeping periods for each active one. If the sensors are on half the time, $Gain = 1$.

The intuition behind calculating the energy gain is simple. We know how long the cycle is given its type. We also know the distribution of each type. The double loop in Equation 3.17 simply iterates over all cycle types and sums the two terms together. The calculation of the new confusion matrix is slightly more complex and can be split into two parts: performance while sleeping and the performance during the active phase. While sleeping in a cycle of type $\langle t, p \rangle$, all the contexts are classified as p while the proportion of the contexts actually appearing in that period is given by the ES. On the other hand, while active we know the proportion of appearing contexts (EA) as well as the misclassification of each context. This is formalized in Equation 3.16.

$$CM^*_{t,p} = \sum_{i=1}^{|C|} D_{i,p} (ES_{i,p})_t + \sum_{i=1}^{|C|} \sum_{j=1}^{|C|} D_{i,j} (EA_{i,j})_t CM_{t,p} \tag{3.16}$$

$$Gain = \frac{1}{w} \sum_{i=1}^{|C|} \sum_{j=1}^{|C|} len(j) D_{i,j} \tag{3.17}$$

D – the distribution of the cycle types. $D_{i,j}$ is the proportion of $\langle i, j \rangle$ cycles.
$CM^*_{t,p}$ – element (t, p) of the confusion matrix of the system that employs duty-cycling with given parameters. The first part of the equation is the performance in the sleeping phase, the second in the active phase. Note: The sum of all the elements in this new matrix is different than in the original CM matrix; this can be easily solved with normalization if need be.
$Gain$ – the proportion of sleeping periods compared to the active ones.

3.2.2 Finding Duty-Cycle Parameters

Section 3.2.1 gives a way to evaluate the performance of a system with given parameters, while here we discuss methods that use this evaluation to efficiently search for sensible parameters. The space of possible parameters is usually large even if we limit the maximum cycle length: if we have $|C|$ contexts with $len \in [0, m_c]$ and $w \in [0, m_w]$, then there are $m_w m_c^{|C|}$ different parameters – making the brute-force approach infeasible.

The first insight is that according to the described model, there is no advantage in having the active phase w longer than the minimum of 1 in most cases. Intuitively, if a context is assigned a sleeping phase with length l, then it is assumed that the context will not likely change in the next l periods; classifying multiple instances in a row is therefore unnecessary. The exception is when trying to get very small energy consumption reductions (to minimally decrease performance), as a *Gain* smaller than 1 is impossible to achieve with $w = 1$. There are two practical advantages, however, in having $w > 1$; a) if there is a cost in turning the sensors on or off (GPS is a notable example) and b) if the classification accuracy is poor, since one might consider classifying more than one instance and applying some smoothing before deciding on the next sleeping phase length.

To decide on the sleeping phase lengths, we can resort to one of the following two methods.

3.2.2.1 Iterative Method

Duty-cycling causes an error if a sleeping period coincides with a transition between two (or more) contexts: c_1 and c_2 for example. Two types of errors (3.18) happen: a) Precision of c_1 drops, as instances of c_2 are misclassified as c_1. The drop in precision depends only on $len(c_1)$ and the average length of c_1 streak. b) Recall of c_2 drops, and the drop depends on $len(c_1)$ and on the average length of c_2 streak.

$$\overset{\text{Precision of } c_1 \text{ and recall of } c_2 \text{ fall}}{\overbrace{}}$$
$$c_1 \quad \underbrace{c_1 \quad c_1 \quad c_2 \quad c_2}_{\text{sleeping period}} \quad c_2 \tag{3.18}$$

Therefore, $len(c_1)$ should take into account both the streak length of c_1 and streak lengths of the contexts that are likely to follow it. If we have requirements on precision/recall for each context we can iteratively find the upper bounds for $len(c)$ for each c. This can be done by increasing the $len(c)$ for a given c (while not duty-cycling any other context) until either the precision or recall of some context drops below the required boundary; and then repeating the process for all other contexts. If we are only interested in precision, the found upper bound is tight, and ideal given requirements. This method is guaranteed to work only if we assume that the classifier works perfectly, as it does not model errors generated by misclassifications.

Using this method can serve as a semantic explanation for the sensible sleeping lengths – for example, it may show that some contexts should not use long sleeping lengths because they are too short or because they are followed by another short context.

3.2.2.2 Multi-Objective Optimization

Similarly as when describing the SCA method, we have an optimization problem with two conflicting objectives: reducing the system's energy requirements, while retaining its classification quality – accuracy/F-score/recall of the most important context or any other metric that can be computed from the confusion matrix of the system. Given the similarity of the problem, we used essentially the same methodology as described in Section

3.1.2.2: We used NSGA-II algorithm in combination with the previously described mathematical model that can evaluate the performance of an assignment on the fly. The result is once again a non-dominated set of solutions, each giving a different trade-off between energy consumption and performance, from which the system designer, who knows the requirements of the system, can pick a suitable solution.

The population was encoded as an integer vector, where the i-th component signified the length of the sleeping phase of the i-th context. The length of the active phase w was not included, as the experimental test confirmed that the length $w = 1$ is ideal. The length of the sleeping phase had an artificial upper bound that was determined independently for each dataset depending on the average lengths of the contexts in it. We used the same algorithm parameters and the same algorithm optimization (seeding two known solutions in the starting population) as in Section 3.1.2.2. In addition, when evaluating a solution, we stored all results given by Equations 3.12, 3.13 and 3.14. These stored matrices can be reused when evaluating the next solution, drastically decreasing the overall time needed for an NSGA-II run.

3.3 Cost-Sensitive Decision Trees (CS-DT)

Both the previously presented SCA and DCA methods assign a setting to the current context. While this can give a lot of flexibility for energy-consumption optimization, it might not always be adequate. A system might classify only a few different contexts or it might not classify discrete contexts at all (e.g., the system uses a regressor instead of a classifier). It could also be the case that some contexts are actually composed of different "sub-contexts"; e.g., the system recognizes *traveling*, but this context could represent either *walking* or *car* and might need different sensor settings to be detected. In these cases it might be beneficial to assign a setting depending on the underlying data instead of the classified context. As an example, the system could have a rule: "if the speed measured by the GPS<x, turn the GPS off and turn on the accelerometer instead".

Our proposed solution uses cost-sensitive learning, which is a machine learning field concerned with both the misclassification cost and with the cost of features that were used to generate the classification. A typical example comes from the field of medicine, where each test on a patient has an associated cost (in money, time, discomfort, etc.), but at the same time increases the probability of a correct diagnosis. The goal is to balance the feature costs with the misclassification cost, for example by trying to minimize their sum.

In the case of context recognition, each feature is calculated from data that is coming from one of the sensors. Its cost comes from having that sensor active, draining the battery of the sensing device (with the additional battery drain resulting from the data processing being implicitly included in this cost).

The presented methods assume that we have a dataset collected from different sensor streams. Data is then split into individual instances (composed of different features), and the goal is to use a machine learning model to classify all of them, each to one of predetermined contexts. We aim to reduce the expected cost of this classification – that is, the sum of the misclassification cost and the cost of the features used. Note that while this section assumes the discrete case, the algorithms can be adapted for the case with a continuous target variable. Additionally note that many sensor settings can be optimized using this methodology, not only selecting the sensor to use – e.g., sensor streams can be generated by the same sensor when working with different frequencies.

3.3.1 Cost-Sensitive Model

A cost-sensitive decision tree is used as the machine learning model for this method due to its popularity as a cost-sensitive model and due to how naturally it can be adapted to the context-recognition setting. As mentioned in Section 2.4.2.3 we chose and implemented the cost-sensitive decision tree as presented in the work of Ling et al. [88], and modified it as described in Section 3.3.2 (we will refer to our modified tree as CS-DT). This particular implementation was chosen because it makes the sum of the feature and misclassification cost explicit, and the trade-offs between them easy to understand. At each tree node – instead of checking each feature's information gain or a similar metric – we compare the expected cost of misclassification if this node becomes a leaf, with the expected cost of the feature and misclassification if the feature is used to further divide the instances. The feature that reduces the expected cost the most, if any, then expands this node.

To illustrate with an example: using the node as it is, yields a misclassification cost of 5. If you expand the node with feature A, the misclassification cost falls to 2, but in exchange you must pay the feature cost for feature A – 4. Alternatively, if you split the node with feature B, the misclassification cost is 3 and the feature cost is 1. The total costs for all three cases is therefore 5, 6 and 4, respectively, and thus the feature B is chosen. If the total cost of the original node were to be the lowest, that node would become a leaf node.

An immediate concern is that feature and misclassification costs are measured in different units with no obvious way to combine them. The most straightforward solution to this problem is to decide on a trade-off in advance (for example, one might be willing to sacrifice 2% of accuracy for an extra hour of battery life). This approach, however, requires some expert knowledge of the domain and it yields only one solution. The alternative that was used in this work was to iteratively test different trade-offs and then present the Pareto front approximation of different solutions to the user. To create the different trade-offs, the misclassification cost was iteratively scaled up by multiplying it by a constant.

3.3.2 Using CS-DT for Context Recognition

This context recognition setting is somewhat different from the standard cost-sensitive setting; the differences and our adaptations are listed in Sections 3.3.2.1–3.3.2.4.

3.3.2.1 Delayed Sensor Activation

In a real-life setting it is impossible to retrieve a sensor reading for the current instance, if the sensor is not currently active. Therefore, we cannot simply traverse the decision tree and calculate the features as necessary. We can, however, leverage the fact that the next instance will be similar to the current one, and that probably the same sensors will be needed. The CS-DT is modified so that while it is classifying an instance, it traverses the tree until it comes to a leaf node or a node with an unknown feature. Then, it classifies that instance using that node as a leaf node, and in the case of an unknown feature, it signals to the system to turn the needed sensor on. It also notes which sensors were not needed, and they are turned off for the next instance. This way, the system gradually and fluidly adapts its sensing capabilities to the current need. This process is graphically illustrated in Figure 3.3.

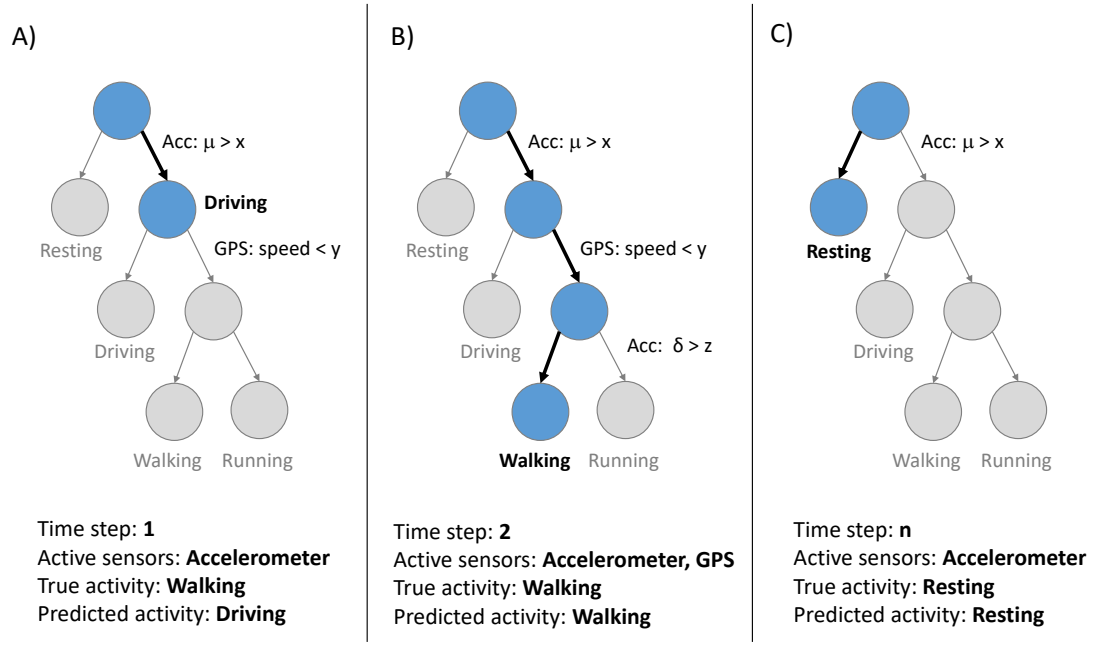

Figure 3.3: Demonstrating the delayed sensor activation in a CS-DT. We start (A) by having only the accelerometer active. Using it we traverse the first half of the tree, where we need the value of a feature that is derived from the GPS sensor. Not having it, we make the classification as if that node was a leaf node and then turn the GPS on. In the next instance (B) we can traverse the tree as normal and make the correct classification. This continues for n steps, after which (C) we go to the left part of the tree where no GPS feature is required. As a result, in the next instance the GPS is turned off.

3.3.2.2 Shared Feature Costs

A data stream from a single sensor can be transformed into many different features. These features have a shared cost – calculating the first requires activating the appropriate sensor, while the subsequent features derived from the same sensor are essentially free. Furthermore, the costs of different sensors are not independent from each other (e.g., look at the Commodity12 dataset in Section 4.4). To solve both problems, the CS-DT was adapted so that when the tree is traversed, it tracks which features were used in all previous nodes. These features serve as an additional input for the cost function of the current node's feature.

To make it more concrete: the decision tree takes a cost function that takes a set of features and returns their total cost – $f(r_1, r_2, ..., r_n) \rightarrow \mathbf{R}$. At each node, the cost of a feature r is then defined as:

$$energy(r) = f(r_1, ..., r_n, r) - f(r_1, ..., r_n)$$

$(r_1, ..., r_n)$ – features used in the nodes above the current one.

This can be interpreted as how much additional energy is needed to use this feature, compared to not using it. If the cost of every feature is independent of each other, the above definition is equivalent to $f(r)$. If all features of the same sensor share a cost, and if the feature r comes from the same sensor as some other feature r_i, then $f(r) = 0$.

3.3.2.3 Taking Advantage of Shared Feature Costs

Tree generation process is myopic – each feature is evaluated individually. This problem is compounded by the shared feature costs. One feature, for example, may not be worth activating the corresponding sensor for, but all the features from that sensor could be. To partially mitigate this, we changed the CS-DT generation so that when at a particular node no feature reduces the expected cost, the best feature expands the node anyway. This can happen maximally n times in each tree branch (n is a parameter). This is often beneficial because the feature selected for the next node is often free and the two can be jointly useful. The tree is then pruned using a different independent part of the dataset. Pruning works backwards, and removes the nodes whose expansion increased the total (misclassification and feature) costs.

3.3.2.4 Advanced Use of Shared Feature Costs

An alternative to the solution presented in Section 3.3.2.3 is to drastically change the tree-generation process. At each node, instead of choosing the best feature, we choose the best sensor by comparing sub-trees potentially containing all features using a single sensor. The best sub-tree replaces the node and this process is then repeated at each leaf of the sub-tree. This resulted in most cost-efficient trees, but at the cost of higher generation time. Which features are derived from which sensor must be provided by the user in the form of a mapping function. Note that this whole approach is sensible only in the case where features can be split into groups that completely share a cost – adding one feature of the group is as costly as adding all of them.

3.3.3 Generating Different CS-DTs

This section lists three methods for generating different CS-DTs. Generating multiple CS-DTs is useful for getting different trade-offs between energy consumption and classification quality. This in turn can be used for presenting a Pareto front of different solutions to the system's developer, or for assigning different CS-DTs to different contexts (Section 3.4.2).

3.3.3.1 Changing Misclassification Cost

The first method and the most straight-forward method was already described in Section 3.3.1. Different CS-DT can be generated by gradually changing the misclassification cost. Note that these trees are all semantically similar – usually only increasingly smaller as the misclassification cost falls. Misclassification cost of 0 generates a majority classifier that always predicts the most common context, while the misclassification cost of ∞ behaves as the regular decision tree.

3.3.3.2 Using Different Settings

Another way to create different energy/quality trade-offs is to use different settings (as with the SCA method from Section 3.1) to create a different tree using each setting. The optimization in this case can be looked as being performed in two steps: choosing a setting can already imply some energy savings, but the CS-DT can optimize that setting further. The most common example we used in this work was to use active sensor subsets as the setting for the system. Using this method, one CS-DT was generated from each possible sensor subset – using only features from those sensors. Algorithm generating the CS-DTs may determine that not all features from the given subset are needed, at least not all the time, increasing the energy-efficiency further.

In theory the method from Section 3.3.3.1 should already find all the best energy/quality trade-offs, however, in practice we found it beneficial (although more time consuming) to use this method for generating as diverse solutions as possible. An additional benefit is that the CS-DTs generated this way are more semantically different and are thus more suitable for the use in combination with the SCA method (Section 3.4.2).

3.3.3.3 One-vs-all Trees

Another method for generating different CS-DTs is to generate one tree for each context. Such context-specific trees, for example for the context c only classifies instances into "c" or "not c". To use these trees, the system architecture is changed to perform a two-step classification. In the case of "not c", a general-purpose decision tree is used to classify the next context. This is similar to the scheme described in Section 3.1.1.3. The whole dataset – with modified labels – is used for each contex-specific tree generation. This approach can be combined with the one one described in Section 3.3.3.1 to generate more trees. These type of trees are solely used in combination with the SCA method (Section 3.4.2).

3.3.3.4 Context-Specific Trees

Another way for creating different CS-DT is to train trees on different portions of the dataset. Given that such trees are to be assigned to different contexts, a natural idea is to train a tree only on data from and around one context. *Walking* context-specific classifier, for example, could be trained only on data where the user walked and the data that shortly follows. Such a classifier should theoretically be well suited to detect *walking* and distinguish it from possible transitions.

3.4 Combining the Methods

So far, we described three different methods for optimizing the energy consumption, each having its own advantages and disadvantages. The Setting-to-Context Assignment (SCA) method (Section 3.1) is very general and can work with basically any setting, but can only optimize at the granularity level of a context. It requires one experiment for each setting to predict the performance of a system that dynamically switches between them. The Duty-Cycle Assignment (DCA) method (Section 3.2) negates the last point by mathematically modeling the performance of a setting, but does so only for duty-cycle settings. Lastly, the Cost-Sensitive Decision-Trees (CS-DT) method (Section 3.3) can optimize at the granularity level of the feature data, but not on the context level. It has the additional downside of requiring to use the decision trees as the classifier.

To combine the advantages of the methods and to partially mitigate the disadvantages we propose a few ways of combining the approaches by either running one method after another or by fusing them into one method.

3.4.1 Combining the SCA and DCA Methods

The DCA method assigns a duty-cycle setting to each context. To do so it requires only the transition matrix of the context and the confusion matrix of the system. As such it can be applied alongside basically any other setting optimization. In this case, one can use the SCA method to optimize all settings but the duty-cycling. From the set of trade-offs, one is chosen and the confusion matrix of the corresponding system (as calculated by the SCA model) is then used as the input to the DCA method. This way the duty-cycling is added to whatever setting optimization was performed by the SCA method with

no additional experiments needed (this combination will be referred to as SCA + DCA). In most practical cases, the optimization of duty-cycling is completely orthogonal to the optimization of some other system's setting.

To make the method combination slightly more accurate in its prediction of the system performance, one has to consider a peculiar interaction between the two methods. If the SCA method, for example, decides to use all sensors for context a and only a few sensors for context b, then duty-cycling when context a is detected is more beneficial then when context b is detected. The latter follows from the fact that classifications are more costly for the context a. To account for this, we can calculate the actual energy consumption in addition to the simple *energy gain* (Equation 3.17). To do so, we multiply the energy consumption of different settings (that are assigned to each context) with the distribution of the corresponding duty-cycle types (Equation 3.19). This gets us the average energy consumed when the sensors are active, from which it is easy to calculate the total energy (Equation 3.20).

$$ea = \sum_{i=1}^{|C|} \sum_{j=1}^{|C|} e(s_j) D_{i,j} \tag{3.19}$$

$$Energy = w * ea + len * es \tag{3.20}$$

D – the distribution of the cycle types. $D_{i,j}$ is the proportion of $\langle i, j \rangle$ cycles.
w – length of the active phase.
len – length of the sleeping phase.
ea – the energy consumption in the active part of the duty-cycle.
es – the energy consumption in the sleeping part of the duty-cycle. This can be either 0 or the baseline cost of the context-recognition application.

One downside of running one method after another is in the roughly double running time. The other is that the SCA method returns a Pareto front approximation of different solutions. Of them, one has to pick one on which to apply the DCA optimization. While DCA method again generates a Pareto front approximation of solutions, it is undesirable to have an in-between non-automatic step that has to be made by the user. One way to mitigate the burden of this selection is to automatically select a few interesting trade-offs – for example equally spaced points on the Pareto front approximation – and calculate the DCA optimization for each of them, and then collect the best solutions from the joined set. This approach, however, again increases the computation cost. Nonetheless, we believe this combination can be highly beneficial as the experiments in Section 5 show.

An alternative to the above approach is to consider that the downside of optimizing the duty-cycles with the SCA method directly is the need for many additional experiments (one for each possible duty-cycle length). The DCA method on the other side can predict the performance of any particular duty-cycle assignments – thus the CM_s in Equation 3.2 which is usually acquired by an experiment, could be substituted by this prediction. This in theory seamlessly combines the two methods, however, it relies on combining two different mathematical models, which can have some unpredictable side-effects. The test of this combination will be left to the future work.

3.4.2 Combining the SCA and CS-DT Methods

CS-DT are good at dynamically activating different sensors. However, classifying all instances using the same CS-DT can sometimes be inflexible. For example, the sensor at the root of the tree will always be active regardless of which context the user is currently in. Having a separate tree for each context and using it when that context is recognized might thus be preferable. Intuitively, a different rule set for sensor switching should be used depending on whether the user is resting at home, for example, or running outside. An example of this can be found in Figure 5.37. This approach adapts to both the current sensor data (within CS-DT) and the last classified context (switching between the CS-DTs), allowing the system to find interesting energy-efficient solutions. The adaptation to use different CS-DTs for each context can be made almost entirely with the SCA methodology – by defining the possible settings used by SCA as different CS-DTs.

This CS-DT + SCA combination starts by generating many different trees as described in Section 3.3.3. That section describes four different ways of making the trees, of which any (or all) may be used. Instead of assigning a setting to a context, a CS-DT is assigned instead. The performance of an assignment is evaluated using the same Markov-chain method as described in the SCA method, and the search for ideal assignment is similarly performed using the NSGA-II algorithm.

One slight improvement can be made to the SCA prediction accuracy, when using CS-DTs as settings. So far we assumed the settings consume a fixed amount of energy per time unit – $e(s)$. This is not the case for CS-DTs; depending on the underlying context, different branches of the tree could be used, resulting in different energy costs. The adjustment is made by changing Equation 3.3 to Equation 3.21.

$$Energy(a) = \sum_{\langle c,c' \rangle \in M} t(\langle c, c' \rangle) e(a(c')|c) \tag{3.21}$$

$t(\langle c, c' \rangle)$ – the proportion of time spent in state $\langle c, c' \rangle$.
$e(a(c')|c)$ – the average energy consumed when using the setting assigned to context c' if the true context is c.

3.4.3 Combining All Three Methods

As described in Section 3.4.1, the DCA can be run after any previous optimization technique. One could therefore try to use all three methods by first using the (SCA, CS-DT) combination and then applying the DCA optimization as the second step.

Unfortunately, some unpredictable performance issues can happen due to the peculiar interaction of the duty-cycling and dynamic sensor switching using a context-specific CS-DT. After a new context is detected, the system switches to a new CS-DT, but not all sensors for working with the current branch are yet active. Usually, after the next few instances the system activates the required sensors (and turns off the unnecessary ones); however, duty-cycling can interfere with this process by turning all sensors off periodically.

The combination (referred to as CS-DT + SCA + DCA) can still be used, but some adjustment and limitations must be applied first. First, the duty-cycles should have the active phase w at least as long as the number of sensor streams. This ensures that the leaf node of the CS-DT is reached before the sleeping phase of the duty-cycle starts (otherwise all the contexts while sleeping could be misclassified). Second, when the sleeping phase starts, the CS-DT remembers which branches (and corresponding sensors streams) were active and reactivates them when the active phase starts.

To summarize the three-method combination we show the pseudocode of two algorithms. First one (Algorithm 3.1) shows how to create good assignments. It returns a set of assignments from which the system designer picks the most suitable one.

Algorithm 3.1: CS-DT + SCA + DCA (Create assignments)

Data: Contexts C, Settings S
Result: Setting-to-context assignments using the specified settings, duty-cycling and CS-DTs.

```
// generate different CS-DTs (Section 3.3.3)
trees = generateCSDT(C, S);
// use CS-DTs as settings for SCA method (Sections 3.1 and 3.4.2)
scaAssignments = sca(C, trees) ;
// select few assignments to be the baseline for the duty-cycling
// for example choose them equidistant in energy consumption
scaAssignmentSubset = selectSubset(scaAssignments);
scaDcaAssignments = [];
for a ∈ scaDcaAssignemts do
    // Apply DCA method, while taking the assignment a as the baseline
       (Sections 3.2 and 3.4.1)
    scaDca = dca(a);
    scaDcaAssignments.append(scaDca);
end
// eliminate all assignments that are Pareto-dominated by some other
   assignment
return paretoOptimal(scaDcaAssignments)
```

The second algorithm (Algorithm 3.2) shows how to apply all three methods when the context-recognition system is in use. The working of this algorithm is also later shown on a simple example (Figure 5.21).

Input to the algorithm is a single instance, which is then classified into a context. In addition we provide it with the previously selected assignment that we here explicitly split into two parts:

$s(c)$ – the setting assigned to the context c. In this case settings are different CS-DTs.
$len(c)$ – length of the sleeping phase, given the current context.

Algorithm assumes the existence of the following global variables that keep the internal state of the system from one instance to another.

context – last classified context
decisionTree – the CS-DT currently in use
currentSensors – a list of sensors that are currently needed. Unless the system is in the sleeping phase of the duty-cycle, these sensors are currently turned on. While this pseudocode assumes that the sensors are system settings being adapted, any kind of data-streams (e.g. different frequencies of the same sensor) can be used instead.

Algorithm 3.2: CS-DT + SCA + DCA (Use assignments)

```
Data: Instance x, s(c), len(c)
Result: x classified into context
usedSensors = [];
if sleepTime > 0 then
    // sleeping phase of the duty cycle
    sleepTime = sleepTime − 1;
    return context;
else if sleepTime == 0 then
    // sleeping phase finished, restore all sensors
    for s ∈ currentSensors do
        turnOn(s);
    end
    sleepTime = sleepTime − 1;
node = decisionTree.root;
// traverse the tree as long as the required data is available
while node is not leaf & requiredSensor(node) ∈ activeSensors do
    usedSensors.add(requiredSensor(node));
    node = the appropriate child node given x;
end
// classify x using the current node, regardless if it is the leaf
   node or not
previousContext = context;
context = node.classify(x);
if node is leaf then
    // choose the new decision tree
    decisionTree = s(context);
    if context != previousContext then
        // if tree was changed, different sensors are needed
        currentSensors = [requiredSensor(decisionTree.root)];
        usedSensors = currentSensors;
    // turn off all sensors and start the sleeping phase
    for s ∈ currentSensors do
        turnOff(s);
    end
    sleepTime = len(context);
else
    // additional sensor is needed
    turnOn(requiredSensor(node));
    currentSensors.add(requiredSensor(node));
    usedSensors.add(requiredSensor(node));
end
// remove all not needed sensors
for s ∈ currentSensors do
    if s ∉ usedSensors then
        turnOff(s);
end
return context;
```

Chapter 4

Datasets

The presented methodology was tested on four different real-life datasets and on a family of artificial datasets. They cover a variety of context-recognition tasks from the recognition of basic human activities (walking, running, resting, etc.), high-level activities (work, home, etc.) to very specialized context-recognition tasks (eating, using the fork, etc.) The artificial datasets were entirely computer generated and each of them was designed to demonstrate some basic properties and behaviours of our methods. The four real-life datasets were collected by attaching various sensors to volunteers and/or their environments. The author of this work participated in the collection of the two of these datasets: E-Gibalec and Commodity12. In both cases the datasets were collected as part of different national [92] or European [93] projects. The remaining two datasets – the Opportunity and SHL – are publicly available and often used for comparing different methods for context recognition.

This section covers each of the datasets in enough detail for the reader to understand their properties and be able to interpret the solutions returned by the presented methods. Each real-life dataset also references a paper with a more detailed description. In each case we also specify which sensing settings could be modified in order to improve the energy-efficiency of the context-recognition method used for that dataset and how we estimated the energy requirements of these modifications.

4.1 Artificial Datasets

To test our methodology under ideal conditions – where the assumption of the Markov property holds – we created different artificial datasets. As such, they can be used as a baseline test for the proposed mathematical models (Section 3.1.1.2 and Section 3.2.1) as we expect these models to make almost perfect predictions on these datasets. These datasets also provide a clear and convenient way to demonstrate the strengths and weaknesses of the proposed methods (SCA, DCA, CS-DT). Different experiments presented in Section 5.1 use different artificial datasets, choosing the ones that best demonstrate the properties of the tested method.

Given a matrix of transitions (Section 2.1), one can easily generate a random, arbitrary long string of sequential contexts that has the given transition probabilities from one context to another. In all cases we generated a sequence of length 100000. The number of different contexts is implicitly given by the size of the matrix. In some cases we handcrafted the transition matrix, in others we generated it randomly. We generated a total of 11 different artificial datasets, which can be split into two categories.

The first category of datasets was designed specifically to test the property of the DCA method (Section 3.2). When testing this method, the sensor data does not need to be

	Average duration															
Set1	20	20														
Set2	20	20	20	20												
Set3	20	20	20	20	20	20	20	20								
Set4	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
Set5	100	10														
Set6	3	4	1	6	2	5										
Set7	20	10	5													
Set8	20	20	5													
Set9	20	3	12	5	14	19	10	1								

Table 4.1: The average duration (number of instances) of each context for the artificial datasets 1-9. The number of columns in each row implies the number of contexts in that dataset. As an additional note, in most datasets the contexts have the same transition probability to every other context. Exceptions to this rule are the Set6, where the transition matrix was randomly generated; in Set8 only the second context can transition to the third. In the set9 the transition matrix is crafted so the transition probabilities are as different among themselves as possible.

generated – only the context sequence. Instead of creating a *real* classifier, we simply provided a randomly generated confusion matrix and then created the classifications in accordance with the probabilities specified in the confusion matrix. Nine different sets were generated this way and their summary is listed in Table 4.1. The first four were designed to test the effect of different number of contexts, the next four the effect of different transition probabilities between contexts. The last was designed to have many contexts with different properties.

On the other hand, for testing the CS-DT method (Section 3.3) the sensor readings were required and we thus generated them after generating the context sequence. We defined 11 different contexts and 6 different sensors. The contexts represented different activities or modes of transportation (*still*, *walking slow*, *walking fast*, *running*, *running fast*, *subway*, *car*, *car in traffic*, *home*, *work*, *being close to workplace*). The sensors had a different distribution of output values, depending on the context. They were roughly modeled after real sensors: GPS, Wi-Fi, accelerometer, gyroscope, etc., and the main objective in choosing their generated values was that each sensor would be good at distinguishing a small subset of contexts, but not at distinguishing all of them. Each sensor was assigned an energy cost between 2 and 15 (of an abstract unit), with the total sum of all sensors being 44. In deciding the energy costs the main objective was their variety and the usefulness of the corresponding sensors and not real-world realism.

We generated two similar datasets with the previously defined set of sensors and contexts. The first one, named *Full*, was comprised of all 11 different contexts. The second one, named *Hidden*, was derived from the first by joining different contexts together. As an example: *walking slow* and *walking fast* were joined into *walking*. The *SCA* method tries to optimize the active sensors given the current context; by having less granularity in the recognized contexts, the method should have less space for optimization and could be more easily outperformed by the CS-DT method that can optimize based on the data. Returning to the previous example, *walking slow* could be recognized using one sensor. While *walking fast*, however, the system needs another sensor to distinguish it from *running*. The *SCA* method could either always use the extra sensor when *walking* (consuming more battery), or never use it (reducing the overall accuracy). The *CS-DT* method, on the other hand, could alternate between using the sensor or not, based on the received sensor data. Since the sensor output distribution is different in different parts of an activity, we could

interpret it as some activities being composed of some number of hidden sub-activities, hence the dataset name. It is of note that joining the activities together also created some non-Markovian transitions between them.

The SCA method (Section 3.1) could in theory be tested on the first nine artificial datasets that do not have the generated sensor data. This would, however, require to manually create a custom confusion matrix for each sensor setting. We thus decided to test this method on the *Full* and *Hidden* datasets instead, using sensor subsets as possible settings.

4.2 SHL Challenge

The SHL dataset is one of the largest activity recognition datasets, suitable for a wide range of studies in fields such as transportation recognition, mobility pattern mining, localization, tracking and sensor fusion [94]. It was recorded over a period of 7 months by 3 participants engaging in 8 different activities in a real-life setting in the United Kingdom. A subset of this dataset was used for the SHL Challenge and was subsequently used throughout this thesis. The dataset for the SHL Challenge is available on the challenge website [95] and is described in detail in the baseline paper [96]. Our submissions to the SHL challenge won both the first [97] and the second place [34].

All of the data was provided by a single smartphone, worn by the same user in their trousers' pocket. The data came in five different modalities: acceleration, magnetic field, angular velocity (measured by gyroscope), air pressure and the phone's current orientation. The acceleration was broken down into two additional data streams: the acceleration as the result of gravity, and the acceleration without the gravity component (linear acceleration). Notably, sensors that could identify the user's location – GPS, Wi-Fi, cell network – were not included. The data was sampled using a 100 Hz sampling rate for all the sensors. All sensor readings and activity labels coincided at the same timestamps.

Altogether, there were 82 recorded days (62 for training, 20 for testing). This provided 16310 minutes, or roughly 272 hours, of data. There were 8 activities of interest: *still*, *walk*, *run*, *bike*, *car*, *bus*, *train*, *subway*. The activity distribution was mostly uniform, with the exception of the *run* activity, which was (understandably) under-represented.

Example samples of the acceleration and magnetic field data are given in Figure 4.1. Both exhibit a periodic and distinct pattern with different magnitudes and frequencies when the user is either walking, running or cycling. Acceleration data also captures the noise present when driving in vehicles. In addition, the magnetic field sometimes wildly oscillates while the user is in the train or subway, which can be potentially exploited for recognizing them.

The full pipeline of our system can be found in our previous work [97]. In summary, we first downsampled the data to 50 Hz to make the system more energy efficient, while retaining the classification accuracy. Second, we derived additional sensor streams from the original ones. These were the magnitudes, orientation in *(roll, pitch, yaw)* format and original sensor streams rotated into (North-East-Down) coordinate system [98].

Next, the data was split into 60-second non-overlapping windows. The same features were calculated separately for each sensor stream (and separately for each axis of a sensor in case of 3 and 4-axial sensors), with the exception of a few specialized features that were calculated for either only acceleration, angular velocity or orientation data. In aggregate, a total of 1,696 features were computed and used in the subsequent steps. They can be split into frequency-domain features and time-domain features.

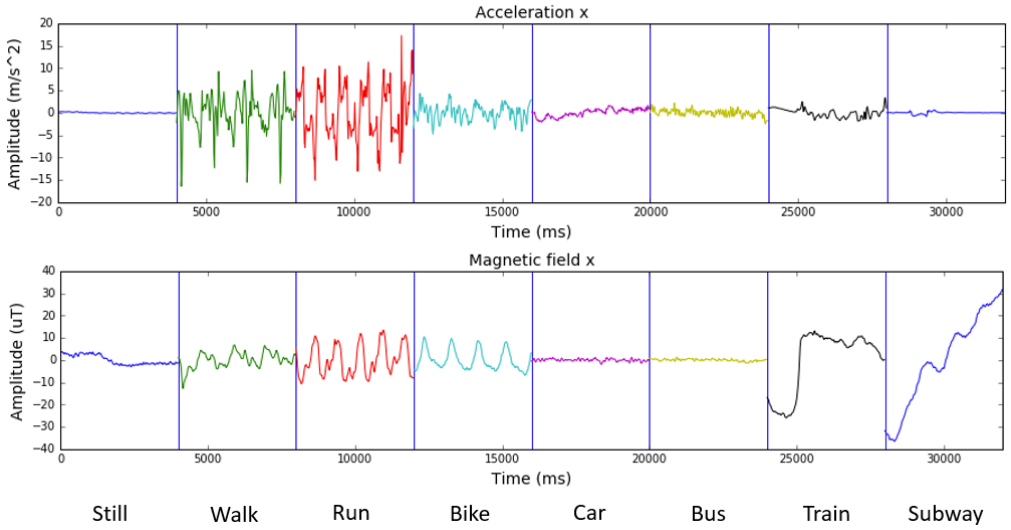

Figure 4.1: Acceleration and magnetic field data samples (both taken from the x-axis). A four-second sample is shown for each activity.

Frequency-domain features. These features were calculated using the power spectral density (PSD) of the signal, which is based on the fast Fourier transform (FFT). They are: three peaks with the largest magnitude from the PSD, energy calculated as the sum of the squared FFT component magnitudes, information entropy of the normalized FFT component magnitudes, a ten bin normalized histogram, skewness and kurtosis.

Time-domain features. A description and analysis of the expert features can be found in our previous paper [39]. In summary, the magnitude data stream provided the information on the intensity of the activity, while the individual axes provided the information on the orientation of the device and subsequently on the position of the user. Some features come from statistics and describe the intensity and "shape" of the signal: the mean, variance, Pearson's correlation between axes, their covariance, skewness, kurtosis, quartile values and range between them. Others have a more physics-based interpretation, such as velocity and kinetic energy. The rest came from expert knowledge of the domain: the number and height of peaks in the window, the signal's mean, its sum and squared sum, and the number of times the signal crosses its mean value. In addition we included the signal minimum, maximum, the number of times the signal is above/below its mean, the signal's mean change and its different autocorrelations (correlations of the signal with a delayed version of itself, for different delays).

The classification was performed using an ensemble of classifiers: Decision Tree [19], Random Forest [20], Naïve Bayes [99], K-Nearest Neighbours [21], SVM [100], Bagging - using Decision Trees [101], Adaptive Boosting, Extreme Gradient Boosting, Multilayer Perceptron [102] and a deep learning model. In this thesis we used this ensemble without the deep learning model, as it can be impractical to use on a smartphone.

The recording scenario of the SHL dataset was relatively natural, with the test subject moving around the city for several days (as opposed to the scenarios typically performed and recorded in a lab). This resulted in activities that on average lasted a long time (Table 4.2), with sensible transitions from one to another (Figure 4.2).

As the system setting to be optimized, we took all possible sensor subsets in addition to different duty-cycle lengths. The energy cost function was defined as the number of

Activity	Still	Walk	Run	Bike	Car	Bus	Train	Subway
Duration [mins]	14	12	12	20	60	34	64	34

Table 4.2: Average duration of each activity in the SHL dataset.

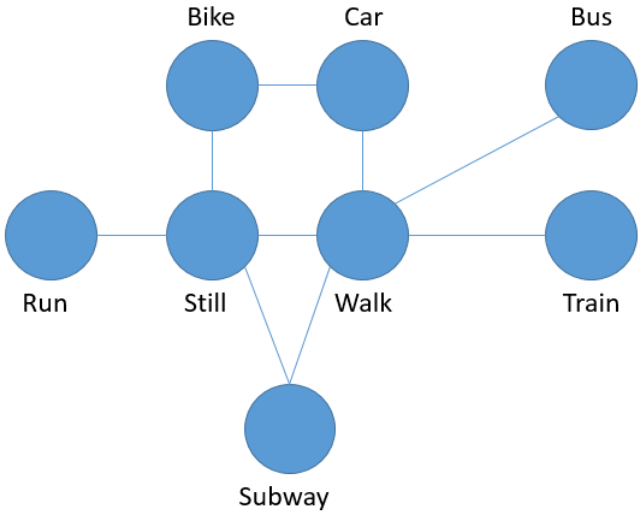

Figure 4.2: All activities in the SHL dataset, connected if the probability of a transition from one to another is more than 30%.

sensors active. This function is overly simplistic as the energy consumption details were not given by the dataset authors. Note, however, that the presented methodology could work just as well with any other complex cost function.

4.3 Opportunity

Opportunity [103] is a popular, publicly available dataset devised to benchmark human activity recognition algorithms. Four users were recorded while performing different tasks in an apartment.

In that apartment, there were 30 sensor clusters, some on the user's body, and some on objects that the user interacted with. The full list of sensor locations: user's left knee, hips, left and right upper arm, left and right lower arm, his hips, left and right shoe, left and right wrist, left and right hand in addition to being on a cup, salami, water bottle, cheese, bread, on two different knives, sugar, plate and on the drinking glass. Each cluster contains some of the following sensors: accelerometer, gyroscope and magnetometer.

The dataset provides different sets of labels, out of which we decided to test on the case where the problem was to recognize which object is the user currently holding in their right hand. There were 24 classes: *bottle*, *bread*, *chair*, *cheese*, *cup*, *dishwasher*, *door1*, *door2*, *drawer1*, *drawer2*, *drawer3*, *fridge*, *glass*, *knife*, *lazychair*, *milk*, *plate*, *salami*, *salami2*, *spoon*, *sugar*, *switch*, *table*, *none*), each representing an object held, except for the *none* class, which represented no object in hand. It is important to note that the class distribution was highly unbalanced and that the *none* class had a representation of 57%. Some of the objects were indistinguishable using only the described sensors (opening *door1* is the same as opening *door2* from the point of view of the accelerometer). We joined such

classes – both doors into one class, the same for the three drawers, both salamis, both chairs and the fridge with the dishwasher – which resulted in a set of 18 classes.

The classification process was made relatively simple to match the introductory paper [73] of the dataset. Data from each sensor was split into 500-millisecond non-overlapping windows on which we calculated the mean and the standard deviation. K-Nearest neighbours classifier (k=3) was then used for classification. The *F-score* was used as the measure of the classification quality.

As the settings for the optimization of the energy consumption, we used the different sensor subsets and different duty-cycling lengths. Due to the unusually high number of sensors (30 sensor clusters) as well as the high number of contexts (up to 24) we used only some sensor subsets (not all, as usual) as the settings for our methods. These subsets were chosen in the following way: we started with an empty set. Then the sensor cluster that increased the *F-score* the most was added to set. This was repeated until adding no single sensor cluster could increase the *F-score*. Each of the resulting subsets was added as a sensor setting (each subset had one more sensor active than the previous). The procedure was then repeated for each context, this time adding sensors only if it increased the *F-score* for recognizing this context only. All subsets generated were again added as the sensor settings. The justification for this greedy procedure is that most sensor subsets are redundantly large and are inflating both the energy consumption and are needlessly increasing the search space of different system configurations. After this selection, each setting should represent an increase of *F-score* for at least one of the contexts. This resulted in a total of 159 subset settings.

While a more detailed energy consumption estimate is made for the Commodity12 dataset described in the next section, we assumed here for the sake of simplicity that all sensors present have a roughly similar energy consumption. To model their combined consumption we simply summed up the individual energy consumptions. The sum is – given the previous assumption – proportional to the number of sensors active. The final energy requirement of the system is therefore proportional to the average number of sensors used. An interesting alternative would be to consider that since sensors at different locations probably each have their own battery, we could determine which location consumes the most energy by being active the longest. Such location could be considered a bottleneck for our system. By trying to reduce the energy spent on the currently most power-hungry sensors, we would expect to get configurations that switch between the sensors as often as possible, reducing the load on each individual one. This metric was used in our past work on the topic [91].

4.4 Commodity12

The goal of the Commodity12 project was to create a system that would be used by diabetic patients – it would monitor their activities, helping them to manage their lifestyle more easily. All details can be found in our previous work [39][11][104][105][106].

As part of this project a dataset was collected. For the data collection, a smartphone and a chest-worn heart-rate monitor [107] were used to monitor 10 participants. Each participant captured continuous two weeks of data and hand-labeled the following contexts: *sleep*, *work*, *home*, *eating*, *transport*, *exercise*, *out* (out of house, but not in any of the previous contexts). The data came from ten different physical and virtual sensors: accelerometer, barometer, light sensor, GPS location, a list of visible WiFi networks, location description by the Foursquare web service, sound, time, heart rate and respiration rate. The first eight were measured with the smartphone, and the last two with the heart-rate monitor, which was connected to the smartphone via Bluetooth.

Features were calculated for each minute of the data, and one minute became one learning instance. Different features were used for different sensor streams. They are summarized as follows:

Barometer. Three features were extracted: minimum, maximum and average pressure within each one-minute window.

Light sensor. Three features were extracted: minimum, maximum and average light reading within each one-minute window.

Wi-Fi. The list of visible Wi-Fi networks was used to create a fixed number (20) of clusters independently for each user. The first feature is the availability of the Wi-Fi network. If the Wi-Fi is available, the second feature indicates to which cluster (if any) the current Wi-Fi list corresponds.

GPS. The GPS readings were used to create a fixed number (20) of clusters independently for each user. The first feature is the availability of the GPS sensor. If it is available, the second feature indicates to which cluster (if any) the current GPS location corresponds.

Foursquare. The Foursquare webservice can determine if the user's current location corresponds to one of the following categories: *Arts and Entertainment*, *College and University*,*Event*, *Food*, *Nightlife Spot*, *Outdoors and Recreation*, *Professional and Other Places*, *Residence*, *Shop and Service*, *Travel and Transport*

Acceleration. From the acceleration data, the elementary activities – *running*, *walking*, *standing*, *cycling*, *upright*, *sedentary*, *transition*, *other* – were first calculated using methods from our previous work [11]. The distribution of these elementary activities was then used as features (e.g., the percentage of time the user was walking). Similarly the distribution of low, medium and high user's energy expenditure was calculated and used.

Sound. To preserve the user's privacy, the sound from the smartphone microphone was recorded for 100 ms out of every second. Additionally, after each minute of sound recording there was a five-minute pause. Sound features are first computed in windows of size 20 ms and then averaged within each one-minute window. There were 89 sound features altogether, including the spectral-centroid, zero crossing, mel-frequency-cepstral-coefficient (MFCC), linear predictive-coding (LPC) and method-of-moments values.

Heart rate. Three features were extracted: minimum, maximum and average heart rate within each one-minute window.

Respiration rate. Three features were extracted: minimum, maximum and average respiration rate within each one-minute window.

Time. Two features were extracted: time in the day, and the day in the week.

Different testing scenarios were created by using varying degrees of personalization. In this work we recreated the highly personalized setting where the first week of data was used for training, while the second week was used for testing. This setting was chosen as we hoped it would produce interesting user-specific solutions. Out of the ten participants we selected four to test our methodology on. The criteria for the selection were that the selected user had data from all 10 sensors and that they engaged in at least 6 different

Activity	Work	Eating	Transport	Out	Home	Exercise	Sleep
User1	180	13	32	42	43	16	213
User2	286	23	25	60	98	108	253
User3	127	16	11	31	27	20	/
User4	235	36	42	35	159	/	212

Table 4.3: The average duration [min] of each activity for different users in the Commodity12 dataset.

activities at least once. The average length of the activities for each user is listed in Table 4.3. The duration in minutes corresponds to the number of instances, as each instance was made from a one-minute window. Testing the methodology on the data from different users allows us to compare the resulting solutions, which may differ due to the different habits of each individual.

Random Forest was determined to be the best performing classification model and was thus chosen to be used in this work on this dataset. The exception are the experiments when using the CS-DT method, where – naturally – the decision trees were used. While the classification accuracy was reasonably high (ranging from 73% to 88% depending on the user), the energy consumption of the application prevented it from seeing any practical use. In this work we attempted to remedy it by dynamically changing both the sensor subset that is in use and duty-cycling lengths (those two modalities were the possible settings for our system).

Predicting the energy consumption of a smartphone is not trivial. Due to sensors sharing hardware and various software optimizations, the cost of activating sensors does not add up linearly. For example, the cost of accelerometer and gyroscope being active is much lower than the sum of their individual costs.

To create a cost function, we did some simple measurements for the purpose of presenting our methodology, but an interested reader may consult other papers on how a smartphone's energy consumption works in more detail [51][108]. All our measurements were done on a Samsung Galaxy S4 device (without a SIM card), using a multimeter attached directly to the device's battery contacts. We measured the current, which is proportional to the energy consumption, given a relatively constant voltage.

We made the following simplifications: 1) the time "sensor" is free, 2) the heart rate and respiration are free as long as the Bluetooth is active, 3) the Foursquare web service is free if both GPS and Wi-Fi are active, 4) GPS added an extra 40 mA on average regardless of other sensors and 5) having one, two or all of the following sensors – accelerometer, light sensor and barometer – resulted in a very similar energy consumption, perhaps because they share many resources to operate. The remaining combinations are shown in Table 4.4.

4.5 E-Gibalec

E-Gibalec was a project aimed at encouraging a more active lifestyle in elementary-school children. The developed mobile application could detect the activities of its user, so they could be rewarded for their active lifestyle. It allowed the child to set its daily movement goals, compete with classmates, and keep upgrading their virtual avatar using the won virtual awards. The whole system is thoroughly described in our two previous works on the topic [109][110].

To learn the specific activity patterns of elementary-school children we recorded the E-Gibalec dataset. Ten volunteers (aged 10-12) performed a one-hour-long predetermined

Combination	Current	Combination	Current	Combination	Current
/	20	A (5 Hz)	24	A	46
B	45	W	30	S	55
A, B	52	B, W	75	B, S	68
A, W	70	A, S	61	W, S	84
A, B, W	74	A, B, S	70	B, W, S	90
A, W, S	85	A, B, W, S	100		

Table 4.4: Energy cost of different sensor combinations measured in mA. Sensors that were active were labeled A – accelerometer (50 Hz), A – accelerometer (5 Hz), S – sound, B – Bluetooth, W – WiFi.

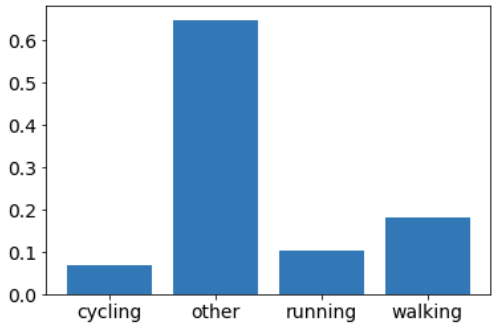

Figure 4.3: Activity distribution for the E-Gibalec dataset.

scenario, which included different physical activities (*lying*, *sitting*, *slow walking*, *fast walking*, *slow running*, *fast running*, *stretching*, *sit-ups*, *push-ups*, *jumps*, *dodgeball*, *basketball*, *volleyball*, *football*, *cycling*). While a detailed description of their activity was being annotated, the labels were later joined into four elementary activities: *walking*, *running*, *cycling*, *other*. The proportion of activities is given in Figure 4.3.

The participants in this experiment carried three smartphones in three different locations: in their trousers' pocket, in their jacket pocket and in their bag. These were chosen to cover the most often used locations for carrying the smartphone in that age group. However, only one smartphone (carried in any of the locations) would be used by the trained context recognition system.

The smartphones recorded only with the three-axial accelerometer, and did so with the sampling frequency of 50 Hz. Data was split into five-second non-overlapping windows, and features were calculated for each window. Each smartphone was independently generating its own data-stream and generated its own series of windows. These different windows series were then appended one after another.

We started with approximately 60 features from our previous work [104], from which we removed all computationally expensive ones, and all that depended on the phone's orientation (so we were not required to compute it). The rest was run through a feature selection procedure that was iteratively adding features to the final set if doing so increased the classification accuracy. The final five features were the average of magnitudes, the value of the third quartile, the variance, the coefficient of variation (Equation 4.1), and the number of peaks.

$$\text{coefficient of variation}(x) = \frac{std(x)}{mean(x)} \tag{4.1}$$

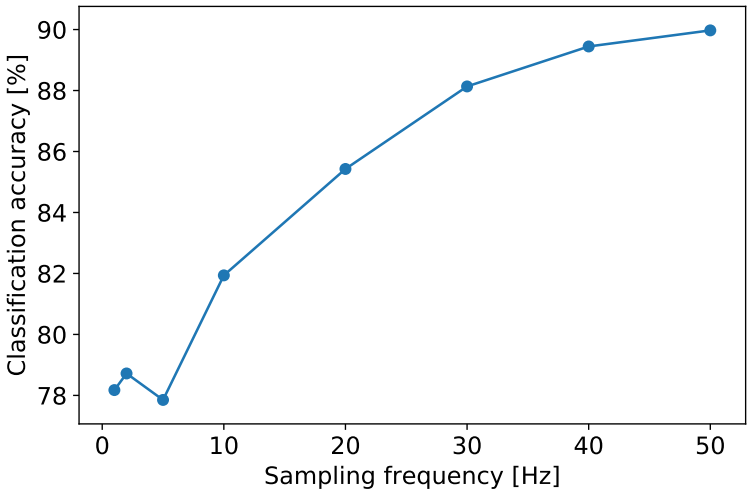

Figure 4.4: Classification accuracy in the E-Gibalec dataset using different sampling frequencies of the accelerometer.

In the experiments performed in this work, we used the first five volunteers for training, and the last five for testing. A decision tree model was used as the classification model, as we strived to make the classification process as computationally light as possible.

While the reduced computation time and using only one sensor already reduced the energy consumption, we additionally used the presented methodology to optimize two other system settings: the sampling frequency and duty-cycling.

Different sampling frequencies were created by downsampling the original data. Figure 4.4 shows how different sampling frequencies affect the accuracy of the classification. As expected, reducing the sampling frequency gradually reduces the classification accuracy – especially when the sampling frequency falls under 30 Hz. The under-performance of the classifier working at 5 Hz is probably a coincidence. For the optimization of frequencies we picked the set *1,5,10,20,50* as they are the most frequently available settings for smartphone sensors.

To determine the energy consumption of the defined settings we started with the empirical measurements that were made as part of the Commodity12 dataset (Section 4.4). From them we determined that the sampling frequency of 5 Hz needs 24 mA of current, while the frequency of 50 Hz needs 46 mA. For the rest of the sampling frequency values we linearly interpolated the energy consumption. This can be justified by the finding of the related work [26][27] that the energy consumption of the accelerometer device increases roughly linearly with its sampling frequency. For the duty-cycling settings we used the same formula as with every other dataset – multiplying the energy consumed when sensors are active with the proportion of time active and vice versa for the sleeping period (Equation 3.20). In this dataset the energy cost for the case without sensors is 20 mA.

When using our energy-reducing methodology on this dataset, a problem arises as the transitions from one activity to another are predetermined and same for all recorded users. This, however, does not reflect the actual real-life state and may cause the system to overfit to these artificial transitions (e.g., *running* was always followed by *cycling* which always transitioned into *other*). To partly alleviate the issue we first used the expert knowledge to assemble a sensible transition matrix between the activities and then shuffled the dataset with respect to these transition probabilities.

Chapter 5

Experiments

5.1 Artificial Datasets

Properties of these artificial datasets are known and can be controlled and changed. This allows us to understand how our proposed methodology works under ideal conditions and how its performance changes when changing some property of the dataset. We first tested optimizing energy consumption using the SCA method (Section 5.1.1), followed by the DCA method in Section 5.1.2. We then tested the CS-DT method (Section 5.1.2) and, finally, all three methods on the same version of the artificial dataset – both independently and combined (Section 5.1.4). This section also serves as a step by step description of all made experiments – which will then be repeated for each real-life dataset.

5.1.1 SCA Method Experiments

The sensor-to-context (SCA) method assigns a different setting to each context and uses that setting when the appropriate context is recognized (Section 3.1). It can predict the performance of each assignment as well as use multi-objective optimization to find the Pareto front approximation of different assignments – each being a trade-off between energy and classification quality. All the tests for these methods were performed on the *Full* and *Hidden* datasets described in Section 4.1. Both datasets are very similar, with the only difference being that some contexts in the *Full* dataset were merged into one for the *Hidden* dataset. While the former has the Markov property, the latter is not completely Markov – and contains some of the non-Markovian transitions that would be expected in a real-life dataset. In both cases the settings were the subsets of the sensors used. For each setting a different classifier was trained and then used when the setting was selected.

5.1.1.1 Evaluating the Accuracy of the Mathematical Model

We started with the evaluation of the prediction accuracy of the SCA model. We compared it to the simple model described in Section 3.1.1.1. For this experiment we generated 100 random assignments (an example in Table 5.1) and evaluated them using both mathematical models in addition to actually testing the assignments by simulating the sensor switching on the whole dataset. The simulation is taken as the ground truth – if the SCA model predicts that the system will work with 80% accuracy, but the simulation shows it works with 78% accuracy, then the model made a 2 percentage point (p.p.) error.

There was a concern that assignments found by the NSGA-II algorithm would exhibit a larger error – as being the best performing they could be outliers in some way – so the experiment was repeated using 100 assignments found by the NSGA-II algorithm (Section 5.1.1.2).

Still	Walking	Running	Subway	Car	Home	Work
A	G	A,G	W	W,A,G	G	G

Table 5.1: A random assignment made for the *Hidden* dataset. Whenever the context in the top row is detected, the sensors in the bottom row are activated. A-Accelerometer, G-GPS, W-Wi-Fi.

	SCA model		Simple model	
	Acc.	Eng.	Acc.	Eng.
Full - Random	0.05	0.1	6.1	1.5
Full - Pareto	0.04	0.1	12.4	1.7
Hidden - Random	1.5	0.2	8.0	1.4
Hidden - Pareto	1.7	0.1	8.2	2.1

Table 5.2: The difference (mean absolute error) between the predicted accuracy/energy and the one calculated in a simulation. The experiment was repeated for both the *Full* and *Hidden* datasets, for both Simple and SCA method and for both random assignments and assignments found by NSGA-II method. The difference in accuracy was measured in percentage points (p.p). The energy was defined in abstract units, with values ranging from 0 to 44 (the maximum value reached when using all sensors).

Both sets of experiments were done for both the *Full* and *Hidden* artificial datasets, with the results shown in Table 5.2. On the *Full* dataset the SCA model works basically flawlessly – and definitely much better than the simple model. On the *Hidden* dataset that does not have the Markov property, the prediction accuracy is lower, but again much higher than using the simple model. The prediction of the energy consumption followed the same trend, with the overall error being smaller even if adjusted for the value range.

To measure the time of evaluation, we again took 100 random assignments and evaluated them with the two mathematical models in addition to the simulation on the dataset. The results are shown in Table 5.3. It shows that the mathematical evaluation is more than 1000 times faster than the simulation for this dataset.

5.1.1.2 Using the Mathematical Model for Finding Good Solutions

We first demonstrated that using different sensors depending on the last classified context is better than always using the same sensor. For each sensor combination (6 sensors $\rightarrow 2^6 = 64$ combinations) we created an experiment where this combination was used to classify all instances. The Pareto front of the obtained trade-offs (*static solutions*) is easily obtained by any algorithm that can find a non-dominated set from a given set. This front for the *Full* dataset is presented in Figure 5.1. Alongside are plotted the solutions obtained by using the SCA method that uses a different sensor combination for each context. These solutions were generated using the NSGA-II algorithm with the SCA model being used as the surrogate model for the population evaluation.

	Simple	SCA	Simulation
Evaluation time [ms]	0.8	11	5234

Table 5.3: Time of evaluation of the mathematical models (both the SCA and Simple model) compared to the time of the simulation.

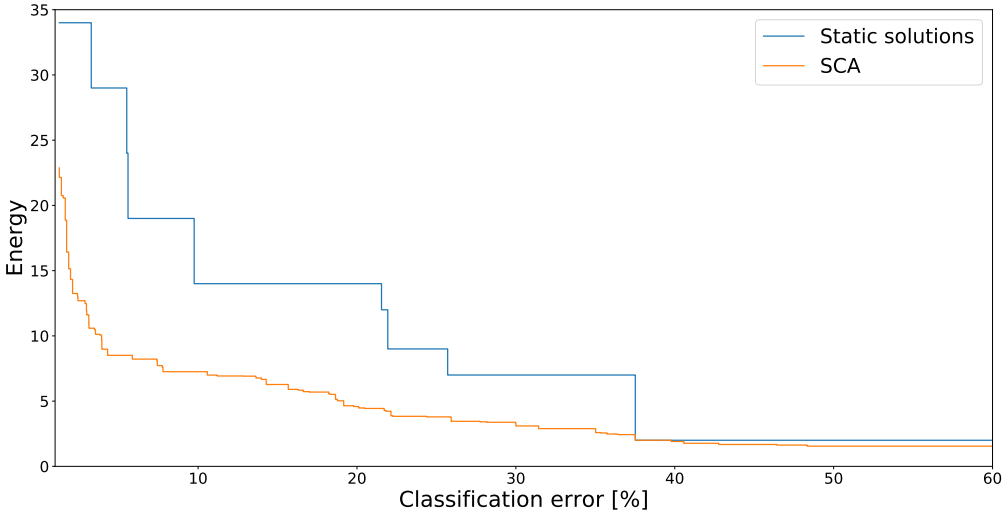

Figure 5.1: Comparison of assignments where all contexts have the same sensor assigned (Static solutions), to the assignments where active sensors dynamically change based on the last classified context (SCA). This comparison was done on the *Full* dataset.

The results show that the SCA method generates noticeably better trade-offs. As an example, perfect accuracy can be achieved consuming only 23 (out of the maximum of 44) units of energy. Alternatively, the accuracy of 95% can be achieved using only 8 units of energy. We will not delve into the semantics of the achieved solutions, given that the dataset is artificial – plenty of examples will be presented in the real dataset experiments.

To visually represent the accuracy of the SCA model predictions (that are numerically represented in Table 5.2), we re-evaluated all non-dominated trade-offs found by the SCA method, using both mathematical models as well as doing the real simulation. This was done for both the *Full* and *Hidden* datasets. These new sets of solutions are plotted in Figure 5.2. As expected, the SCA model matches the real simulated values better than the simple model.

While the predictions of the SCA model are better than using the simple model, we also tested if using the simple model instead of the SCA model generates worse trade-offs – as good trade-offs are what we are ultimately interested in. There are two ways of using the simple model: the first often used in related work [53] is to find for each context the sensor combination that is "most suited" for that context. According to the simple model, the "most suited" combination is the one with the greatest accuracy for recognizing the given context, when that context actually appears. The resulting assignment of sensor combinations to each context is depicted in Figure 5.3 next to the Pareto front approximation generated using the SCA model. This assignment is as good as others on the Pareto front. However, an obvious drawback of this method is that it produces a single point, instead of the whole array of different trade-offs. The second way of using the simple model is in combination with the NSGA-II algorithm in a similar manner as the SCA model is usually used. Unfortunately, the resulting Pareto front consisted of only one point. This is due to inaccurate predictions of the simple model, which in one case wrongly predicted very optimistic values for one assignment that in turn dominated all others. This assignment is also depicted in Figure 5.3.

Lastly, we tested the consistency of the found solutions by the SCA method. The search performed by the NSGA-II was repeated 30 times, each time running for 500 generations. It turned out that this dataset/method combination was easy to optimize, with all thirty runs converging to the same set of solutions after 100-150 generations.

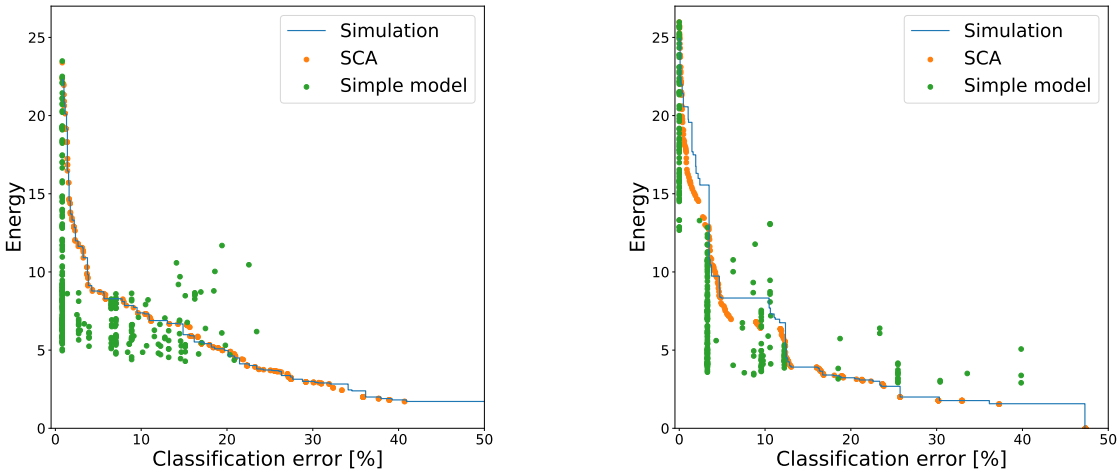

Figure 5.2: Assignments from Figure 5.1 evaluated with both mathematical models and compared to the values achieved by a simulation. This experiment was done for both the *Full* (left) and *Hidden* (right) dataset. On the *Full* dataset the SCA model makes almost perfect predictions while the Simple model only roughly hits the correct area. On the *Hidden* dataset, the SCA predictions were slightly less accurate, but still better than the Simple method predictions, which form curious clusters around a few different classification accuracies.

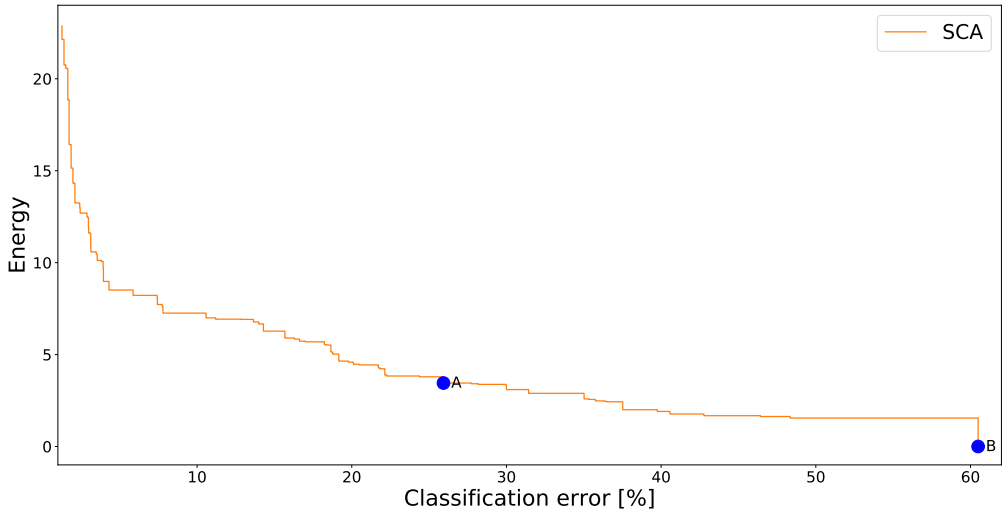

Figure 5.3: Two solutions found using the simple model (blue dots), compared to all solutions found with the combination of SCA model and the NSGA-II algorithm (*Full* dataset was used). The solution A was found by assigning a setting for each context in isolation, while the solution B is the only solution returned by NSGA-II algorithm that uses the simple model for evaluation. The assignment B uses no sensors for the most common context; this results in a system that eventually reaches that context and is then "stuck" using a majority classifier, classifying into that context over and over again. The simple model cannot accurately model this situation and thus over-optimistically predicts the performance of assignment B.

5.1.1.3 Two-Step Classification

Section 3.1.1.3 presents a different system that uses a two-step classification. Each context has a specialized classifier that detects if context transition happened or not. If the transition happens, then the current instance is marked as *other* and a general classifier is used to classify the new instance into a new context. A similar methodology to SCA can be used on such a system to generate energy-efficient solutions. We compared the SCA solutions found with the original system with the solutions found on the modified system and depicted both sets in Figure 5.4.

The problem this modified system faces on this dataset is the high frequency of context transitions – the change of a transition happens with the average probability of 14%. Since every instance that is detected as a transition is classified as *other* and is thus technically a misclassification, the system cannot hope to achieve an accuracy of more than 86%. One could discard the *other* instances when calculating the accuracy or alternatively re-classify all such instances into the first class that happened after the transition. Doing any of the two results in solutions with the adjusted accuracy, also shown in Figure 5.4. Even so, they are outperformed by the original SCA solutions.

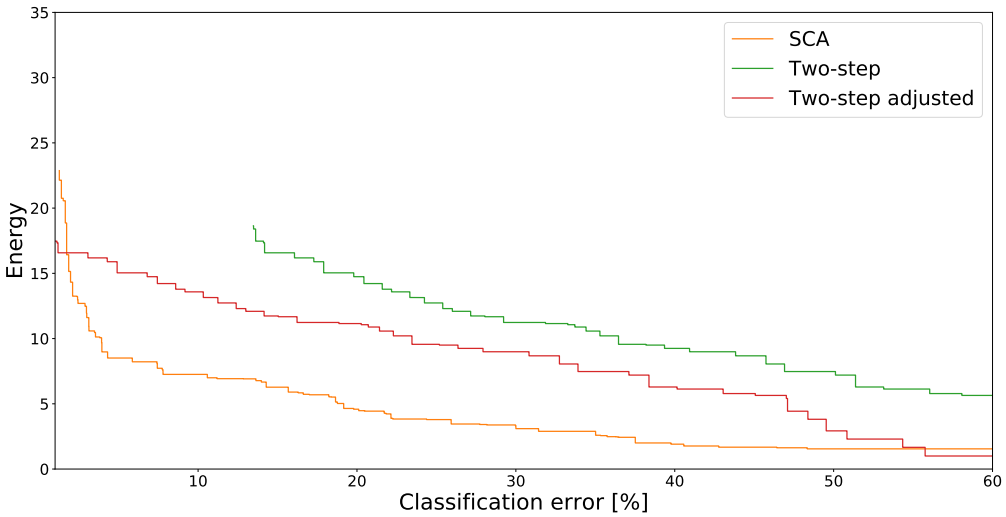

Figure 5.4: Solutions found when using two-step classification on the *Full* dataset. Solutions are compared to those that were achieved using the SCA method.

5.1.2 DCA Method Experiments

Duty-cycle assignment (DCA) method tries to assign a different duty-cycle length to each context and can evaluate the performance of the system that uses such an assignment – in terms of both the classification quality and energy savings. It then uses these predictions in conjunction with the NSGA-II algorithm to find good assignments. Testing this optimization does not require generating actual sensor data – just the context sequence. As a result we could easily generate many different datasets (Section 4.1) for an in-depth analysis of this method. Using them we tested the accuracy of the predictions and inspected the quality of the found assignments. In addition we showed how the context transition probabilities affect the type of assignments and how the DCA model was beneficial for finding them.

5.1.2.1 Evaluating the Accuracy of the Mathematical Models

Section 3.2.1 presents a way of predicting the accuracy of a system that uses duty-cycling. Alongside it presents three different simple approximations of the system performance. The first set of experiments shows that while the approximations can be decent under some simple conditions, their accuracy quickly falls in any complex scenario. It also shows that the proposed DCA model generates almost perfect predictions, and does so much faster than it could be achieved by simulating the system behaviour.

We started with a simple example, having two contexts that have a 0.05 probability of transitioning from one to another (*Set1*). On such a dataset we tested the performance of different systems that use different duty-cycle lengths (in all cases both contexts had the same length assigned). We assumed that the classifier can produce perfect predictions and all errors happen exclusively due to duty-cycling. We used both the DCA model as well as its approximations and compared the output predictions of the performance to actually running a simulation of the duty-cycling through the whole dataset. The simulation was used as the ground truth for the performance. The results are shown in Figure 5.5 and show a great match between the DCA model predictions and the actual performance. The approximations also display a high fidelity in this case, especially for short duty-cycle lengths. Approximation B was the best performing of the three in this case.

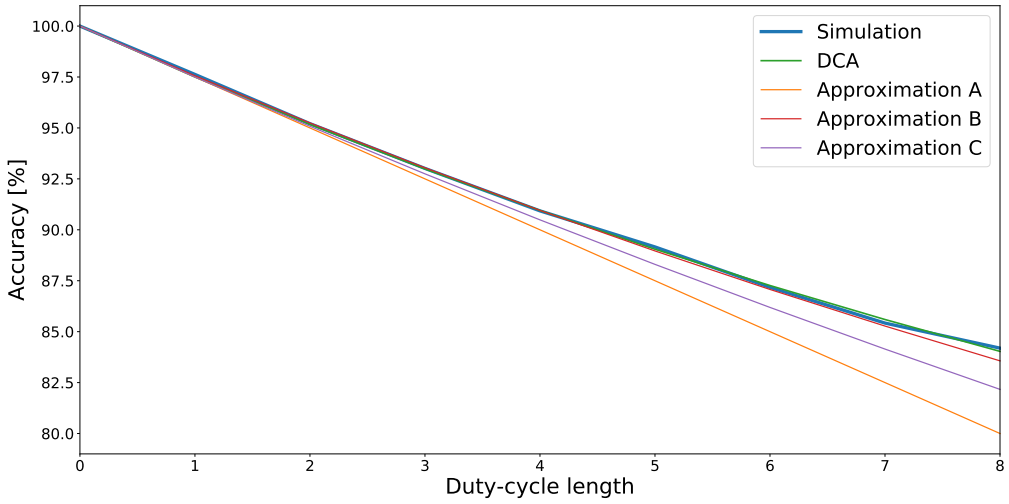

Figure 5.5: Predictions of the accuracy of the system when using different duty-cycle lengths on *Set 1*. Predictions are made with different mathematical models, as well as actually running the simulation. The duty-cycle length represents the number of periods spent sleeping. The length of the active phase w on current and similar figures is always 1 period, unless stated otherwise.

The performance of the simple approximations, however, breaks under many circumstances. Different scenarios are displayed in Figure 5.6. They vary from using long duty-cycle lengths (again using *Set1*), using a dataset with many contexts (*Set3*), datasets with contexts that have a different average duration (both *Set5* and *Set6*), using a random confusion matrix to simulate misclassifications (using *Set1* as the base) and having an active phase of length > 1 (*Set1*). The results show that the proposed DCA model consistently makes accurate predictions about the system performance, while other approximations do not. Note that in all these scenarios all contexts had the same assigned duty-cycle length – as the approximations cannot model having different lengths for different contexts.

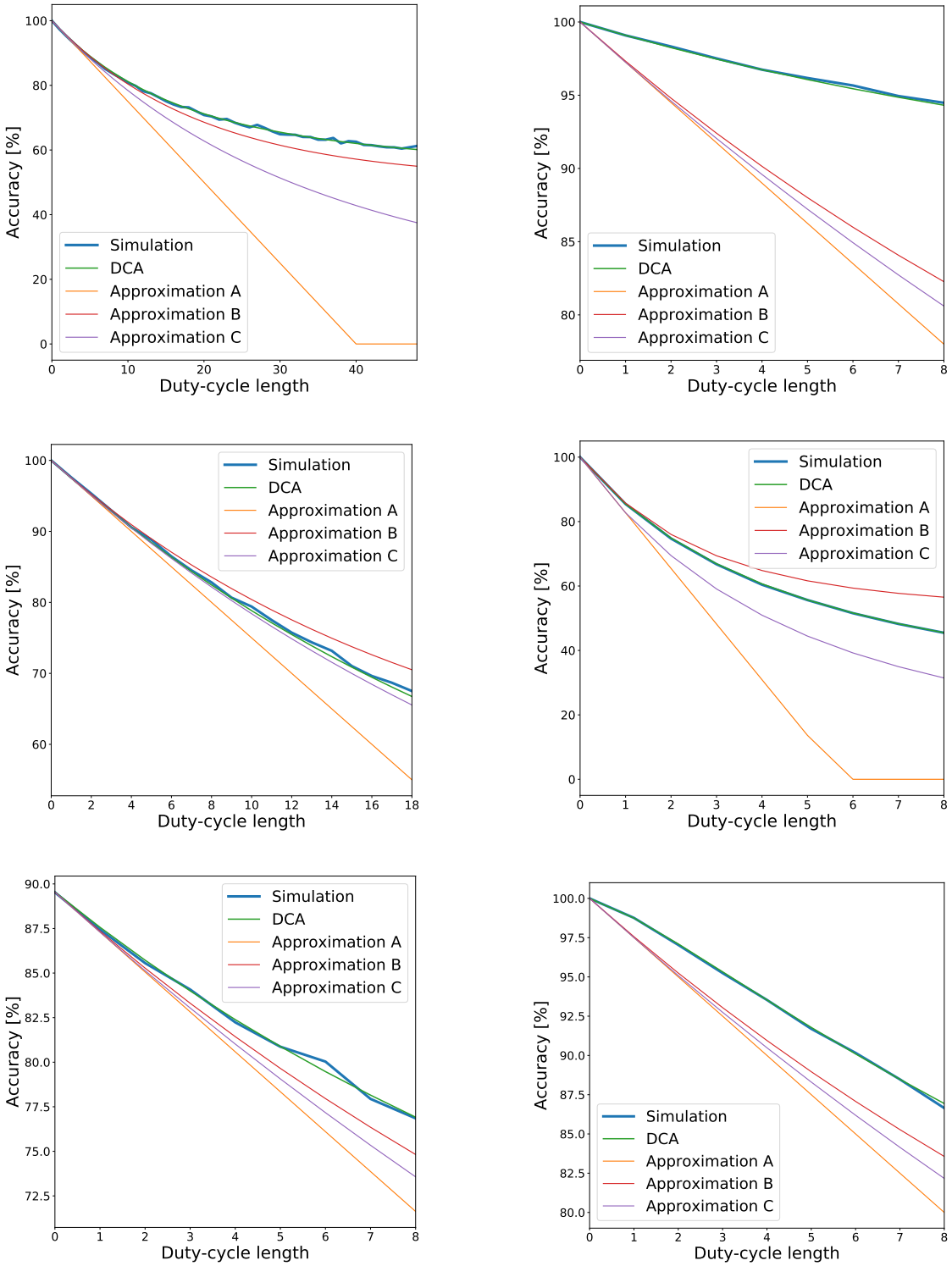

Figure 5.6: Predictions of the accuracy of the system when using different duty-cycle lengths. Different scenarios are tested: (top left) – duty-cycle lengths are long, (top right) – contexts have different average lengths, (middle left) – more than two contexts, (middle right) – more than two contexts and different average context lengths, (bottom left) – misclassification errors are present, (bottom right) – active phase is long.

	DCA	Approx. A	Approx. B	Approx. C
Accuracy [p.p.]	0.05	16.9	2.7	6.4

Table 5.4: MAE (measured in percentage points) between the predicted accuracy and one calculated in a simulation. DCA method clearly outperforms the simple approximations.

	Accuracy [p.p.]	Energy gain
Same lengths	0.05	0.0001
Different lengths	0.03	0.0053
Combined	0.04	0.0027

Table 5.5: MAE (measured in percentage points) for the DCA method between predicted accuracy/energy gain and one calculated in a simulation. Two sets of experiments were conducted – in the first the same duty-cycle length was assigned to all contexts, in the second the duty-cycle lengths varied based on the classified context.

To quantify the accuracy of the predictions we aggregated all the results from all the experiments and computed the MAE between values predicted by all 4 mathematical models and the values achieved by the simulation. The results are listed in Table 5.4.

All the experiments described so far were then repeated, but this time having different random duty-cycle lengths assigned to each context. This is a more realistic use case, as having different duty-cycle lengths is usually beneficial (and such assignments will thus be searched for by the DCA method). This time in addition to system accuracy, we also measured the predicted and actual energy gain of the system – a prediction that was trivial when the same duty-cycle length was always used. The results are listed in Table 5.5 and show that the DCA predictions are almost flawless in all scenarios.

The main advantage of using a mathematical model instead of a full simulation on a dataset is in the speed of the evaluation of an assignment. The speed of the DCA model evaluation mostly depends on the number of contexts used, as this number dictates the size of matrices and subsequently the cost of their multiplications. Measurements of the speed of the evaluation when using different number of contexts (Table 5.6) show that the mathematical evaluation can be a hundred times faster than the simulation. The speed, however, begins to quickly fall if the number of contexts increases so the use of this model is not recommended in the case of an extremely large number of contexts. While 311 ms for the full simulation does not seem much, one has to take into account that when searching for good assignments, one might have to evaluate tens of thousands of different assignments. Additionally, the simulation time is linearly dependent on the dataset size, while the time required by the mathematical model is not. Comparing this simulation time against the simulation time for the SCA method (Table 5.3), it is much shorter due to the actual classification not being needed in this case.

Number of contexts	2	4	8	16
Evaluation time [ms]	0.5	2	14	173
Evaluation time (opt.) [ms]	0.1	0.4	2	20
Simulation time [ms]	311	311	311	311

Table 5.6: The time of evaluation of the DCA model compared to the time of the simulation of duty-cycling on the whole dataset. The test was performed with different number of contexts (*Set1*–*Set4*), using both base and optimized implementation.

In Section 3.2.2.2 we noted that evaluating many assignments for the same dataset can greatly decrease the evaluation time for each individual one due to pre-storing partial results. This implementation was used for the NSGA-II search for this and all other datasets, and the average time of one assignment is listed in Table 5.6.

5.1.2.2 Using the Mathematical Model for Finding Good Solutions

Having a mathematical model that can accurately and quickly model arbitrary assignments allows us to efficiently search the space of different assignments. Using the NSGA-II algorithm alongside the DCA model, we found good assignments and compared them to assignments where all contexts have the same duty-cycle length. This test was conducted for three different scenarios. Lastly, we tested the quality of solutions depending on the available processing time for the NSGA-II algorithm.

In the first scenario we used the *Set7* which had three different contexts with different average lengths (5, 10, 20) with no classification errors. Intuitively, we can expect that the longer-on-average contexts would benefit from longer duty-cycles and reversely for the shorter ones. The accuracy/energy trade-offs acquired with the genetic algorithm are presented in Figure 5.7 alongside the trade-offs where all contexts are assigned the same duty-cycle length. There are two representations in the figure: in the first one we show the *Energy gain* – the average number of idle sensor cycles for each active one, while in the second one the actual spent energy, assuming the energy of an always active sensor is 1 and the energy of using no sensors is 0. The reason for showing both representations is to demonstrate what is happening as the duty-cycle length is increasing. The energy gain increases linearly, while the energy decreases asymptotically – the longer the duty-cycle lengths, the less impact each additional increase of length increase has. Note that in the second representation, the x-axis is flipped to be consistent with other figures (Figure 5.1, Figure 5.2, Figure 5.3). As a result, the ideal point in the first representation is top right, while for the second it is the bottom left. One can observe that the solutions found with the DCA method outperform the simple ones that used the same duty-cycle lengths for all contexts. Figure 5.8 shows the actual lengths in the assignments and confirms the hypothesis that longer contexts are assigned longer cycles.

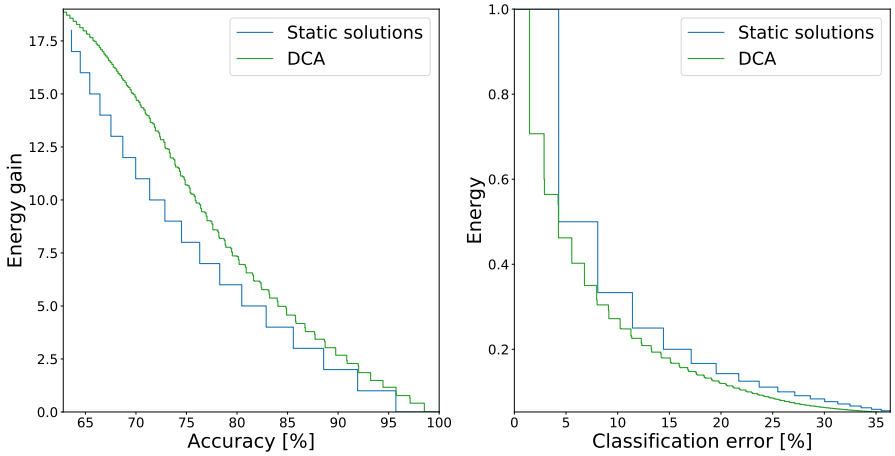

Figure 5.7: Comparison of the assignments found using DCA method to assignments where all contexts are assigned a duty-cycle of the same length (Static solutions). The resulting set of non-dominated solutions is shown in two different representations: (left) energy gain – the number of periods sleeping for each active one, (right) energy – assuming that an always active sensor has the energy cost of 1.

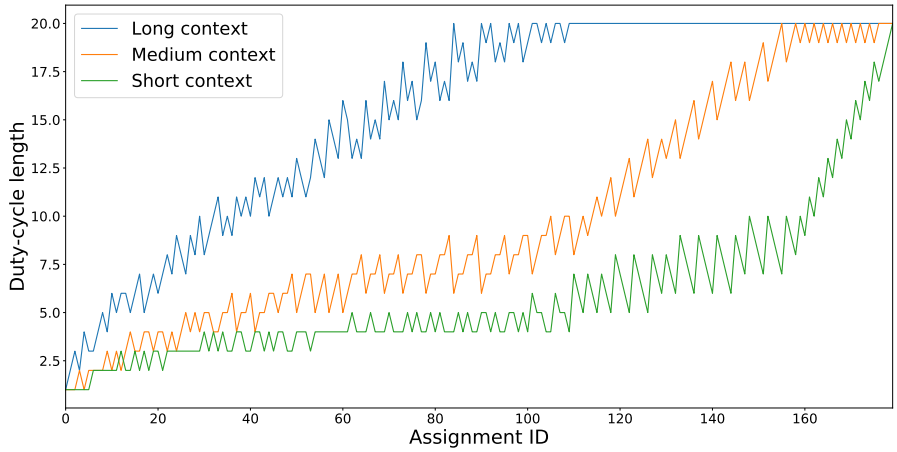

Figure 5.8: Assignments that correspond to the trade-offs in Figure 5.7 are represented here by the assigned duty-cycle length for each of the three different contexts. The order of IDs is the same as in the right part of Figure 5.7 – longer duty-cycles correspond to the solutions where the classification error is high, but energy is low. Flat line at the end of the long context length is a result of the search space being limited to the length of 20. While it is apparent that longer contexts have longer duty-cycles assigned, the ratio between the lengths varies.

In the next case, we used the same dataset (*Set7*), only this time we assumed that the misclassifications can happen with the probability of 20%. The results are shown in Figure 5.9. Adding the misclassifications seems to substantially increase the benefits of the search compared to using the simple solutions that use the same duty-cycle length.

In the third scenario we had three contexts (*Set8*) – two equally long ones and one short. However, only one of the longer contexts could transition into the short one. In this scenario we were interested only in the recall of the short context instead of the overall accuracy. The found assignments are shown in Figure 5.10 (right). Even though the average length of two contexts was the same, their suitable duty-cycle lengths were not. The reason is that the context that could transition into the short one had to have short a duty-cycle length, or else one cycle could skip the entire short context, drastically decreasing its recall. This demonstrates that the optimal duty-cycle length is connected not only to the expected length of the context, but also to the structure of expected context transitions. In addition, it demonstrates that the presented methodology can work with an almost arbitrary criterion, not only the classification accuracy.

Finally, to measure the performance of the NSGA-II algorithm that uses the DCA model, we chose a less trivial example featuring eight contexts with different average lengths and transition probabilities to each other (*Set9*). Misclassification rate was set to 10%. The search space was limited to the maximum duty-cycle length of 40 time periods. The search was repeated 30 times, each time running for 500 generations. After every 50 generations, the hypervolume indicator [7] was calculated. It was normalized so the best possible set of found solutions that maximizes both criteria (no such set exists in practice) would result in a hypervolume of 1, a and the worst set in a hypervolume of 0. The results of all 30 runs were averaged and shown in Figure 5.11. The results show that all of 500 generations were needed for finding the best solutions, but after 200 generations the solutions were mostly adequate and their performance increased only gradually.

The average search time was roughly 2 hours per run. Given that the most time

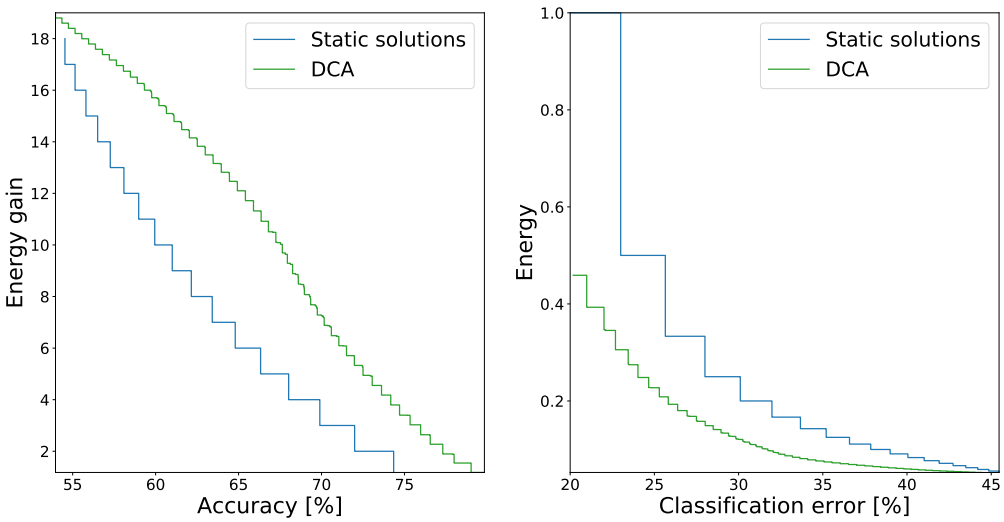

Figure 5.9: Comparison of the assignments found using DCA method to assignments where all contexts are assigned a duty-cycle of the same length (Static solutions). In this case *Set7* was used and miclassifications were assumed to happen, thus the accuracy is never 100%.

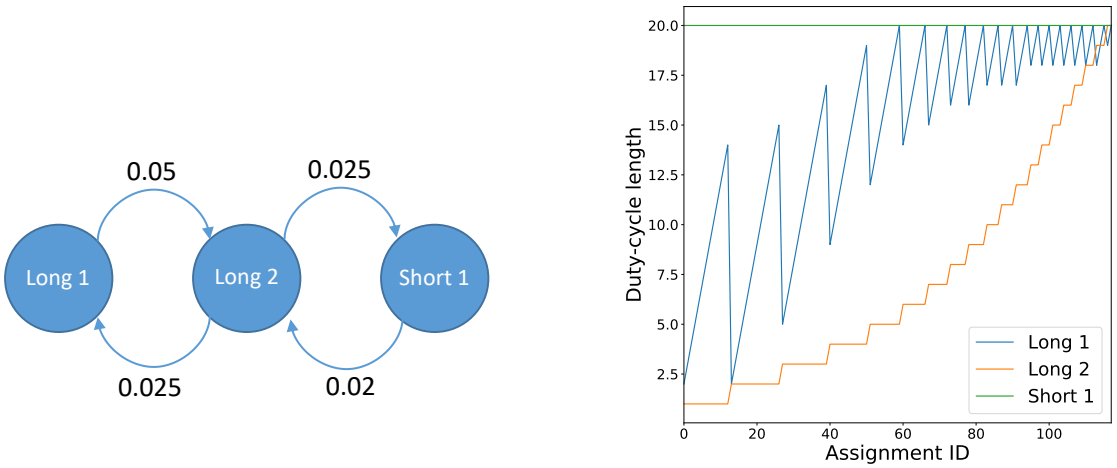

Figure 5.10: (left) The structure and the probability of the context transitions. The contexts *Long 1* and *Long 2* have the same average length, while the context *Short 1* is shorter on average. The optimization objective is the recall for the context *Short 1*. (right) Duty-cycle lengths for all three contexts, for all found non-dominated assignments. Context *Short 1* has always the maximal duty-cycle length assigned due to the nature of the optimization criterion.

consuming part of the search algorithm is the evaluation of the population fitness, one could extrapolate from Table 5.6 that using the full simulation instead of mathematical model would take roughly 100 hours per run. The latter is obviously less practical, demonstrating the benefit of the mathematical model.

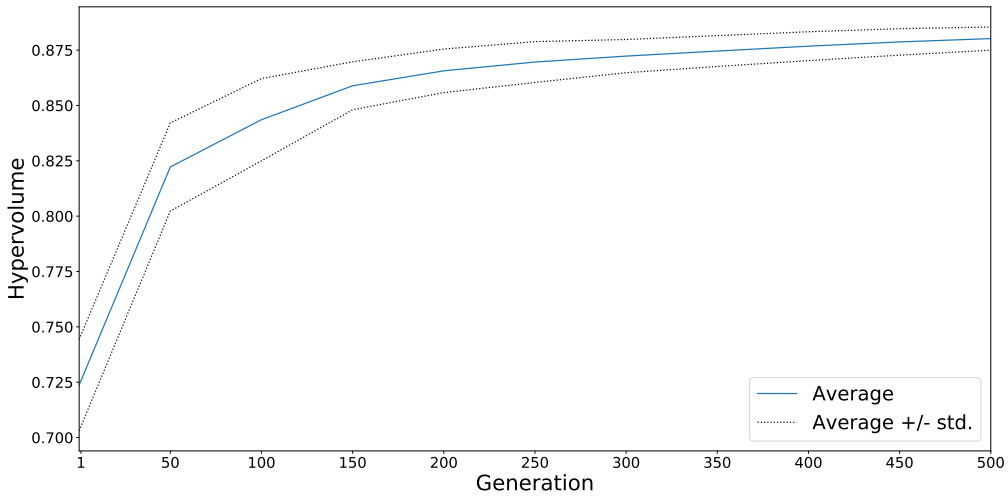

Figure 5.11: The mean of the hypervolume over 30 runs. The gray lines denote the mean +/- the standard deviation over the same 30 runs.

5.1.3 CS-DT Method Experiments

Cost-sensitive decision trees (CS-DT) are decision trees that were created by taking into consideration not only the energy cost, but also the classification accuracy. They can dynamically turn sensors on and off, depending on which branch of the tree was last used. For testing the CS-DTs we used only the *Hidden* artificial dataset. Since this dataset was created by merging some of the contexts from the *Full* dataset, it can be interpreted to have some hidden states. This states can change the distribution of the incoming sensor values, even if the visible context (as classified by the algorithm) remains the same. Cost-sensitive decision trees can capture this change by branching depending on the sensor values. This allows us to further (beyond what was possible using the SCA method alone) optimize the energy consumption.

We started by testing some properties that a cost-sensitive decision tree (CS-DT) should possess. We created a CS-DT where sensors had no cost (Figure 5.12–a). As expected, most sensors could be found in every branch. If we put a cost to the sensor at the top of the tree, that sensor sunk to the bottom of the tree (Figure 5.12–b). If we increased the cost of all sensors, the whole tree started to shrink – trading off its accuracy for its energy efficiency (Figure 5.12–c). The process ended with trees having only one node, becoming majority classifiers. The cost of the sensors can thus affect both the tree size and the tree structure.

After reverting to the same sensor costs that were used for the SCA method, we used the same iterative approach to create different CS-DTs that would present different trade-offs between energy consumption and classification accuracy. Note that the scaled cost was only used for tree generation and not for tree evaluation (where the original sensor cost was always used). The resulting trade-offs are depicted in Figure 5.13. As a note, Section 3.3.3.1 explains how the trees are generated using scaling of the misclassification cost not

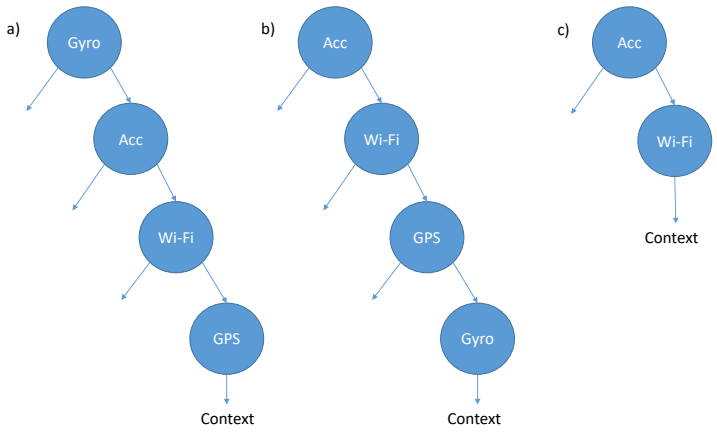

Figure 5.12: Cost-sensitive decision trees, built with different costs of sensors. The nodes represent features calculated by the data of the listed sensor. Note that sensor data was artificially generated for this dataset. Only one branch is listed for clarity.

the sensor cost – both processes, however, are symmetrical as only the relative size of one to another is of relevance.

A major benefit of using CS-DT is the dynamic switching of which sensors are currently in use. To test the effect of this sensor switching, we used the same trees as previously described and kept all the sensors present in the tree on all the time. This generated different trade-offs, which are plotted alongside the original ones in Figure 5.13.

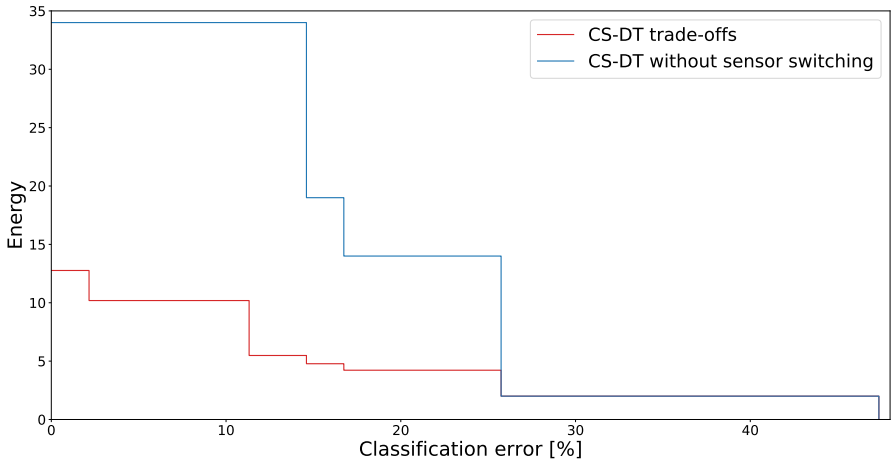

Figure 5.13: Trade-offs between energy consumption and classification accuracy achieved by different CS-DTs on the *Hidden* dataset. Alongside are the results of the same decision trees, modified so they do not employ sensor switching. A regular decision tree would perform the same as the most top-left CS tree without sensor switching.

5.1.4 Comparing and Combining methods

To compare the contributions of each of the three methods, we ran all three on the *Hidden* dataset. Figure 5.14 presents the trade-offs generated by each of them. Having all sensors active all the time is presented as a baseline. However, given that there exists a sensor subset that performs just as well as using all the sensors, that subset can be used as the baseline instead (marked *ideal sensor subset)*. This subset is also used for creating the starting classifier for the DCA method.

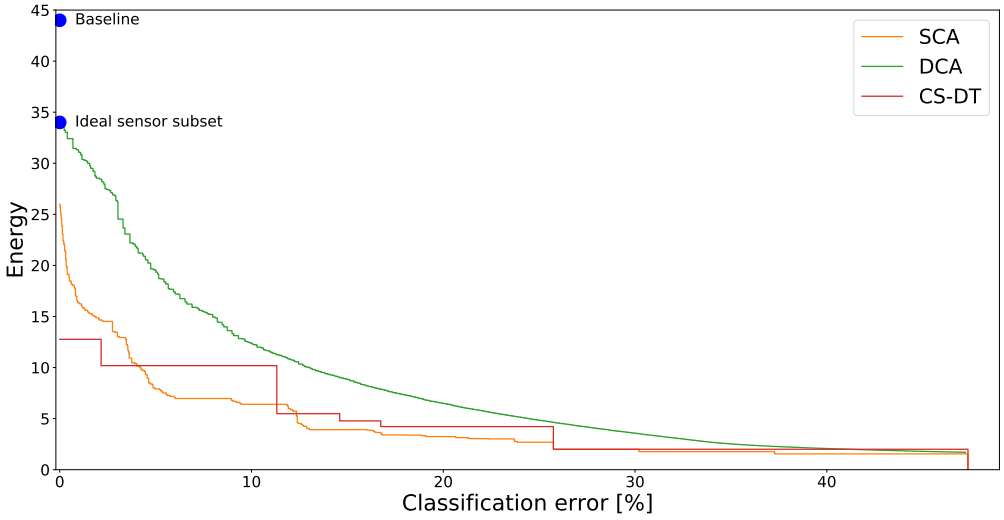

Figure 5.14: Comparison of trade-offs generated by different methods on the *Hidden* dataset.

All three methods generate a wide range of different trade-offs, with the SCA method being arguably the best on this dataset and DCA being the worst-performing one. The simple explanation is that duty-cycling works best when contexts are long on average, which is not the case for the *Hidden* dataset (avg. length = 7). It is interesting to note that both SCA and CS-DT find solutions that consume less energy than the baseline, while retaining the same accuracy.

To further increase the performance of the presented methodology, we combined the methods as described in Section 3.4. We tested the combination of the CS-DT method with the SCA method, and then the combination of the SCA method with the DCA method.

5.1.4.1 CS-DT and SCA Combination

As described in Section 3.4.2, the key insight is that one can use a different cost-sensitive decision tree for each context. To do so, first multiple different trees are generated, then the methodology from SCA is used to determine how an assignment of trees to each context affects the system performance. Finally, the genetic algorithm is used to find the best assignment.

There are multiple ways of generating a set of CS-DTs – four of them are presented in Section 3.3.3. Trade-offs from three sets are shown in Figure 5.15 (the binary trees were omitted as they cannot be used as stand-alone trees). All sets seem to be roughly comparable. Note, however, that at this point the same tree is used for the whole dataset.

Next, for each set of trees we independently run the SCA method to determine which tree to use for which context. We then compared the acquired trade-offs to trade-offs

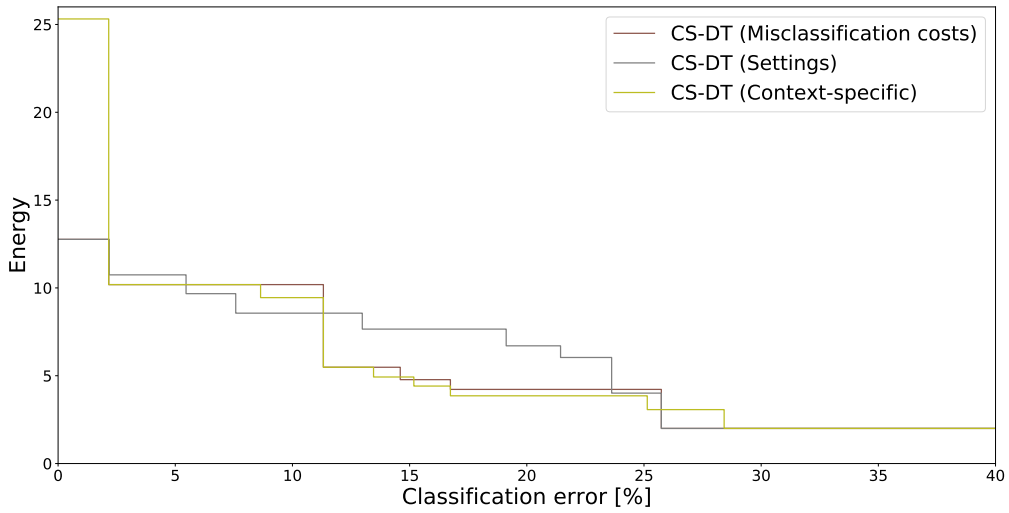

Figure 5.15: Comparison of trade-offs generated by different sets of cost-sensitive decision trees on the *Hidden* dataset.

that could be achieved by either SCA or CS-DT method alone. The results are shown in Figure 5.16 and show the benefits of the method combination. No single tree set proved to be clearly the best, with some being better at high accuracy solutions, others at low accuracy ones. The *misclassification cost* tree set (which we used as the default set in Figure 5.13 and 5.14) had a balanced distribution of trade-offs and was used for the final combination of all three methods. The *binary* tree set was consistently the worst performing one and it demands a different system architecture from all others, and was thus omitted in experiments on real-life datasets. The other three sets could theoretically be joined into one set before combining with the SCA method, but we wanted to test the individual contributions of each one.

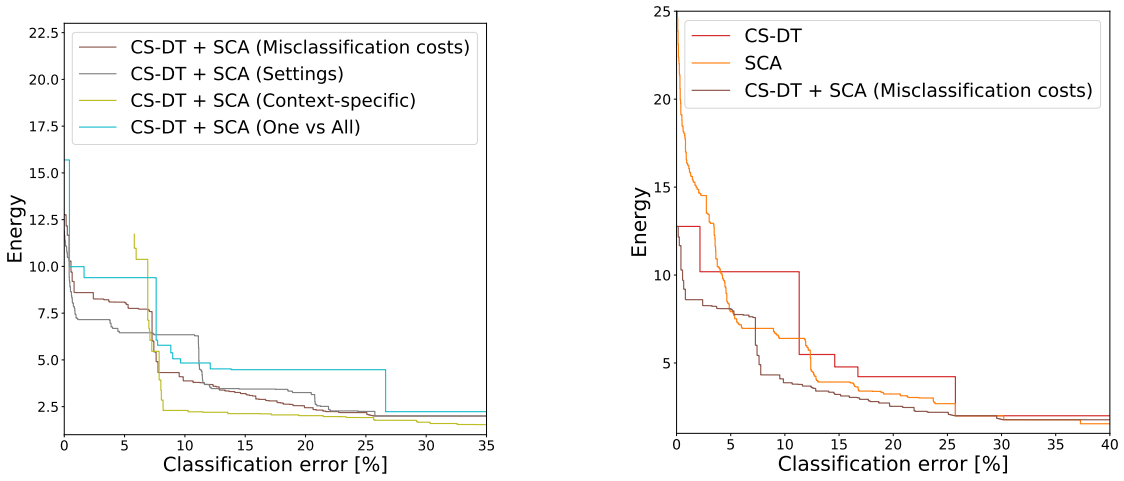

Figure 5.16: (Left) Comparison between different (started with different tree set) CS-DT and SCA methods combinations. (Right) Comparison between using CS-DT and SCA methods individually against using their combination.

5.1.4.2 SCA and DCA Combination

This method combination happens in two steps. First the SCA method generates a Pareto front approximation. Then one or more trade-offs from this front are selected for further optimization using the DCA method. This choice can be made by the system designer, but we opted here for an automated route. The first trade-off selected was the most accurate one, and every following trade-off was selected if it was at least 3% less accurate and consumed 3 units of energy less. This created a spread of chosen trade-offs, as marked in Figure 5.17.

The results do not show a dominance of the method combination, as it was the case in the CS-DT + SCA combination. Instead, the SCA method achieves better trade-offs at high system accuracy, but is then dominated by the SCA + DCA combination. This happens because many sensors are redundant for many contexts, and eliminating them is more effective than duty-cycling. Note, however, that even when the method combination works better than the SCA method, not all the best solutions were made by starting at the same trade-off. Trade-offs generated from Point 3 dominate the others for a while, but are later overtaken by trade-offs generated from Point 4. These results can be combined by joining the solutions from all experiments from Figure 5.17 and calculating the Pareto-optimal set from them. This set is shown in Figure 5.19 alongside other method combinations.

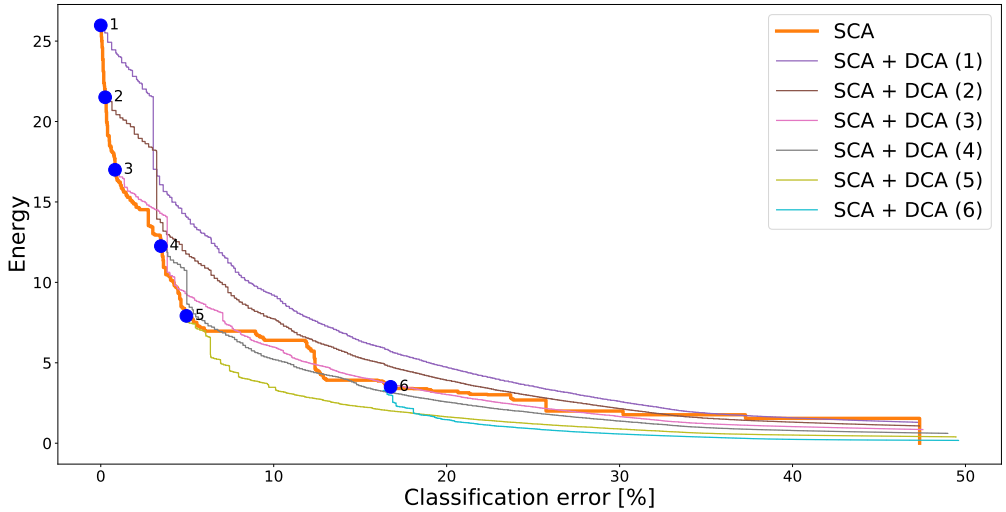

Figure 5.17: Comparison of trade-offs generated by applying the DCA method on the marked trade-offs (blue points) generated by the SCA method.

5.1.4.3 SCA, DCA and CS-DT Combination

The last step was to combine all three methods. As described in Section 3.4.3 we first started with trade-offs computed by the CS-DT + SCA combination, using the *misclassification cost* tree set (Figure 5.16) and then adding the duty-cycling using the DCA method.

The trade-offs expanded using the DCA method were automatically chosen using the same heuristic as before. The results are shown in Figure 5.18. From all sets of solutions shown in that figure, a Pareto-optimal set was calculated and used as the final solution set for this method combination.

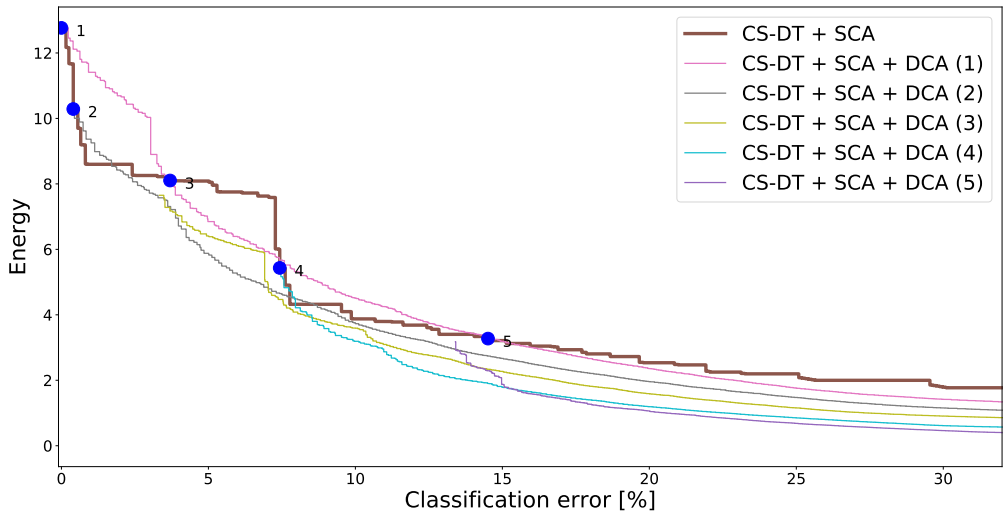

Figure 5.18: Comparison of trade-offs generated by applying the DCA method on the marked (blue points) trade-offs generated by the CS-DT + SCA method.

A comparison of all methods and all their combinations is given in Figure 5.22. Looking at the results it is clear that the presented methods greatly optimize the energy consumption of the system. One can also observe that method combinations generally outperform using them independently. Finally, the combination of all three methods is the best performing one.

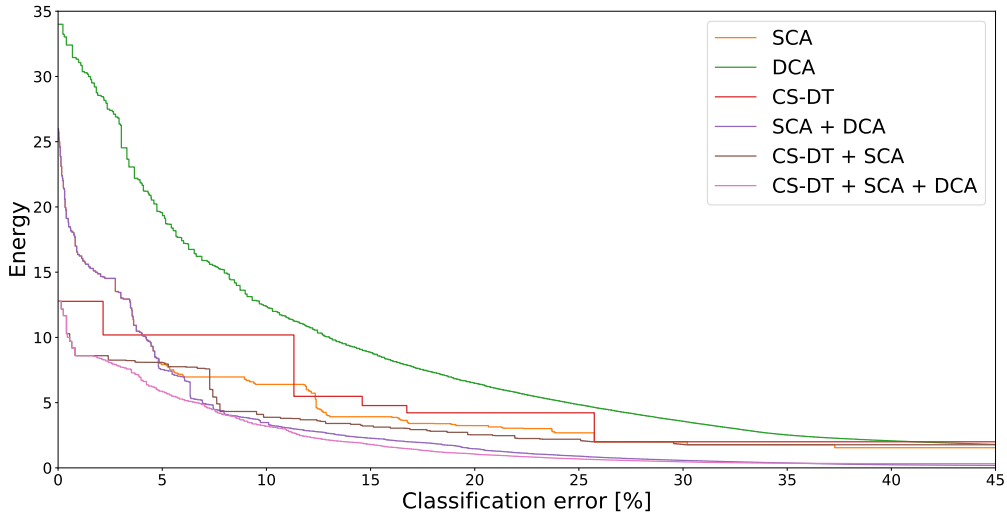

Figure 5.19: Comparison of trade-offs generated by all presented methods on the *Hidden* dataset.

In Section 3.4.3 it is described how the combination of all three methods can cause some performance issues. Some limitations on DCA parameters were thus introduced that mitigate the problem. Using these DCA limitations (the main one was using the active phase of length 6 – the number of sensor streams) we generated another version of the CS-DT, SCA and DCA method combination trade-offs. These trade-offs are shown in Figure 5.20. In this case, the combination of all methods dominates only some small parts of the solution space compared to other methods. This is due to the combination of short average activity lengths and a long active phase length – a combination of which diminishes the contribution of the DCA method.

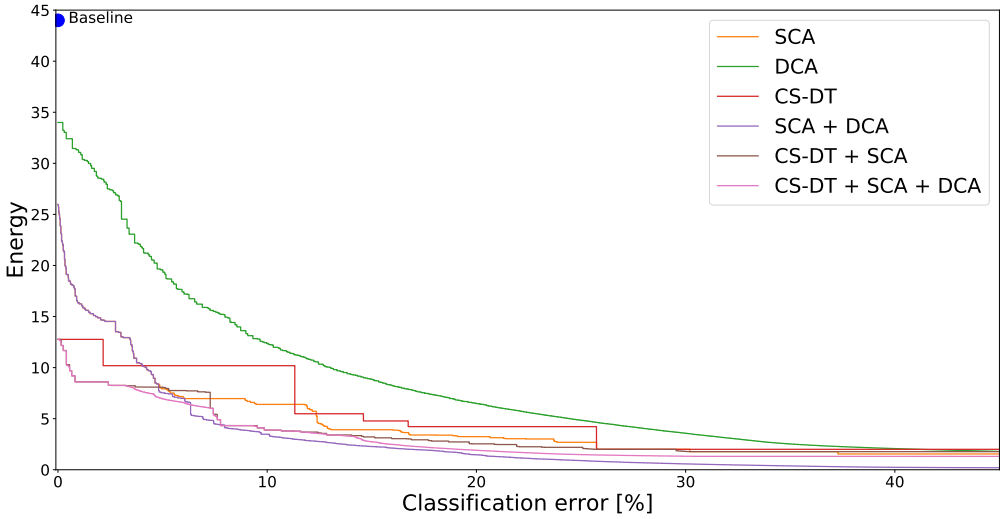

Figure 5.20: Comparison of trade-offs generated by all presented methods on the *Hidden* dataset. In this case of the CS-DT + SCA + DCA combination we used limitations on the DCA parameters.

Finally, we tested the accuracy of the method combination prediction. We have already tested each method independently, but one might wonder if combining these different mathematical models has any negative side-effect. We took all solutions from the CS-DT + SCA + DCA combination (Figure 5.20) and simulated running the system using these solutions. A comparison between the actual and predicted performance is shown in Figure 5.22. With the exception of some solutions around the 10% classification mark, the predictions almost perfectly match the actual performance.

Before we conclude with the artificial dataset, we show an example of a system that uses the combination of all three methods. The implementation of each one individually is relatively simple, but one might get confused on how they interact with one another. A simple excerpt is shown in Figure 5.21 (decision trees were simplified for clarity).

a) Assuming the last classified activity was *walking*, we start with using the CS-DT assigned to walking. This tree is in use for the whole active phase (which we set at 2 for this example). The tree first predicts *driving*, but no CS-DT change or sleeping happens during the active phase. b) The active phase gives the CS-DT enough time to reach a leaf node, and using it correctly classifies the activity into *walking*. b) Sleeping phase of the duty-cycle starts, no sensor is being used and all instances are classified into *walking*. In this example the length of this phase was set to 6. d) After starting the active phase the same CS-DT is used with the same sensors active as before the sleeping phase. During the second period of the active phase, *running* is classified. e) Six periods of sleeping. f) The

CS-DT to be used is now different (the one assigned to *running*). The only sensor active is the one assigned to the root of the tree and is different from any of the sensors used for the previous CS-DT.

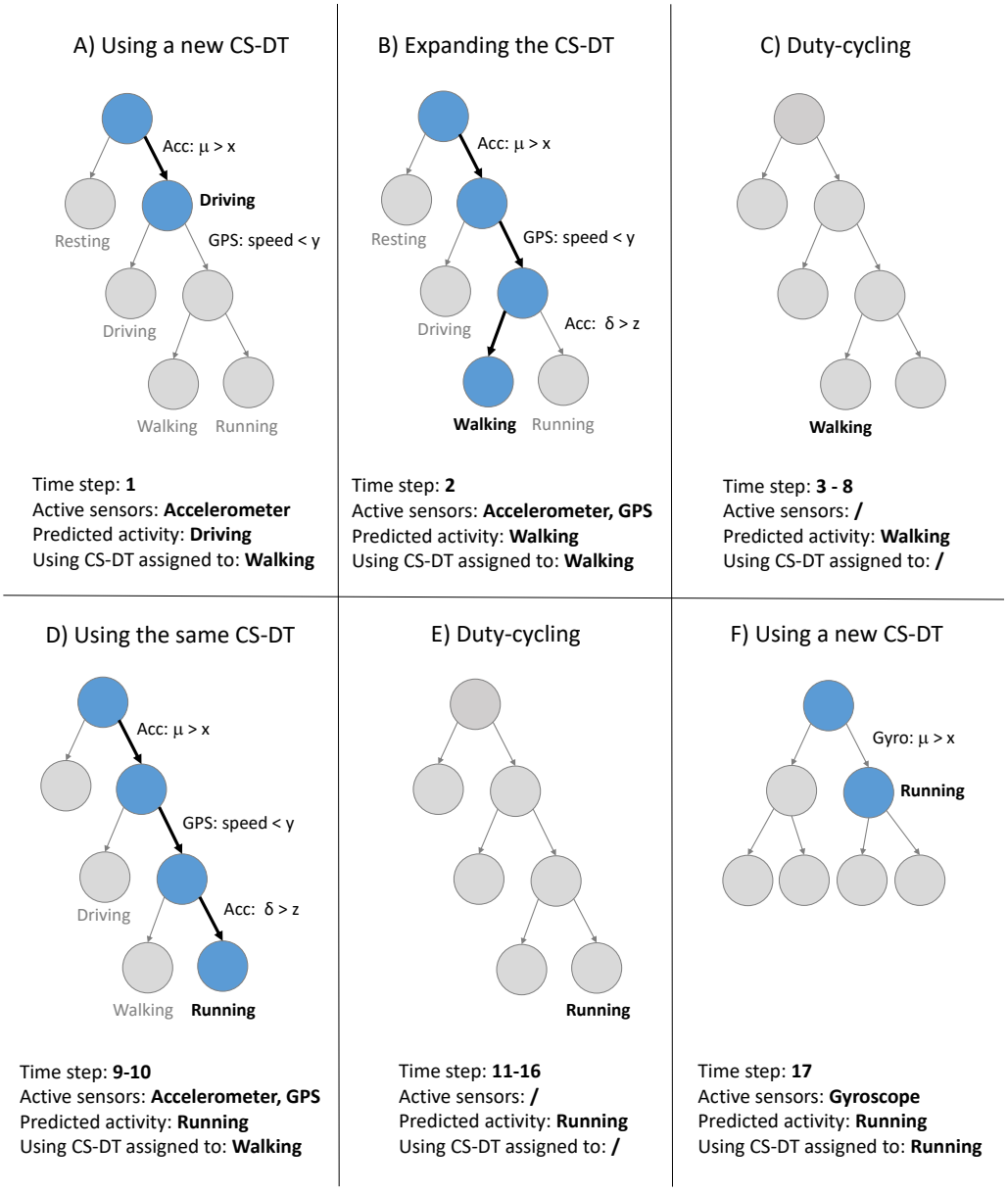

Figure 5.21: An example of using a combination of all three methods. For each consecutive time step, we list the sensors used, CS-DT tree used and the predicted activity.

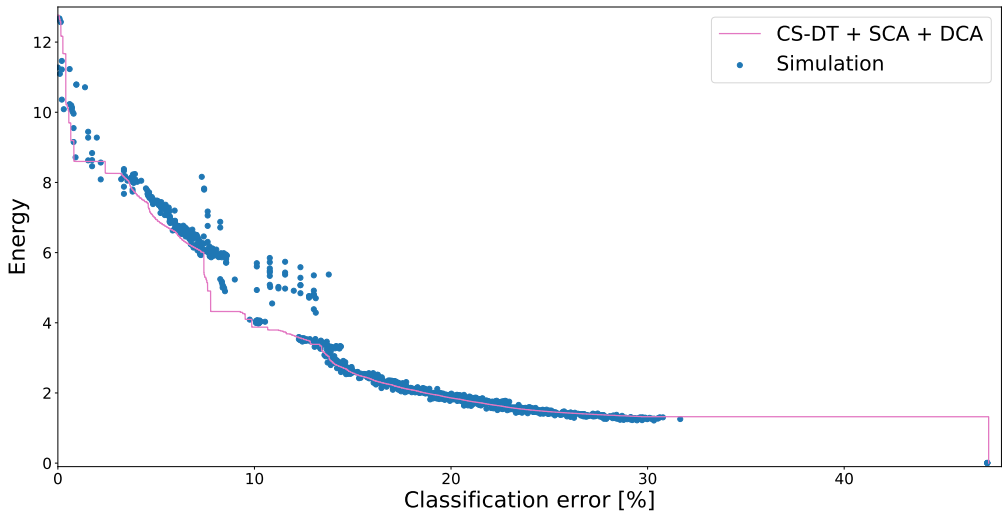

Figure 5.22: CS-DT + SCA + DCA predictions on the *Hidden* dataset, compared to the actual performance values achieved in a simulation.

5.2 Real-Life Datasets

This section analyzes the use of the presented methodology on four real-life datasets. For each, we compare different methods between themselves and to various baselines. We test the time of evaluation and the accuracy of predictions. Some found solutions are also explicitly shown and semantically evaluated. The comparison of the solutions found with our methods to those found with algorithms from the related work is left for Section 5.3.

All four datasets from Section 4 were used for testing the SCA and DCA methods and their combination. However, only the Commodity12 and E-Gibalec were used for testing the CS-DTs. This is due to the weak performance of decision trees (both normal and cost-sensitive) on the remaining two datasets. As an example, on the SHL dataset the maximum accuracy achieved using a decision tree was 77% compared to the maximum of 88% achieved by the random forest classifier. This decrease in performance outshadows any potential gains that could be achieved by additional optimisation by the cost-sensitive trees.

Finally, the E-Gibalec dataset is used to demonstrate how to use our methodology in cases where the distribution of contexts is not known in advance.

5.2.1 SHL Experiments

The SHL dataset was recorded using five different sensors and labeled with eight different contexts as described in Section 4.2. The energy requirements for each sensor were not provided, and we simply assumed that each has the same energy cost. This simplifies the energy cost to being proportional to the amount of the data used. Using half of the dataset thus has the cost of "50%" regardless of whether the half was due to using half the sensors, sensors being off half the time due to duty-cycling or sampling frequency being down-sampled by a factor of two.

5.2.1.1 SCA

For the SCA method we used the same settings that we used when testing for the SHL competition [34]: sampling frequencies of 100 Hz and 50 Hz, and every combination of sensors. Using the 50 Hz sampling frequency resulted in the same classification accuracy as using the 100 Hz sampling frequency, with half the energy cost. It was thus always used, making the baseline energy cost "50%". The trade-offs acquired with the SCA method that use different sensor combinations are shown in Figure 5.23 alongside the *static solutions* where the same setting is constantly used.

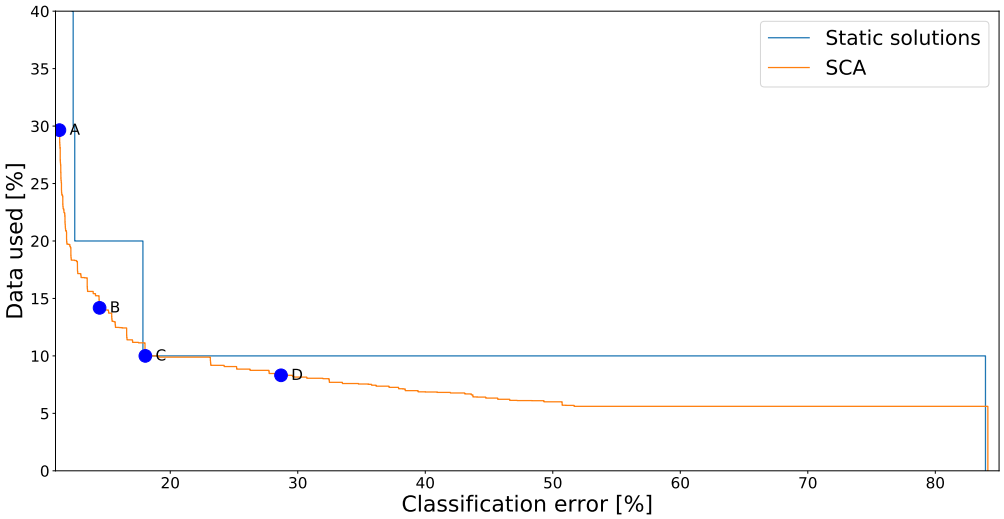

Figure 5.23: Comparison of assignments, where all contexts have the same sensor combination assigned (Static solutions), to the assignments where active sensors dynamically change based on the last classified context (SCA). Some points of interest are marked with letters A-D. This comparison was done on the SHL dataset.

Figure 5.23 also marks some sample solutions. Their details are shown in Table 5.7.

- Solution A uses the accelerometer regardless of the activity, which is not surprising as the accelerometer is often used in similar context recognition tasks. Additionally it uses the magnetometer whenever a vehicle is detected, which makes it easier for the classifier to distinguish them. Pressure is used when detecting subway and walking (potentially towards it) – as the pressure is different underground. These automatically found sensor assignments thus neatly match the human intuition about the task. Additionally, it is interesting to note that the accuracy of this assignment is 1 percentage point better than in any of the static solutions.

- Solution B uses only the accelerometer during all activities, and the magnetometer when a vehicle is detected. While not using three of the five sensors reduced the accuracy, the reduction was quite minimal compared to the solution A (cca. 3 percentage points). Some similar solutions (not explicitly shown here) used the gyroscope instead of the accelerometer when either walking or running – implying that both sensors are roughly equal for the task.
- Solution C uses only the accelerometer for all activities. Its performance therefore matches one of the static solutions.
- Solution D starts abandoning the recognition of some activities altogether by not using any sensors when that activity is detected. The *train* and *subway* activities were the first to be dropped, probably due to having the worst recognition accuracy even when appropriate sensors were used. Note that if no sensors are active, the majority classifier is used – classifying everything as *car*.

	Solution A	Solution B	Solution C	Solution D
Accuracy [%]	88.7	85.5	82.0	71.3
Energy [%]	29.6	14.2	10	8.3
Bike	acc, gyr, ori	acc	acc	acc
Bus	acc, mag, gyr	acc, mag	acc	acc
Car	acc, mag	acc	acc	acc
Run	acc, gyr	acc	acc	gyr
Still	acc, gyr, ori	acc	acc	acc
Subway	acc, mag, bar, ori	acc, mag	acc	\
Train	acc, mag, gyr, ori	acc, mag	acc	\
Walking	acc, bar	acc	acc	acc

Table 5.7: The sensors used when the corresponding activity is detected for each of the four solutions marked in Figure 5.23. Sensors are abbreviated: acc-accelerometer, mag-magnetometer, ori-orientation sensor, gyr-gyroscope, bar-barometer. Both accuracy and energy costs (measured in the percentage of data used) are provided as well.

As with every dataset we tested the accuracy of the SCA model and its evaluation time. The results are shown in Table 5.9 and Table 5.8 respectively. Again the SCA model proves to be much faster than the actual simulation and almost as accurate.

	Simple	SCA	Simulation
Evaluation time [ms]	4	7	440

Table 5.8: Time of evaluation of the mathematical models compared the time of the simulation. Test was repeated and averaged over 100 random assignments.

5.2.1.2 DCA

We tested different duty-cycle lengths – up to a maximum length of 30 – using the DCA method. The active period of 1 was used in all cases. Note that since one window of data corresponded to one minute, the duty-cycle length of 30 corresponds to sensors working one minute, and then sleeping for half an hour. As the starting point we chose the configuration where all the sensors are used. The results are shown in Figure 5.24 alongside the

	Accuracy [p.p.]	Energy [p.p.]
SCA - Random	0.8	0.1
SCA - Pareto	0.5	0.1
Simple - Random	13.4	1.5
Simple - Pareto	18.9	4.8
DCA - Random	2.8	0.47
DCA - Pareto	2.0	0.04

Table 5.9: Difference (mean absolute error) between the predicted accuracy/energy and the one calculated in a simulation for the SHL dataset. The experiment was repeated for both Simple, SCA and DCA method and for both 100 random assignments, and the assignments that form the corresponding Pareto front approximation. The difference was measured in percentage points (p.p).

solutions where the same duty-cycle length was used for every activity (*static solutions*). The solutions found with the DCA method outperform the static solutions, especially when short duty-cycles are used. Compared to SCA solutions, the energy gains at high accuracy are lower, but they get substantially higher if low-accuracy solutions are acceptable.

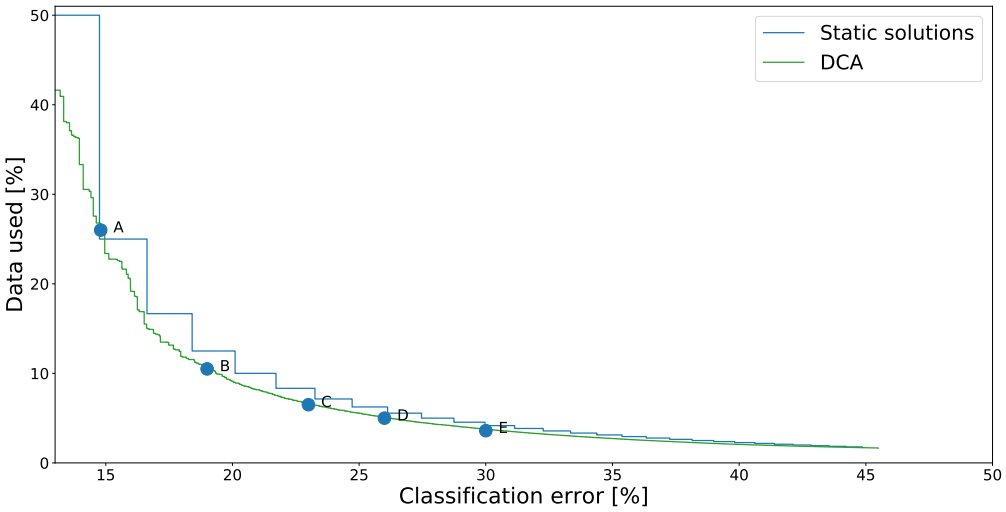

Figure 5.24: Comparison of assignments on the SHL dataset, where all contexts have the same duty-cycle length assigned (Static solutions), to the assignments where duty-cycles lengths dynamically change based on the last classified context (DCA).

Some sample solutions are listed in Table 5.10. They show that activities: *run*, *walking*, *still*, have short duty cycles assigned. Intuitively, *run* is a relatively short activity, while *walking*, *still* act as hubs from which other activities often transition. The *car* and *train* activities are longest on average (as listed in Table 4.2), and have correspondingly long duty-cycles. Interestingly, *bus* has longer duty-cycles assigned than *train*, maybe due to the fact that *bus* classifications are more likely to be correct than *train* classification.

The measured time of evaluation is given in Table 5.11, while the accuracy of the DCA prediction is both in Table 5.9 and graphically in Figure 5.25.

	Bike	Bus	Car	Run	Still	Subway	Train	Walk	Acc. [%]	Data used [%]
A	0	2	5	2	1	1	0	1	85	26
B	4	4	7	4	3	5	4	3	81	10
C	4	10	13	3	5	7	7	5	77	7
D	7	13	13	6	6	10	11	6	74	5
E	11	13	20	7	8	14	15	9	70	4

Table 5.10: Sample solutions returned by the DCA methods for the SHL dataset. For each activity the assigned duty-cycle length is given (the length of the sleeping phase of the cycle). In addition, both the accuracy and energy costs are listed. Duty-cycle length of 0 implies that duty-cycling should not be used.

	DCA	Simulation
Evaluation time [ms]	1.3	420

Table 5.11: Time of evaluation of the DCA model compared to the time of the simulation. The test was repeated and averaged over 100 random assignments.

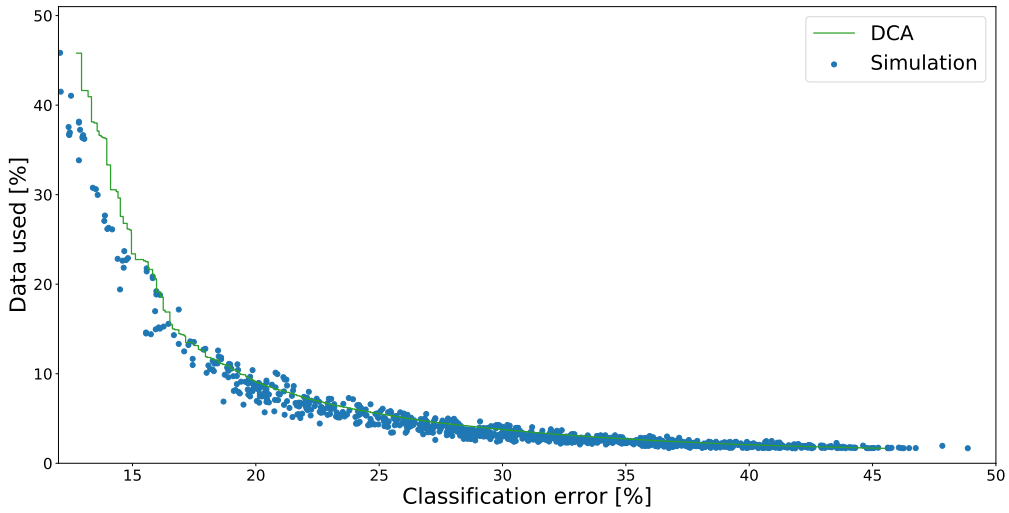

Figure 5.25: DCA predictions on the SHL dataset, compared to the actual performance values achieved in a simulation.

There is a visible bias of the DCA method in predicting less accurate performance of the system, compared to the real performance (Figure 5.25). On the closer inspection, it was determined to be an artifact of non-Markovian misclassifications (subsequent instances are more likely to be either both correct or both misclassified than two separate instances even for the same context due them usually being very similar). When we simulated the performance using random misclassifications – still with respect with probabilities given by the confusion matrix – the bias disappeared.

5.2.1.3 SCA and DCA Combination

Lastly, we tested the SCA + DCA combination. Five SCA solutions were automatically chosen and expanded with the DCA method. The resulting five sets of solutions are shown in Figure 5.26. The five solution sets all converge to similar values, but as with other datasets, the sets that are worse at high accuracy outperform others at low accuracy. Additionally note that in case of solution sets 4 and 5, the sets do not start at the corresponding points 4 and 5, but a little below. This seems unusual, as combining a SCA solution with the DCA assignment that assigns no duty-cycling to each context, should be the same as using the original SCA solution – thus at least one solution should touch the SCA line. The analysis of the phenomenon reveals that in cases in question there existed a DCA assignment with low duty-cycle lengths that did not reduce the accuracy. Such a solution obviously dominated the previously mentioned one.

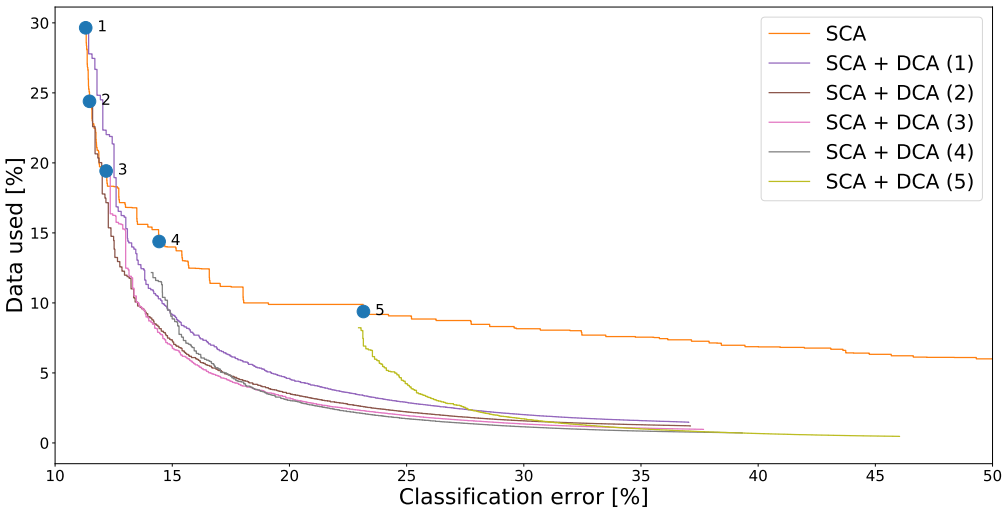

Figure 5.26: SCA, DCA predictions on the SHL dataset, generated by starting at different SCA solutions (marked with points 1-5).

The comparison of SCA, DCA and combined solutions (the non-dominated set from Figure 5.26) is shown in Figure 5.27. The SCA + DCA combination is much better than the other two individually. Two solutions (arbitrarily chosen) are singled out and marked in the same figure. The first one represents a trade-off with the accuracy of 86% – 2 p.p. less than the best achieved accuracy – using only 10% of all available data. The second one represents a trade-off with the accuracy of 83%, using 5% of all data. Thus (assuming the correlation between the data used and energy consumption) we can conserve 95% of battery life by sacrificing only 5 p.p. of accuracy.

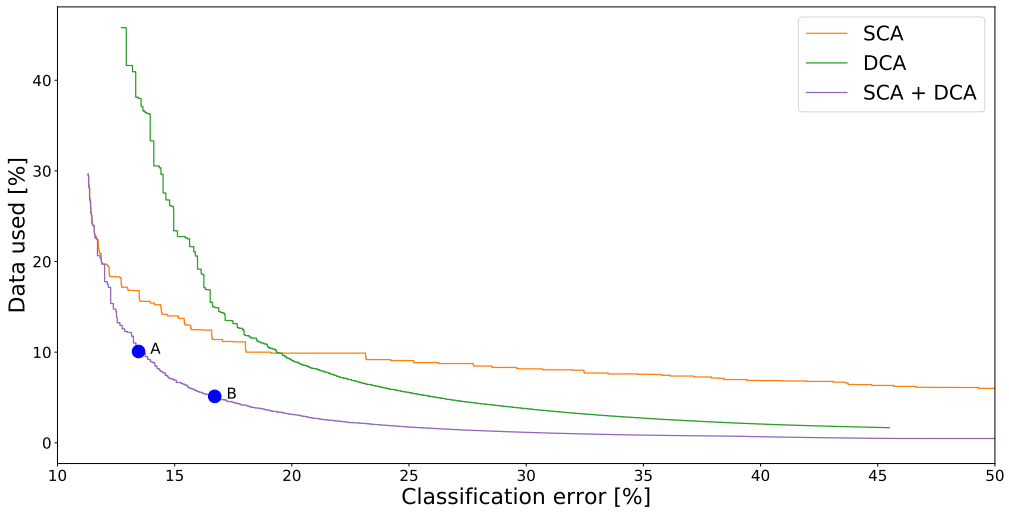

Figure 5.27: Comparison of SCA, DCA and their combined solutions on the SHL dataset. Two solutions are singled out as reasonable trade-offs between accuracy and energy consumption.

5.2.2 Opportunity Experiments

The unique challenge of the Opportunity dataset lies in the large number (18) of possible contexts in addition to the large number of sensors (30 sensor clusters, each containing 1–3 different sensors). Instead of accuracy, the *macro-averaged F-score* was used to determine the quality of the classification to avoid the system disregarding minority classes. The *macro-averaged F-score* is simply the arithmetic mean of *F-score* (Equation 5.1) for each activity. The starting *F-score* using all the sensors is 0.33 while the *F-score* using the best sensor subset is 0.44. The relatively low score is due to the large number of (hard to distinguish) contexts.

$$F\text{-}score = \frac{2 * precision * recall}{precision + recall} \tag{5.1}$$

5.2.2.1 SCA

Defining each possible sensor subset as a possible setting (or even sensor location subsets) would create an enormous search space in addition to requiring the training of 2^{30} classifiers (one for each subset). Instead, we used a feature selection procedure defined in Section 4.3 to select 159 specific subsets that are used as potential settings for the SCA method. Using these settings, the SCA method found trade-offs depicted in Figure 5.28 compared to the static solutions, where one of these settings is constantly used.

One trade-off is marked and described in Table 5.12. Due to the large number of sensor locations, the marked assignment can be somewhat challenging to interpret. As a general rule, one sensor on each hand is always active (upper and lower arms are the preferred locations). Even though the target for classification is the recognition of the object in the right hand, both hands are usually being used – the user is eating most of the time he uses an object – and movement of left hand can help infer the object in the right. Still, the sensor locations on the right hand are picked roughly three times as often as on the left hand. Some objects, such as door or fridge, are handled using only the right arm, and

in those cases the SCA method correctly detects that left hand sensors are not needed. The only non-hand sensor that is consistently used is the back sensor. Many objects have sensors attached to them and one could assume that those sensors should be used to detect if the given object is being held. This is sometimes the case (e.g., with bread, plate, knife and spoon), but using those sensors results in many false positives if the object is usually handled by the left arm. As an example, to the detect the *salami* instead of using the sensor attached to salami, the sensor attached to the knife is used instead (the salami is usually held with left hand, while being cut with the right. As a last note, the context is rarely recognized as either *spoon* or *switch* and the assignment to those two contexts has little influence on the overall energy consumption. Thus their assignments are mostly random and have unnecessarily many sensor locations assigned. The most frequent *none* context uses only four different sensor locations.

Point A	Sensor locations chosen
Bottle	hips, right knee, left wrist, right lower arm, left shoe, bottle
Bread	**back**, right wrist, right upper arm, right lower arm, left lower arm, **bread**, **knife1**, **knife2**
Chair	**back**, right wrist, right lower arm, right, upper arm, left upper arm, left lower arm
Cheese	hips, left wrist, right lower arm, left shoe, **bottle**
Cup	**back**, right upper arm, right lower arm, left lower arm, knife
Door	**back**, right upper arm, right upper arm
Drawer	right upper arm, right lower arm, left lower arm
Fridge	right upper arm, right lower arm, left shoe
Glass	right knee, right upper arm, left upper arm, **bottle**, **cup**, cheese, knife2
Knife	**back**, right upper arm, right lower arm, left lower arm, cup, **knife1**, **knife2**, milk, plate
Milk	**back**, right lower arm, left lower arm
None	**back**, right wrist, right lower arm, left lower arm
Plate	right knee, black, right upper arm, right lower arm, **cup, salami, knife1, knife2, plate, glass**
Salami	back, right upper arm, right lower arm, left lower arm, **knife1**, **knife2**
Spoon	hips, right upper arm, **back**, right knee, right upper arm, left upper arm, right shoe, cup, knife1, **spoon**, plate
Sugar	left upper arm, right upper arm, left shoe, cup, glass
Switch	right knee, **back**, right upper arm, left upper arm, right lower arm, left upper arm, cup, salami, knife1, knife2, milk, plate, glass
Table	**back**, right lower arm

Table 5.12: For each object we list sensor locations to be activated if the assignment in Figure 5.28 is selected. Some locations of interest are bolded. This assignment uses 5 sensor locations on average and has an *F-score* of 0.42.

The time of evaluation of a single assignment is given in Table 5.13. One can note that while the SCA model is still faster than the simulation, the margin is much smaller than usual. This is due to the large number of contexts in this dataset. If all 24 contexts were used instead of the 18 selected in this work, the use of the mathematical model would be discouraged (the genetic algorithm for search would still be useful).

The accuracy of SCA predictions is listed in Table 5.14. The predictions are shown to be accurate, with a small bias (systematic undervaluing of the performance) similar to the SHL case for the same reason (Section 5.2.1.2).

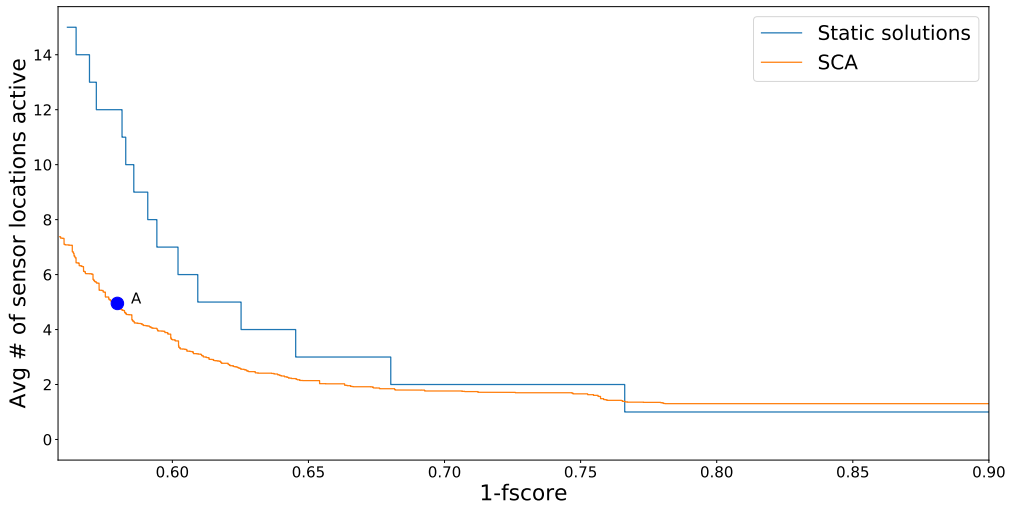

Figure 5.28: Comparison of assignments on the Opportunity dataset, where all contexts have the same sensor assigned (Static solutions), to the assignments where active sensors dynamically change based on the last classified context (SCA). A point of interest is marked and shown explicitly in Table 5.12

.

	Simple	SCA	Simulation
Evaluation time [ms]	64	158	263

Table 5.13: Time of evaluation of the mathematical models compared to the time of the simulation. The test was repeated and averaged over 100 random assignments.

	F-score	avg. # of active sensor locations
SCA - Random	0.01	0.08
SCA - Pareto	0.02	0.02
Simple - Random	0.09	0.6
Simple - Pareto	0.06	0.2

Table 5.14: Difference (mean absolute error) between the predicted F-score/average number of active sensors and the one calculated in a simulation for the Opportunity dataset. The experiment was repeated for both Simple and SCA method and for both 100 random assignments, and the assignments that form the corresponding Pareto front approximation.

5.2.2.2 DCA

For the duty-cycling we limited the sleeping length to 15 instances (7.5 seconds) as the contexts were shorter on average than with other datasets. Solutions found with the DCA method are listed alongside the static solutions (all contexts have the same duty-cycle length assigned) in Figure 5.29.

Assignments in some sample solutions (equidistantly picked) are shown in Table 5.15. Contexts with long duty-cycles assigned belong to two different categories: the first are the long-on-average contexts (e.g. *milk*, *none*, *plate*, *salami*); the second are the contexts with very low recall when using the selected sensor locations and no duty-cycling (e.g. *bottle*, *cheese*, *sugar*). The latter is a consequence of the fact that when increasing the duty-cycling length of one context, its recall increases, but the precision decreases. Since *F-score* values balance between the precision and recall, this trade-off becomes beneficial.

	Recall	Point A	Point B	Point C	Point D	Point E
Bottle	0.04	5	5	8	13	14
Bread	0.67	1	1	1	2	7
Chair	0.00	1	1	1	13	8
Cheese	0.02	15	10	14	12	15
Cup	0.64	1	1	2	3	8
Door	0.91	1	1	1	2	5
Drawer	0.82	1	1	1	2	3
Fridge	0.41	2	2	2	4	7
Glass	0.50	1	1	1	3	6
Knife	0.36	1	1	5	5	7
Milk	0.62	1	1	1	2	6
None	0.84	1	2	3	4	11
Plate	0.15	2	2	5	5	12
Salami	0.42	2	2	3	6	11
Spoon	0.39	1	1	1	3	4
Sugar	0.14	7	8	8	9	11
Switch	0.40	2	2	2	4	5
Table	0.86	1	1	3	3	4

Table 5.15: For each object we list the length of the duty-cycle assigned in addition to the recall for that object in case no duty-cycling is used.

The time of evaluation is given in Table 5.16. The time for the DCA model is longer than on the other domains, but in this case much shorter than the time for the simulation or compared to the time of the SCA model. The accuracy of the DCA predictions is given in Table 5.17 which shows that the predictions are generally accurate – with the energy predictions being slightly less so than on the other datasets.

	DCA	Simulation
Evaluation time [ms]	18	628

Table 5.16: Time of evaluation of the DCA model compared to the time of the simulation. The test was repeated and averaged over 100 random assignments.

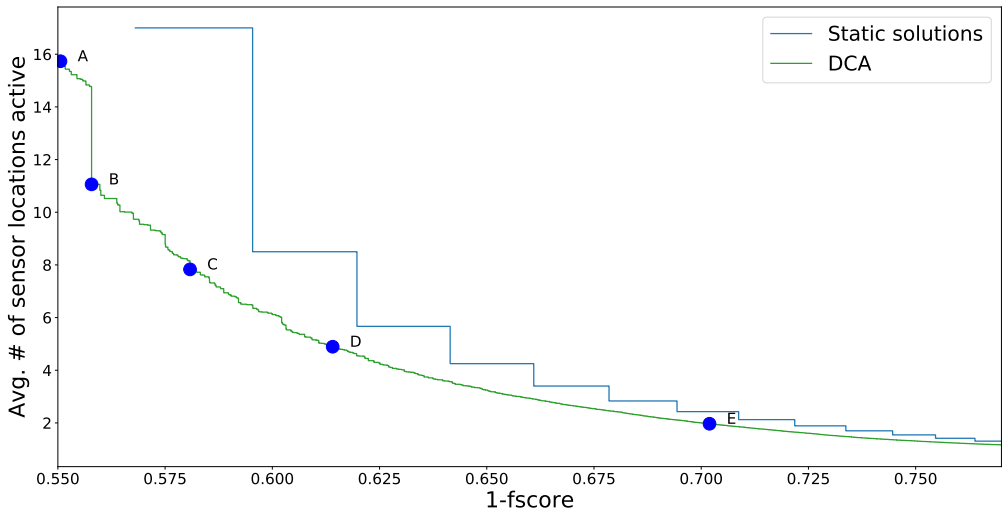

Figure 5.29: Comparison of assignments on the Opportunity dataset, where all contexts have the same duty-cycle length assigned (Static solutions), to the assignments where duty-cycles lengths dynamically change based on the last classified context (DCA). Some points of interest are marked with letters A-E.

.

	F-score	avg. # of active sensor locations
DCA – Random	0.03	0.47
DCA – Pareto	0.03	0.12

Table 5.17: Difference (mean absolute error) between the predicted F-score/average number of active sensors and the one calculated in a simulation for the Opportunity dataset. The experiment was repeated for both 100 random assignments, and the assignments that form the corresponding Pareto front approximation.

5.2.2.3 SCA and DCA Combination

Lastly, we combined the two methods together by using the DCA method to expand the 6 points found automatically with the SCA method, using the DCA method. As usual, pareto-dominant solutions are selected from all 6 solution sets and are shown in Figure 5.30. The accuracy of the predictions is graphically depicted in Figure 5.31 and shows that even when combining two different mathematical models we still get accurate predictions.

To evaluate the energy-efficiency of the found solutions we picked four of them (marked in Figure 5.30) and compared them to the baseline. The original dataset constantly used 30 sensors, but having 14 always active is adequate to retain the original *F-score*. Changing the sensors dynamically (using SCA, DCA combination) we can again achieve the same *F-score* using only 7 sensors on average – more than twice as few (point A). Using 4 sensors on average the *F-score* falls by 0.01 (point B), and using 2 sensors on average, the *F-score* falls by 0.04 (point C). Lastly, using only one sensor on average yields the *F-score* of 0.35 – which is only 0.11 less than the maximum.

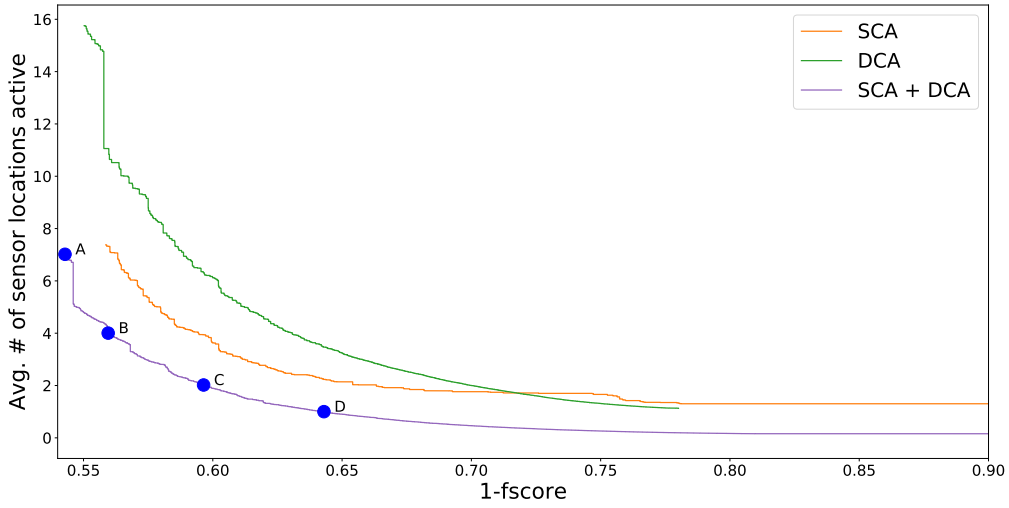

Figure 5.30: Comparison of assignments on the Opportunity dataset, made by three different methods. Some points of interest are marked with letters A-D.

.

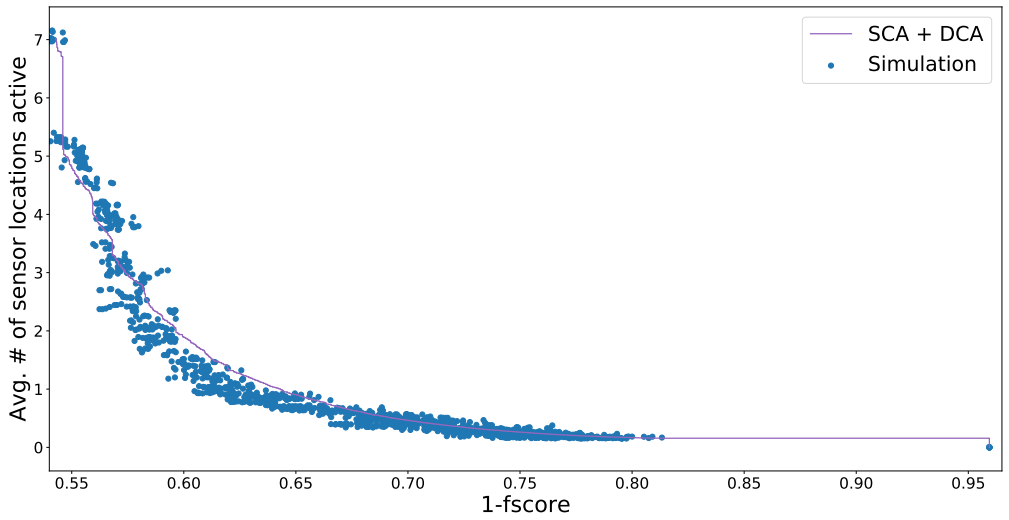

Figure 5.31: Every solution in Figure 5.30 evaluated again using a simulation. Simulated values are a close match to the predicted ones.

.

5.2.3 Commodity12 Experiments

Using the Commodity12 dataset we tested the methodology on four different users. For each we built personalized machine learning models and found personalized energy-efficient solutions. This approach can be considered unrealistic in practice, but can help demonstrate how different user habits can result in different solutions.

In all cases we operated with 10 different sensors, and the setting to be optimized was which sensor subset to use with which duty-cycle length.

5.2.3.1 SCA

For each of the four users we calculated the static solutions (always using the same sensor subset) in addition to solutions found by the SCA method. Both sets were plotted in Figure 5.32. Unsurprisingly, the SCA solutions dominate the static ones. It is interesting to note, however, how the shape of the Pareto front approximation changes from user to user – even though the problem setting is the same (same activities, sensors, energy cost). Note that the highest achievable accuracy varies from 84% to 88% in the case of static solutions and from 87% to 93% in the case of SCA solutions. The slightly higher accuracy using the SCA solutions can be attributed to the fact that some sensors generate only "noise" in some contexts and turning them off provides a boost to the accuracy as well as to energy savings.

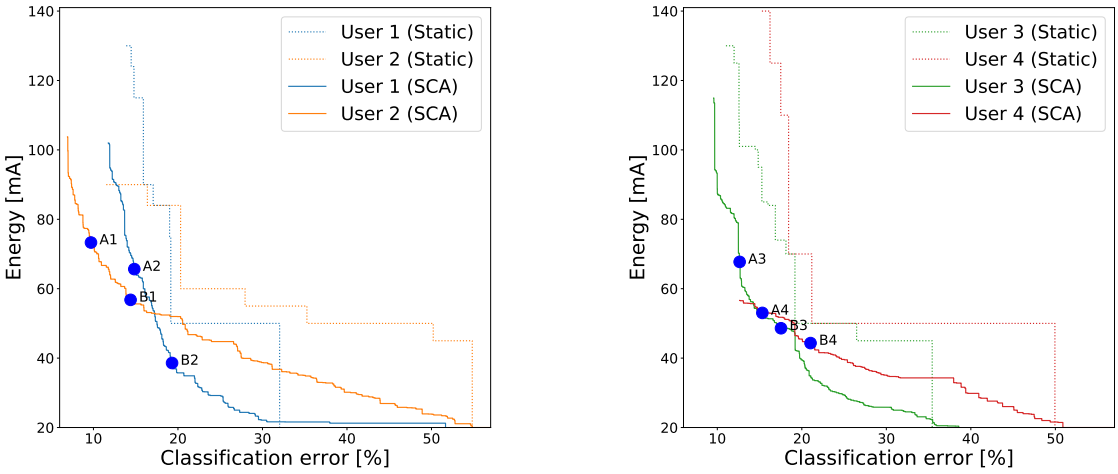

Figure 5.32: Comparison of the assignments, where all contexts have the same sensor assigned (Static solutions), to the assignments where active sensors dynamically change based on the last classified context (SCA). Solutions for two users (in no particular order) are plotted on the left, and the remaining two on the right. Some sample solutions are marked with points labeled A1-B4. Comparison was performed on the Commodity12 dataset.

To analyze the semantics of the found solutions, we isolated two solutions for each user: one where the accuracy is 3 percentage points lower than the maximum for that user (marked A) and one where it is 8 percentage points lower (marked B). The accuracy, energy and active sensors for each contexts are listed in Table 5.18.

Looking at the solutions marked by A points, it can be observed that some contexts have similar sensors assigned independently of the user: *work* and *eating*, for example, both frequently use *time* and *Wi-Fi* sensor. This can be explained by the fact that all

Points A	User 1	User 2	User 3	User 4
Accuracy [%]	85.2	90.3	87.4	84.7
Energy [mA]	65.6	73.3	67.7	53.0
Eating	time, hr, wifi, sqr	acc, time, gps	time, hr, wifi	time, wifi
Exercise	hr, gps, sqr	acc, time, gps, wifi	sound, acc, gps	\
Home	sound, wifi	sound, time gps	wifi, sqr	gps
Out	time, gps	hr, gps, wifi	sound, acc, gps	wifi
Sleep	\	sound, time	\	\
Transport	sound, time, wifi	time, gps	time, hr, gps, sqr	wifi
Work	time, wifi	wifi	sound, time	wifi
Points B	User 1	User 2	User 3	User 4
Accuracy [%]	80.7	85.6	82.5	79.0
Energy [mA]	38.6	56.8	48.6	44.3
Eating	time, gps, wifi	time, gps	time, hr, wifi	time, wifi
Exercise	time, hr, wifi, sqr	time, wifi	sound	\
Home	time, wifi	time, wifi	wifi	time
Out	time, gps	acc, time, gps	hr, gps	acc, time, wifi
Sleep	\	sound, time	\	\
Transport	time, wifi	time, gps	time, wifi	wifi
Work	time	wifi	time, hr	wifi

Table 5.18: The sensors used when the corresponding context is detected for each of the eight solutions marked in Figure 5.32. Some sensors are abbreviated: sqr - FourSquare webservice, wifi - Wi-Fi, acc - (accelerometer, pressure, light), hr - (hear rate, respiration rate). Multiple sensors are listed in brackets as they all share a cost and are thus either all included or excluded. Some contexts/user combinations have no sensor assigned as that user was never in that context in the training part of the dataset.

participants were connected to one Wi-Fi access point while in office and another one while in the cafeteria. Most of other contexts, however, are detected differently depending on the user. *Transport* is detected either by Wi-Fi (more accurately by seeing different access points and not being connected to any), GPS or FourSquare service. Similarly, the *out* context can be detected by GPS location, sound or even by a different heart rate. Solutions marked by B points are a lot more homogeneous – most consist of using time and some localization sensor (Wi-Fi or GPS). Not only is the time "sensor" free, most participants had regular habits of when they are at work, home or transporting in-between.

All found solutions are deeply personalized and as such are not transferable from one user to another. To demonstrate with an experiment, we took solutions from all four users and evaluated them on the dataset from User 1. The results (Figure 5.33) show that the solutions from User 1 work well on User 1, while the solutions from other user are significantly worse (more than 10 p.p. in some cases). Especially the solutions generated by User 4 data are completely incompatible with User 1. To generate more general (unpersonalized) solutions, one could run the methodology on aggregated data of all users. Easily testing how different solutions work in different circumstances – in this case on different users – is one big advantage of our methodology.

The time of evaluation using the SCA model and its accuracy are shown in Tables 5.19 and 5.20. While the model is fast and outperforms the Simple model in accuracy, the MAE of User 2 and User 4 is more than 5% (on solutions that form the Pareto front approximation) which is the greatest error of the SCA model across all tested datasets.

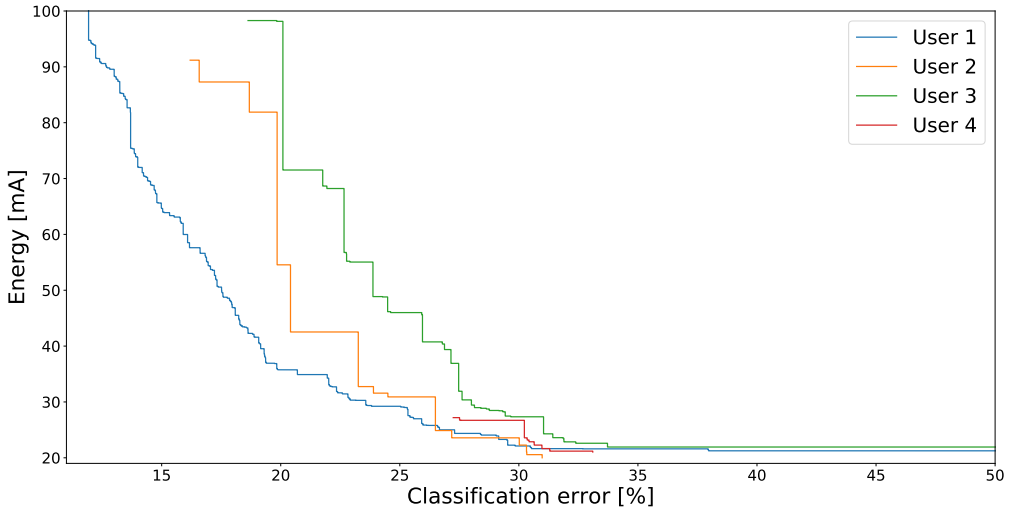

Figure 5.33: Comparison of trade-offs that were generated by using assignments created by using the data from User 1, and applying them to data for each of the four users.

	Simple	SCA	Simulation
Evaluation time [ms]	3	5	121

Table 5.19: Time of evaluation of the mathematical models compared to the time of the simulation. The test was repeated and averaged over 100 random assignments on the Commodity12 dataset.

	Acc. (Simple)	Acc. (SCA)	Energy (Simple)	Energy (SCA)
User 1 - Random	5.2	2.4	3.2	1.3
User 1 - Pareto	4.0	1.1	2.7	1.0
User 2 - Random	7.5	3.8	4.7	1.7
User 2 - Pareto	9.7	5.2	4.3	2.4
User 3 - Random	3.3	2.4	2.0	0.8
User 3 - Pareto	2.7	0.7	2.8	1.5
User 4 - Random	5.1	3.7	3.5	2.2
User 4 - Pareto	9.3	6.8	3.7	3.5

Table 5.20: The difference (mean absolute error) between the predicted accuracy/energy and the one calculated in a simulation for the Commodity12 dataset. The experiment was repeated for each user, for both the Simple and SCA method and for both 100 random assignments and assignments that form the corresponding Pareto front approximation. Difference in accuracy was measured in percentage points, while the difference in energy is expressed in mA.

5.2.3.2 DCA

We started by running the DCA method on all four users. The results are shown in Figure 5.34. Static solutions (all contexts have the same duty cycle length assigned) are plotted alongside for comparison. The first thing of note is the rapid drop in energy consumption when using duty-cycling – much faster than when choosing sensor subsets using the SCA method (they are shown side by side in Figure 5.35). This can be explained by noting the long average lengths of many contexts, e.g., *work*, *sleep*. A small peculiarity in the results for User 1 is that there are two static solutions that outperform those found by the DCA. Given that the static solutions are a part of the search space for DCA (searching using NSGA-II algorithm) and should therefore be found, one can take it as an indication that search parameters can be further tuned for better performance (e.g., bigger population, more generations).

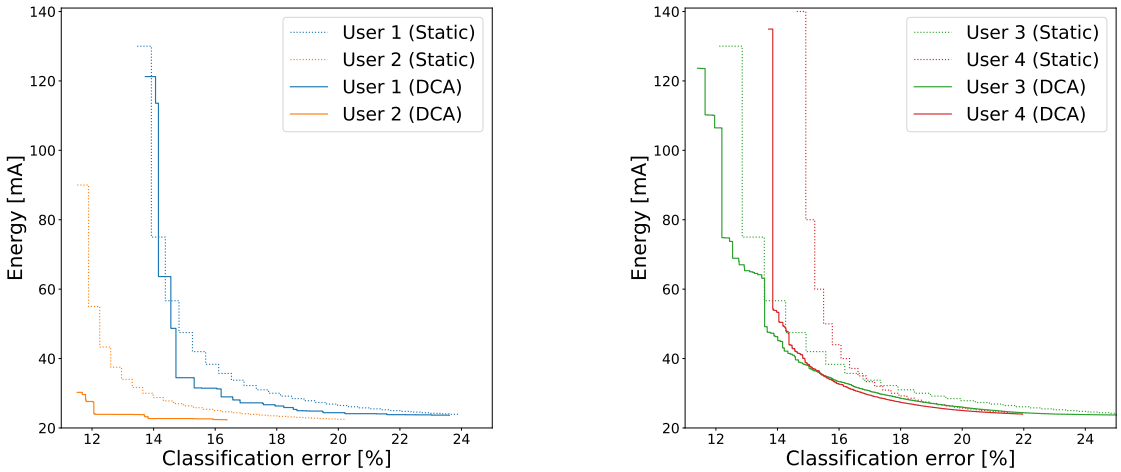

Figure 5.34: Comparison of the assignments, where all contexts have the same duty-cycle length assigned (Static solutions), to the assignments found by the DCA method. Solutions for two users are plotted on the left, and the remaining two on the right. The comparison was done on the Commodity12 dataset.

The time of evaluation and accuracy of DCA model are given in Table 5.21 and Table 5.22. DCA model is again proven fast and accurate.

	DCA	Simulation
Evaluation time [ms]	0.7	225

Table 5.21: Time of evaluation of the DCA model compared to the time of the simulation. The test was repeated and averaged over 100 random assignments.

	User 1	User 2	User 3	User 4
Accuracy	0.7	1.3	2.2	1.2
Energy	0.2	0.4	0.2	0.5

Table 5.22: Difference (MAE) between the predicted accuracy/energy and the one calculated in a simulation. The experiment was repeated for each user.

User	Eating	Exercise	Home	Out	Sleep	Transport	Work	Acc.	Energy
1	3	4	10	16	/	5	25	89	26
2	7	2	10	12	1	8	13	83	30
3	8	9	10	2	/	2	13	83	30
4	14	/	7	19	/	11	17	83	31

Table 5.23: Sample solutions returned by the DCA methods for the Commodity12 dataset. For each activity the assigned duty-cycle length is given. In addition, both the accuracy [%] and energy costs [mA] are listed.

As sample solutions we selected a solution with the energy gain of 10 for each user. The resulting four solutions are shown in Table 5.23. They are much more similar to each other then the SCA solutions and thus more easily transferable from user to user. *Work*, *home* and *out* have long duty cycles assigned, while others have comparably short ones. Sleeping was assigned a duty-cycle length of 1 which is curious, since *sleep* is one of the long activities. The reason lies in the fact that *sleep* is very often misclassified and it is therefore beneficial to make another classification as soon as possible to confirm this context.

5.2.3.3 SCA and DCA Combination

For each of the four users we used the SCA and DCA method combination by expanding five automatically chosen trade-offs found with the SCA method using the DCA method. A non-dominated set is then chosen from all the trade-offs. The results (Figure 5.35) from this method combination are consistently and visibly better than using only one of them, across all users and across the whole solution range.

It is interesting to note that for each user there exists one "knee" in the shape of the Pareto front approximation that can be considered as the area with most reasonable trade-offs. We picked one such solution from each user and compared its energy consumption with the energy consumption of always using the sensor subset with the best accuracy (Table 5.24). In all cases the classification accuracy remained roughly the same (less than 1% difference) and the energy optimization was basically "free". This can be attributed to the facts that using only a small sensor subset could still provide accurate predictions, that activities are long on average and their sequence usually predictable.

ID	1	2	3	4
Energy [mA] (Static)	130	90	130	140
Energy [mA] (SCA, DCA)	30	25	36	23

Table 5.24: Comparison between energy consumption for each of the users before and after the optimization using the SCA + DCA combination. Note that using no sensors has a baseline cost of 20 mA.

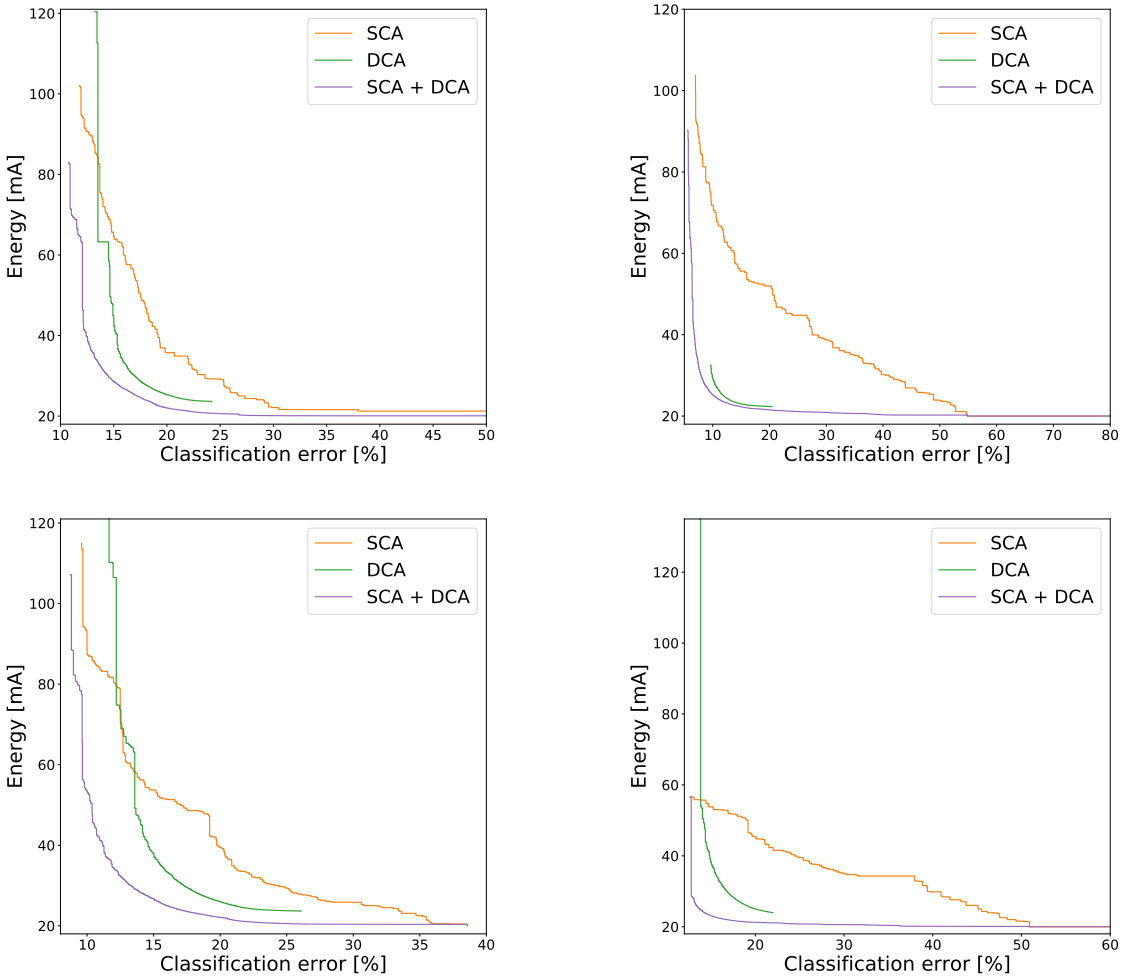

Figure 5.35: Comparison of the SCA and DCA methods to their combination on the Commodity12 dataset. Each figure presents the found solutions of one of the users (1–4) in the order: top left, top right, bottom left, bottom right.

5.2.3.4 CS-DT

The decision trees classifier is less accurate than the previously used classifier (random forest) on this domain, but the difference is small enough (roughly 2–5%) to justify using the CS-DT method. To employ the CS-DTs optimally, however, we had to pick some parameters – which kind of tree building process to use and which tree subset to use for the CS-DT + SCA combination. Both choices were made for each user individually and in each case we chose the parameters that maximized both the quality of trade-offs and their spread over different values.

For the tree building procedure we chose the baseline tree construction method for one user (Section 3.3.1), grouping the features that share the energy cost with one another (Section 3.3.2.4) and extending the tree and pruning it for the remaining two (Section 3.3.2.3). For the last method the best performing value for parameter n was 1. The feature grouping method frequently found only one good solution (when traversing different misclassification costs) and was thus not used more often. One example of the trade-offs generated by the three construction methods is shown in Figure 5.36 (left).

5.2.3.5 CS-DT and SCA Combination

Three different tree sets were tried for the CS-DT + SCA combination and again the best one was picked for each user. Creating CS-DTs using different sensor subsets (*settings*) was used for three users and it usually had the greatest spread of different trade-offs. Creating CS-DTs using different misclassification weights was used for one user, while the *context-specific* subset lagged behind in performance in all cases. An example of using the CS-DT, SCA combination with different tree subsets is shown in Figure 5.36 (right).

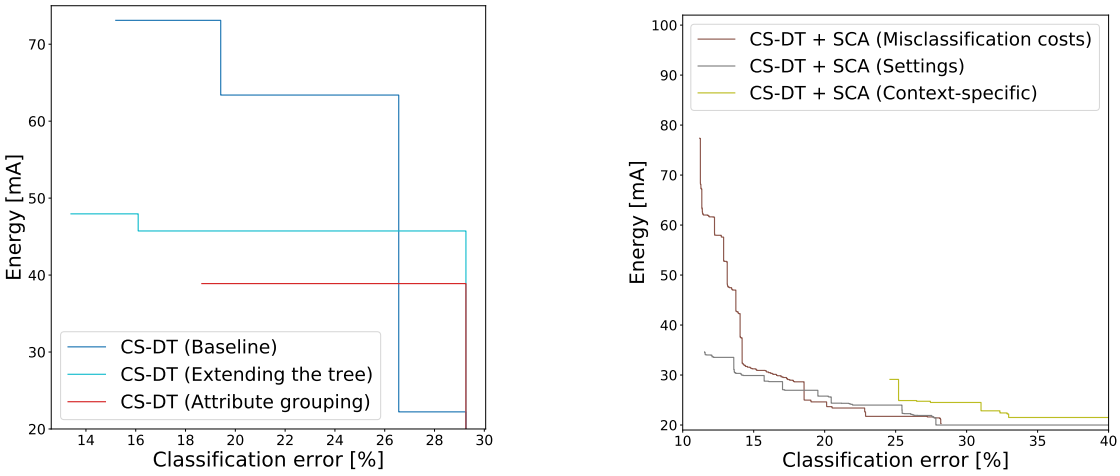

Figure 5.36: (left) Comparison of the trade-offs, where different CS-DT construction methods are used. (right) Comparison of trade-offs, where different tree sets were used for the CS-DT + SCA method combination. Both cases were taken when optimizing for User 1 on the Commodity12 dataset.

Despite starting with a disadvantage of lower accuracy the CS-DT + SCA combination clearly outperformed the SCA method alone in three out of four cases (when testing with User 3, the performance was comparable). The comparison can be seen in Figure 5.38. This demonstrates the strength of adapting not only to the recognized context but to underlying feature values as well.

We found the generated solutions semantically sensible. For example, a small part (pruned for clarity) of one context-specific solution is presented in Figure 5.37. One can observe that a completely different sensor subset can be used depending on the last classified context.

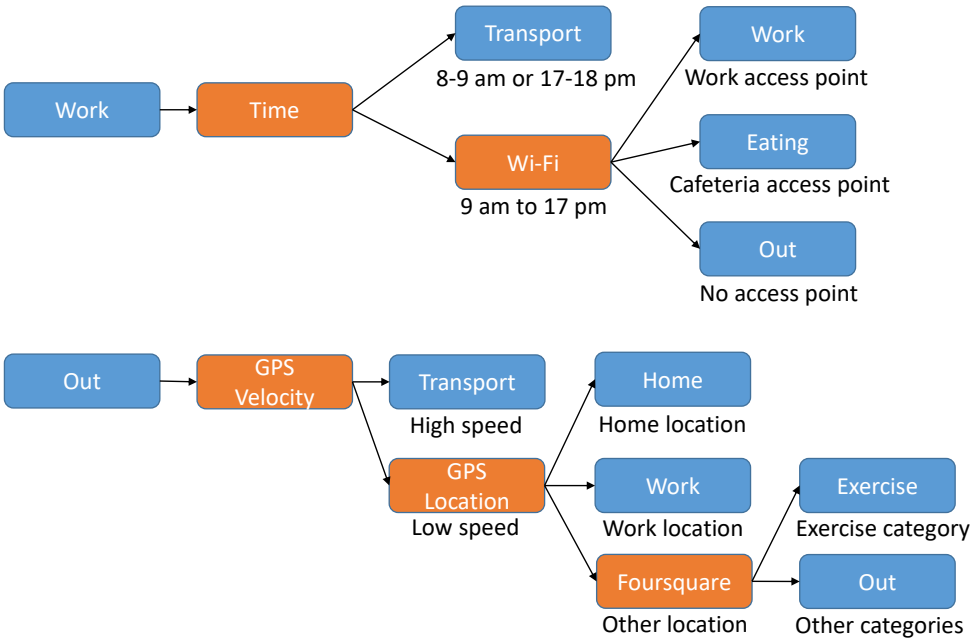

Figure 5.37: Parts of the CS-DTs assigned to the contexts *work* and *out* for User 1. Orange nodes show which sensor must be activated. The left-most blue node indicates for which context the corresponding tree is used, other blue nodes indicate to which context to classify.

5.2.3.6 SCA, DCA and CS-DT Combination

Finally, we used all three methods in conjunction. We chose the active phase of length 7, as this was the maximum number of needed periods in order to expand all necessary branches of a CS-DT. The results for all four users are plotted in Figure 5.38. Using this method combination outperforms using only the CS-DT + SCA combination. It does not, however, outperform using the SCA + DCA combination. This might appear curious, as adding the CS-DT was beneficial in the previous experiments (adding CS-DT to the SCA). The reason is that the contribution of duty-cycling is diminished if the CS-DTs are used, compared to when they are not. To test this hypothesis we tried using the three-method combination with active phase of length 1 and all the sensors needed for the current tree branch "magically" already active – and in this hypothetical scenario it performed better than any other method. In domains like this, where the duty-cycling provides a significant portion of energy savings, the use of CS-DTs can thus sometimes be detrimental.

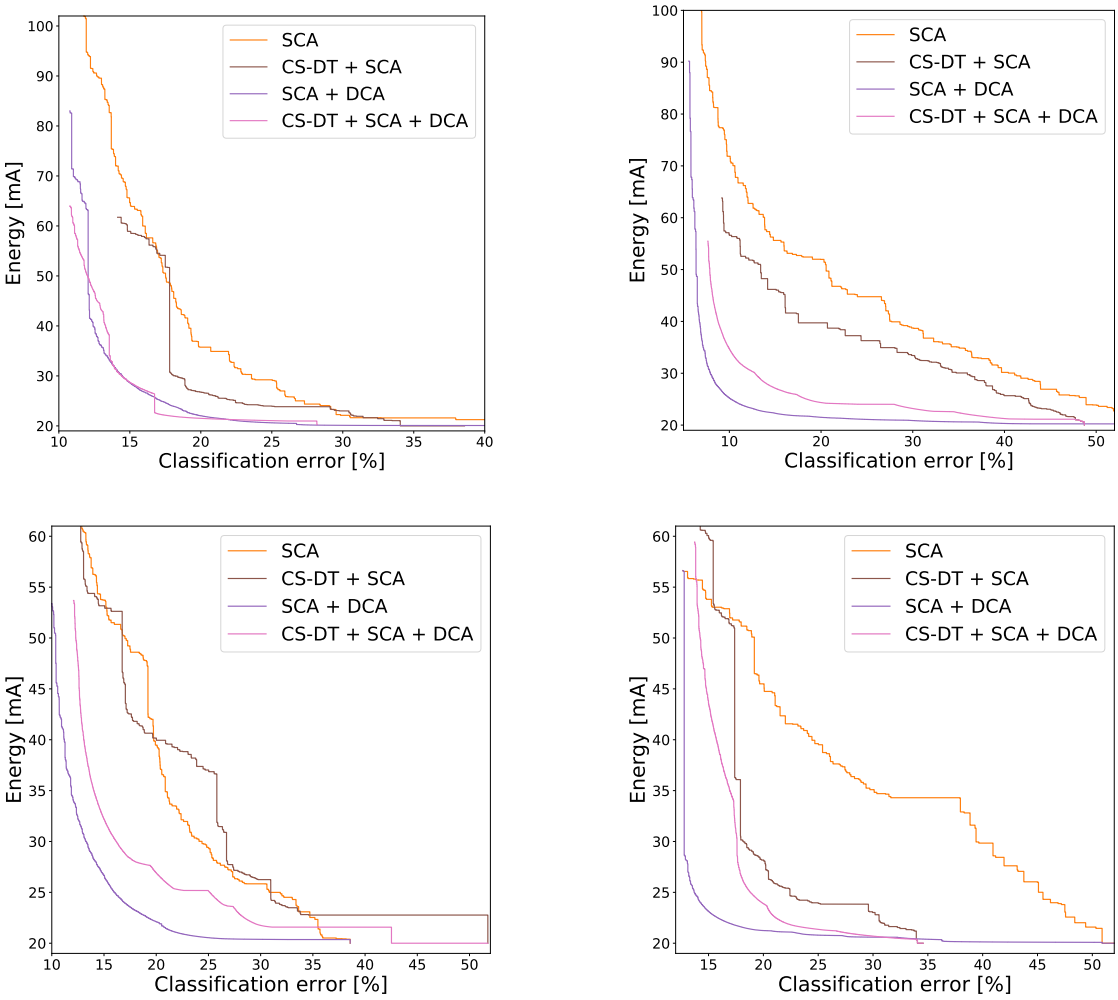

Figure 5.38: Comparison of the SCA method to different method combinations on the Commodity12 dataset. Each figure presents the found solutions of one of the users (1–4) in the order: top left, top right, bottom left, bottom right.

5.2.4 E-Gibalec Experiments

For the E-Gibalec dataset we used different sampling frequencies of the accelerometer sensor as the settings to optimize (in addition to duty-cycle length for the DCA method). There were a total of 8 different frequencies which corresponded to 8 different settings – which is a comparably small number if compared to "sensor subset" settings that numbered 2^{10} for the Commodity12 dataset and 2^5 for the SHL dataset. This, combined with only four different contexts (*other*, *walking*, *running*, *cycling*), heavily reduces the search space for the SCA method – enough in fact that NSGA-II algorithm is not really needed and every possible assignment could be listed and tested.

On this dataset we ran all methods and their combinations. Due to the simplicity of the dataset (indicated by the number of contexts and settings) we omitted the analysis of each individual step. The results are shown in Figure 5.39. There are a couple of points of note when interpreting the results. First, the CS-DTs usually work with streams of different sensors; in this case we simply replaced different sensors with the same sensor using different frequencies – doing so also demonstrating the flexibility of the methodology. Second, the CS-DTs have a roughly 4 percentage points better accuracy than the regular decision trees used for the SCA method. By pure coincidence, the implementation of regular decision trees using the *scikit-learn* library [111] using default parameters created inferior trees than our implementation, which is generally not the case. Third, the duty-cycling was very effective, especially compared to the other methods. Finally, the combination of all three methods was the best in most of the solution space, with the exception of low-accuracy solutions where it was dominated by the SCA + DCA combination.

The time of the evaluation for the SCA and DCA methods is listed in Table 5.25. Each was measured by averaging 100 random assignments. It is shorter than usual due to the low number of contexts. To demonstrate the accuracy of the model predictions, we simulated each solution of the SCA + DCA + CS-DT combination and plotted actual performance values alongside the predicted ones (Figure 5.40). They mostly coincide, with the average deviation of 1.3 percentage points in accuracy and 0.2 mA in energy consumption.

5.2.4.1 Dealing with Unknown Transition Probabilities

Performance of any energy-efficient solution is tightly knit to the proportions and transitions of the contexts to be recognized. It is easy to imagine that one cannot estimate, for example, the effect of duty-cycling without knowing the average length of the underlying context. While we so far assumed that we can deduce these properties from the dataset, this is not always the case. For the E-Gibalec dataset it was due to the artificial nature of the recording, but even if the recording happens in a natural environment without any instructions for the user being recorded, we cannot be certain that all other users will behave in the exact same way. In addition, the proportion of the recorded activities is not representative to the proportion of these activities in the real-life (e.g., even if we recorded the same amount of *running* as *walking* it is improbable that this is representative of real

	Model	Simulation
SCA	0.4	90
DCA	0.3	140

Table 5.25: Time of the evaluation [ms] of the mathematical models compared to the time of the simulation. The test was repeated and averaged over 100 random assignments on the E-Gibalec dataset.

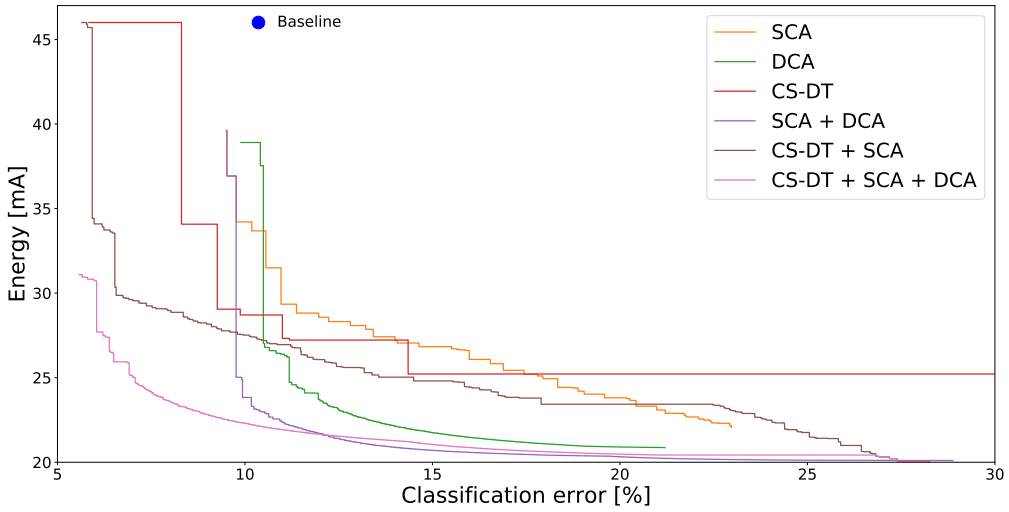

Figure 5.39: Trade-offs generated by each of the methods and their combinations on the E-Gibalec dataset. The baseline (using the maximum sensors frequency and no duty-cycling) is marked for comparison.

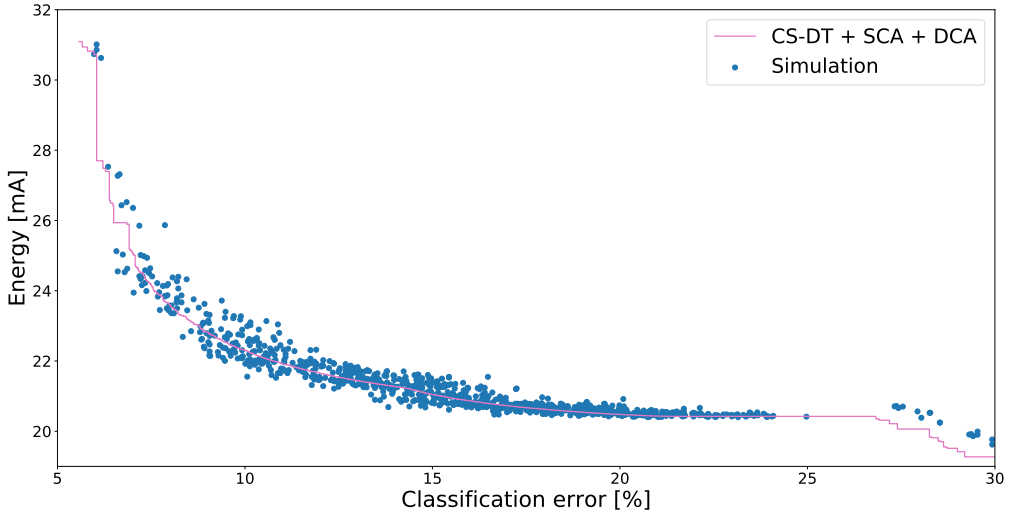

Figure 5.40: SCA + DCA + CS-DT predictions, compared to the actual performance values achieved in a simulation on the E-Gibalec dataset.

user behavior). This problem is common enough to deserve a more detailed analysis.

Using the presented methodology, we can estimate the performance of any solution (of any method or their combination) for different context distributions. This can be done without any modification of the dataset, by simply changing the transition matrix. Using either expert knowledge or observing differences between the recorded users, we can guess at reasonable ranges of context lengths and transitions. For each point in that range it is trivial and very fast to estimate the performance of the system. Doing so, the system designer can visually see how different user behaviors affect the system, and how robust the system is to such changes.

To make it more concrete, we took one sample solution from the SCA + DCA method combination (it is easier to interpret than using CS-DTs). This solution is shown in Table 5.26. We were interested in the effect of increasing the average length of *running* and *other* activities. The number of times the user went running or doing *other* (mostly resting) remains unchanged. Figure 5.41 shows how the system accuracy decreases if we expect the user to be an active runner, but increases if we expect *other* activity to be long. Note that in this case we altered each parameter independently from one another. The results are easy to interpret: *running* requires the highest sensor frequency, has the lowest duty-cycle length assigned, but even so has the worst prediction accuracy. Thus, if the time of running is increased, the energy required will also increase and the accuracy will decrease. The reverse is true for the *other* activity.

We can generalize and automate this process by choosing a maximum deviation in one parameter (e.g. the proportion of one activity, the average length of the activity) and calculating the transition matrix for each combination of such deviations (e.g. the proportion of *walking* increases, the proportion of *running* decreases). Using each of the transition matrices we can re-evaluate all the solutions, creating multiple sets of trade-offs. From these sets we can determine the best- and the worst-case scenario for each solution. As a demonstration, we took all solutions from the best-performing method (three-method combination) and plotted how their performance values change if the context distribution can change for a maximum of 10% for each activity. The best and the worst case are shown in Figure 5.42. One can observe that there is a roughly 5 percentage point accuracy difference between the cases.

This kind of analysis is omitted for other datasets, but is used here to demonstrate that the mathematical models are not only useful for finding good solutions, but also to reason about their robustness.

	Cycling	Other	Running	Walking
Frequency [Hz]	20	20	50	30
Duty-cycle length	7	10	2	8

Table 5.26: One assignment found using the SCA + DCA method. The sampling frequency and the duty-cycle lengths are shown for each activity. In such a system, after the activity is determined, one instance is classified using the listed frequency, then the system sleeps for the listed duration.

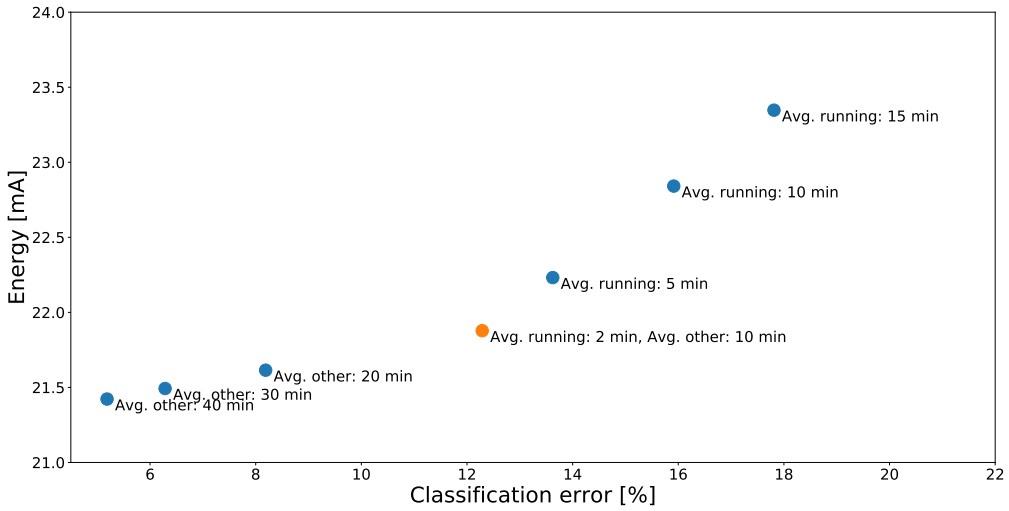

Figure 5.41: Different trade-offs between accuracy and energy consumption that occur when the average duration of activities changes if using the assignment from Table 5.26. The orange point represents the trade-off achieved when using the original activity lengths. Other points represent modifying either *running* activity average duration or *other* activity average duration.

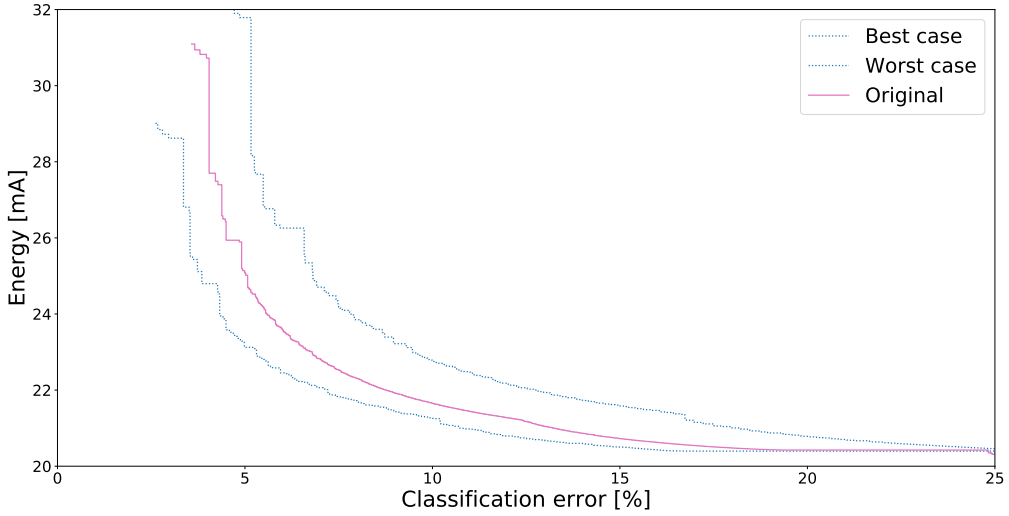

Figure 5.42: SCA + DCA + CS-DT combination trade-offs, compared to the trade-offs generated by the same assignments, but with a change in the underlying activity distribution. Comparison was done on the E-Gibalec dataset.

5.3 Comparing to Related Work

To further validate our approach we compared it to the related work described in Section 2.4.3. So far when comparing our different methods we have already shown that using a dynamic solution (that changes its sensing settings during runtime) is better than using a static one [30]. We have also shown that using our mathematical models is faster than using a simulation, and more accurate than using simpler mathematical models. In this section we additionally implement five alternative approaches from related work and test them on the same datasets to get a direct comparison. All four real-life datasets were used for this purpose, however, for the Commodity12 only the User1 data was used (to avoid having to do separate comparisons for each user).

These approaches optimize different modalities (e.g., sensor subsets, duty-cycle lengths) based on different input (e.g., current context, underlying data) and as such they may not be directly comparable. For the ease of comparison we list to which of our methods each of them is most similar to, based on the previously described parameters. Comparing any of them to one of our method combinations can be considered unfair as the combinations optimize more than one modality. Note, however, that the ease of combining different optimization modalities is one of the strengths of our approach.

In all cases, the source code for these methods was not public, so we re-implemented them to the best of our ability from their descriptions in the corresponding published papers.

5.3.1 Simple Approach

We started with the approach presented in a few different works [32][53] that treats each context in isolation. For each one it decides what sensor settings are *best* for it and uses that setting when such context is detected. Such solutions are expected to retain the classification accuracy and slightly decrease the energy consumption (due to using *best* setting for each context, the expected consumption would still be high).

We tested two different variations of defining what the *best* setting is. The first was to select the setting that maximizes the accuracy for each activity ($CM_{i,i}$ for activity i and confusion matrix CM). The second was to maximize the accuracy of a classifier that classifies every instance to either activity i or *other*. The approaches are labeled as *Simple A* and *Simple B*, respectively, and are shown in Figure 5.43. These approaches were the basis for the Simple model used for comparison to our model throughout Section 5. The solutions generated in this way are those that have the predicted highest classification accuracy using that model.

This approach is directly comparable to the SCA method from the viewpoint of its optimization modalities. From the results it can be seen that in half the cases at least one of the two solutions is as good as the most energy-expensive SCA solution. The results are inconsistent, however, with Simple A being better in some datasets and Simple B in others. In one of the datasets they are both worse than static solutions. Furthermore, this approach finds only one (two if counting both A and B) solution – as the authors have not specified any mechanism to construct the whole Pareto front.

A possible approach to generating the Pareto front of different trade-offs would be to use the same genetic algorithm as with the SCA method. Instead of using the SCA model as the approximation of the assignment quality, we would use the Simple model instead. This approach failed as the returned front usually consisted of only one point with bad performance; due to the bad prediction accuracy of the Simple model there was usually one assignment that was grossly miss-approximated and wrongly predicted to dominate every other.

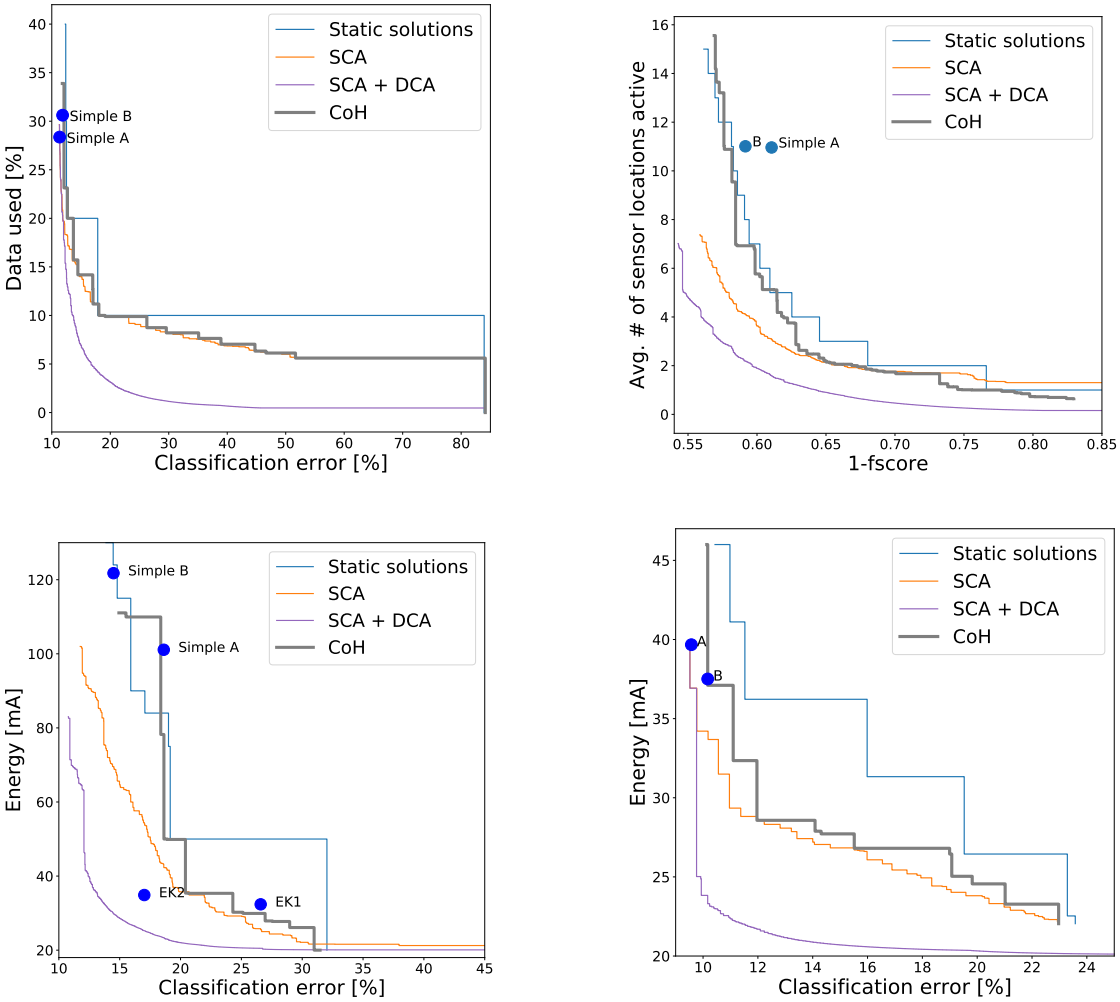

Figure 5.43: Comparison of our methods to three alternatives. Simple A and Simple B (or just A and B) mark the results from Section 5.3.1. EK1 marks the result from expert knowledge rules from Section 5.3.5, while the EK2 marks the result from the rules after improvement. All tests were performed on four datasets, with the exception of the expert knowledge approach. Top-left: SHL, top-right: Opportunity, bottom-left: Commodity12, bottom-right: E-Gibalec.

5.3.2 Settings Based on the Probable Next Context

The next approach [72] decides on which sensor subset to use based on which context (or which subset of contexts) is most likely to appear next. The likeliness is estimated by assuming the context sequence is Markov (the same assumption is made in our approach). Once the likely contexts are decided, the sensor subsets are chosen in a similar fashion as with the previously described simple model.

For this approach to work, the system designer must choose two parameters: the acceptable accuracy loss [0–100%] and the number of likely contexts predicted [1–$|C|$]. Authors do not specify how to determine the parameters, so we conducted a thorough grid search of values in the previously specified range. Grid search also enables us to select non-dominated solutions and construct the resulting Pareto front approximation. The results are shown in Figure 5.43 where we labeled them *CoH* after the paper title (*activity recognition for Creatures of Habit*). The optimization modality is in this case again directly comparable to the SCA method. This is not immediately obvious as the SCA method chooses settings based on the current context, while the CoH method chooses based on the likely next context. However, under the Markov assumption, the likeliness of the next context is entirely dependent on the current context, making the search space of the *CoH* method a subset of the SCA search space.

To evaluate an assignment the paper authors propose the simulation, which we already estimated as much slower than the mathematical model. However, this is less relevant for the *CoH* method as it estimates far fewer assignments than the SCA method.

From the results one can see that the *CoH* method comes close to the performance of the SCA method and in some rare cases even outperforms it. In most cases, however, the SCA method returns better solutions (which is not entirely surprising given a larger search space and a more efficient search algorithm).

5.3.3 Episodic Sampling

Episodic sampling [75] is a method that changes the duty-cycle length based on the frequency of context transitions. If the context remains unchanged, the length is increased by a, if not, the length is divided by b. The a and b are the parameters of the system and were determined by a grid search of sensible values ($a \in [0–15]$, $b \in [0–1]$). Again all non-dominated solutions were kept.

This system is most comparable to the DCA method due to the fact that both of them optimize duty cycles. Comparison is not entirely direct, as using the DCA method, we always have the same duty-cycle length assigned if the same context is classified, whereas the episodic sampling can vary the duty-cycle length based on how frequently the context is changing. Unfortunately, we have not found any related work that can be directly comparable to the DCA methodology.

From the results (Figure 5.44) we can see that neither method really dominates the other. DCA is better in two datasets, worse in one and comparable in the last. Worse performance on the E-Gibalec dataset could be explained by this dataset having only few contexts which limits the adapting capabilities of the DCA method. However, a more general explanation is that the two methods have a different search space and as such their relative performance depends on the properties of the dataset. One could even consider that they can complement one another as the system designer can quickly choose which approach works better on their domain. One could even try to combine them, by searching for the a and b parameters using the genetic search.

Again no model for the solution quality estimation is given and the method relies on the simulation for making such estimation.

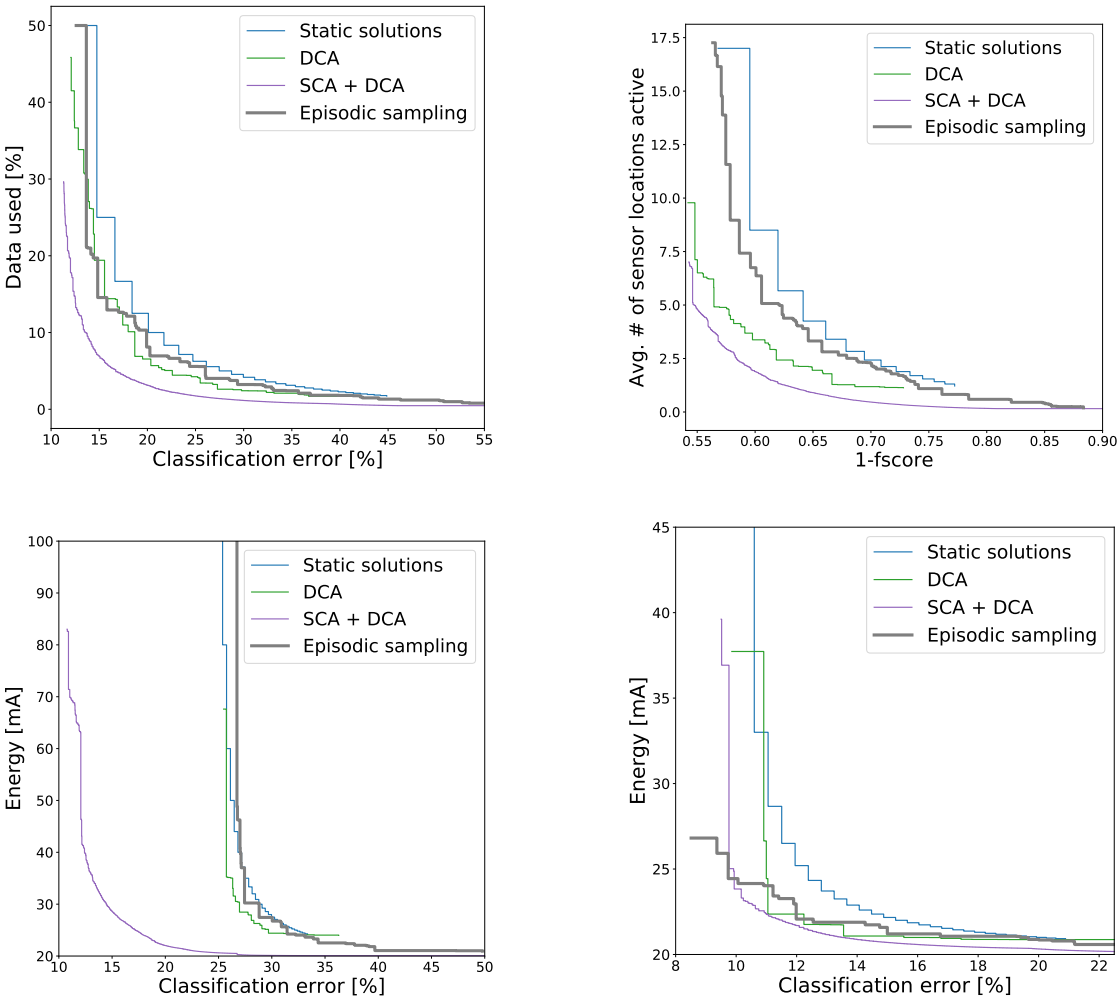

Figure 5.44: Comparison of our methods to the episodic sampling on four different datasets. Top-left: SHL, top-right: Opportunity, bottom-left: Commodity12, bottom-right: E-Gibalec.

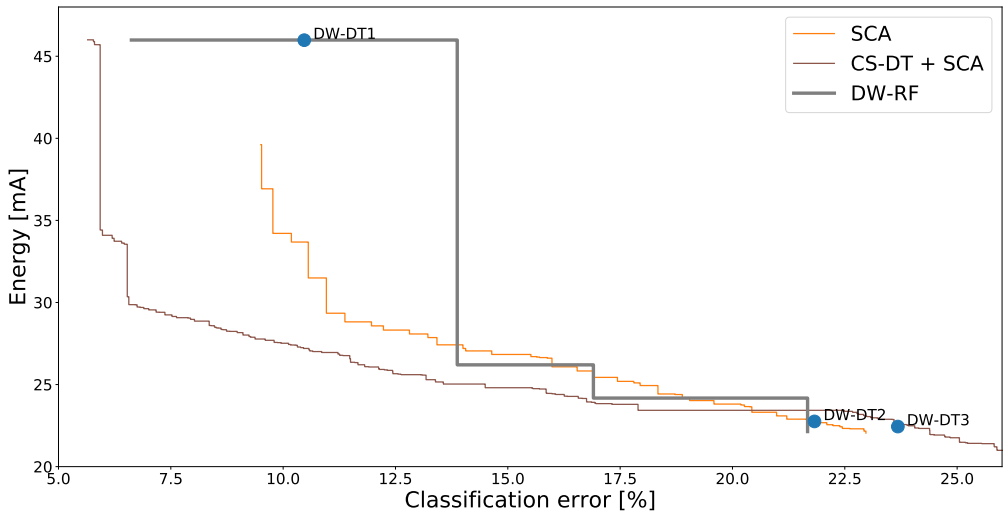

Figure 5.45: Comparison of our methods to the Datum-Wise Sampling Frequency method that uses either a decision tree (DW-DT) or a Random forest (DW-RF) classifier. This comparison was done on the E-Gibalec dataset.

5.3.4 Datum-Wise Sampling Frequency

Datum-Wise Sampling Frequency (DW) [82] is a method that trains a Markov decision process (MDP) to adapt the sampling frequency to both the feature values and to the last classified context.

This method first uses the training data to determine the optimal sampling frequency for each instance. Optimal is defined as the one that minimizes the sum of the misclassification cost and the energy cost weighted by λ. Then it uses Broyden-Fletcher-Goldfarb-Shanno optimization algorithm to determine the parameters of the MDP in order for it to be able to predict the optimal frequency given the feature and context data.

Since the DW uses both the context and feature values, we compared it to the CS-DT + SCA method. We used only E-Gibalec dataset for the comparison, as DW optimizes frequencies only and E-Gibalec dataset is the only one to use frequencies as settings. We tested different values of λ in order to obtain different trade-offs. Values of λ were determined by first trying different magnitudes (1, 0.1, 0.01, etc.) and then iteratively using bisection between values that produced different solutions.

We first used the decision tree as the classifier for the DW method, as it was also used for both the SCA and CS-DT methods on this dataset. Solutions found this way are marked as DW-DT in Figure 5.45. We managed to find only three distinct solutions this way, and thus marked them as points instead of the usual Pareto front approximation line. Then, to emphasise the weakness of our CS-DT method (it is limited to decision tree classifier only) we used the Random Forest classifier for the DW method. This combination was named DW-RF in Figure 5.45. Random Forest has a better classification accuracy on the E-Gibalec dataset, and the found solutions were improved as expected.

Nonetheless, the CS-DT + SCA combination outperformed both versions of the DW method by generating slightly more accurate solutions for the same energy cost, but more importantly, by generating a much better spread of different solutions. It is also of note that the DW was by far the most time consuming method of all methods tested in this work, needing approximately two hours to generate each individual solution. Finally, it was limited by being only able to optimize different frequencies.

5.3.5 Expert Knowledge

Lastly, we compared the strength of the automatically found solutions to those found by expert knowledge. Note, however, that quantifying the effectiveness of expert knowledge in creating energy-efficient solutions is always partly subjective. To alleviate some of the concern, we tried to create conditions that favour the expert as much as possible.

In the Commodity12 dataset the daily lives of participants' two weeks were recorded. One of the participants – which was also a machine-learning expert – was tasked with creating a system (by writing down expert-knowledge rules) that would work for classifying his weeks of data. Given that he was an expert in knowing his daily routine and had plenty of experience on how to interpret sensor data, we considered him the most qualified for the task. The rules could be in any form, optimizing whatever modality based on whatever type of data he liked. After creating the rules, they would be implemented and tested and then he could create another iteration of the rules based on the previous one's performance.

The results are shown in Figure 5.43. The original rules (labeled *EK1*) were better than baseline static solutions but were outperformed by any other method – which testifies to the difficulty of the task. The rules after the permitted iteration (labeled *EK2*) were much better, better than the SCA method but worse than the SCA + DCA combination (the most fair comparison, as the expert rules included both the duty-cycling and choosing the sensor subsets).

While this result is based on a very small sample, it serves to illustrate how non-trivial it is to construct expert-based rules, and why having an automated system that can do so is highly beneficial.

Chapter 6

Conclusion

In this work we developed three novel methods for reducing the energy consumption of a context recognition system. The first method (SCA) tries to assign a setting to each possible context and then uses that setting whenever the corresponding context is detected. If it is provided with the performance of the system when using each of the settings individually, it can predict the performance of the system when switching between the settings in runtime. The second method (DCA) does the same type of assignment, but can do so only for the duty-cycle settings. In exchange it does not need any actual experiments (to determine the performance of a single setting) and can predict the behavior resulting from duty-cycling directly from the dataset. The last method (CS-DT) creates cost-sensitive decision trees and uses them to adapt the settings based on the current feature data, and not on the current context. Such decision tree can use only a subset of its branches, depending on the informativeness and energy cost of the node features.

All three methods were also combined, joining their individual strengths. The three tested combinations were combining the SCA method with the DCA method, the SCA method with the CS-DT method and all three methods together.

These methods were systematically tested on four distinct real-life context recognition datasets and on an array of artificial ones. The datasets were chosen to be as diverse as possible, covering a large range of domain applications. Using them, we recognized different contexts – from basic activities, such as *walking*, to more complex ones, such as *working*. In addition, we used a wide array of different sensors and different functions that modeled sensor energy consumption. It is of note, however, that all the datasets presented a classification task and never a regression one. While the classification is more prevalent in context-recognition, we still revisit the issue in Section 6.3.

This diversity of datasets helped us to demonstrate the generality of the methods, as they performed well on all of them. In addition, they could work for many different types of settings that were up for optimization (which sensors subset to use, sampling frequency, duty-cycle length), and for different classification quality metrics (accuracy, F-score, recall of a specific activity). At no point in the testing was any expert knowledge provided and very few parameters had to be set in advance (which settings to use, what is their energy cost, the classification quality metric, tree construction method). This implies that this methodology could be used in many other different context-recognition problems without any adaptation or work needed on the part of the system designer.

All of the methods present a Pareto front approximation, consisting of many different trade-offs between the energy consumption and the classification quality. This can be highly beneficial for the system designer, both from the viewpoint of choosing the most suitable solution and from the viewpoint of understanding the landscape of possible trade-offs.

6.1 Performance

While many solutions were generated, we here list one sample solution for each dataset to serve as a reference for the efficiency of our methodology. These results were taken from the best performing method combination. For the SHL dataset we were able to use only 5% of the available data (by using lower frequency, duty-cycling and using a sensor subset) and in exchange sacrifice only 5% of the accuracy. For the Opportunity dataset we were able to find a solution that uses 4 sensors on average (instead of the original 30) and in exchange loses a minimal amount (0.01 points) of F-score. For the Commodity12 dataset we analysed four different users with somewhat different habits. Averaging the results of all four we decreased the smartphone's energy consumption from 123 mA to 29 mA. In exchange we lost less than 1 percentage point of accuracy in all cases. Finally, for the E-Gibalec dataset, energy was reduced from 46 mA to 24 mA for less than 1 percentage point of accuracy lost.

The three proposed methods each present a different angle of attack on the problem of reducing the energy consumption. As expected, optimizing the duty-cycles (DCA) was the most consistently performing modality across the datasets. When optimizing other types of settings, we compared which is better: optimizing based on the recognized context only (SCA) or based on the feature data only (CS-DT). Experiments showed that neither was consistently better – their relative performance being mostly dependant on the dataset. Their combination, however, was in all cases superior than using any of them separately.

Combining the SCA and DCA methods also consistently improved the results. This was expected, as they both optimized different types of settings. One could in theory use only the SCA method and included the duty-cycle settings in its set of settings to optimize. This would result in requiring many more experiments (one for each possible duty-cycle settings) for the same, if not worse results.

Using all three methods together in some cases increased the energy efficiency of the system, but in other cases yielded worse results than using a combination of two methods. This was unexpected and may deserve an explanation. The system using the CS-DT method acts in many ways as a flowchart. With every subsequent instance, the system travels up and down the tree nodes adapting its settings. Seeing more instances naturally leads to landing in a more appropriate state. In contrast, using duty-cycling, we want to classify one – or at least as few as possible – instances and then turn all the sensors off for as long as possible. These two objectives are conflicting and detrimental for each other. Having a short active period undermines the CS-DT's ability to decide which tree branches to use, while a long active period greatly reduces the beneficial effects of duty-cycling. The same problem would appear if using many other types of "flowchart" schemes in combination with duty-cycle optimization.

Comparison with the related work is difficult in this domain, as different researchers optimize different optimization modalities, using different system schemes and different degrees of expert knowledge. We picked five related works that appeared the most directly comparable, and through experimentation showed that we compare favorably to all of them. Of note, one duty-cycle optimization method [75] optimized based on frequency of context change (which is different than our methods) and in some cases achieved better results. It is often true that two methods that differ in type of the optimization will outperform each other depending on the type of the dataset used.

We also believe that our method is more general and suitable for a wider range of application domains than many other related works. In addition, note that different approaches to energy-efficient solutions are not necessarily conflicting, but can be often seen as complementing one another. If for example, one related work presents different energy-efficient classifiers, they can be used alongside our proposed optimization or as settings for it.

6.2 Hypotheses, Goals and Scientific Contributions

We started this thesis with two hypotheses:

Hypothesis 1 proposed that performance of a setting-to-context assignment can be mathematically modeled. We showed this by testing our proposed SCA model on all datasets and listing the difference between the predictions of the mathematical model, compared to the values achieved in a simulation. On the artificial dataset with the *Markov* property the predictions were effectively exact – deviating from the simulated values by roughly 0.05 percentage points. On the real-life datasets the predictions were in an acceptable range of deviating from the simulated values by roughly 1-2 percentage points of accuracy on average. The biggest deviation was observed on the Commodity12 dataset (roughly 5 percentage points) which we suspect is the result of some contexts appearing very infrequently in the dataset, making them hard to model correctly. Furthermore, we developed a mathematical model for predicting the duty-cycling on a system that needs even less data from actual experiments. Finally, we created models that predict the performance of method combinations. The accuracy of these models was very similar to the accuracy of the SCA model. The DCA model was slightly more consistent, not having large errors on any particular dataset. In addition we showed that using these mathematical models is much faster than using a simulation (10-100 times on tested datasets). The difference in speed gets larger if the dataset is larger, and gets smaller if there are many contexts in the dataset. The second part of this hypothesis stated that by using the mathematical models we can find energy-efficient solutions that are better than those found by methods in related work. This was confirmed by experiments in Section 5.3 that show our methods compare favourably on all tested datasets.

Hypothesis 2 proposes that adapting the system settings to both the predicted context and to feature values improves the energy-efficiency of the system. This was again experimentally proven by comparing the results of our SCA method (adapts to predicted context) to the CS-DT + SCA method (adapts to both context and data). Comparison was done on three datasets, one artificial and two real-life, as those datasets could be accurately classified by a decision tree. In all cases the CS-DT + SCA method returned better solutions: 2–8 percentage points better accuracy, depending on the dataset, for the same energy consumption decrease.

In addition, Section 1.3 lists the expected goals to be performed in this thesis. We believe to have accomplished all the set goals. **1)** The general method for adapting setting to contexts is described in Section 3.1. **2)** The method specialized in determining duty-cycle parameters is described in Section 3.2. **3)** The method that can adapt to both the context and underlying sensor data is described in Section 3.3. **4)** A mathematical model was created for the first two methods, their description is in the same section as the respective method description. **5)** All listed methods (and as part of that, their mathematical models) were combined in Section 3.4 and experimentally validated in Section 5.

As an unwritten goal we wanted the created methods to be as general and easy to use as possible. To achieve this goal, we will publish the used library, written in Python, on the webpage: `https://github.com/VitoJanko/Energy-Efficient-Context-Recognition`. This library contains all the proposed methods in addition to the three methods from the related work that we used for comparison. To use it, one needs to provide the dataset, a list of possible settings, the system performance using such settings, and their energy cost. Using that information, the energy-efficient solutions are automatically found using one of the specified methods. We hope that such a library will be useful for both other context-recognition researchers and developers of real-life applications.

The main contribution of this work is the development of a general framework for energy-efficient context recognition that outperforms others that have a similar level of

generality. We also made the specific scientific contributions listed in Section 1.4.

1) We believe to be the first to have successfully mathematically modeled the performance of a system that uses a setting-to-context assignment. Accuracy and speed of this model was already discussed as part of proving Hypothesis 1. **2)** We showed that the methodology from multi-objective optimization can be very useful for the task of searching for energy-efficient solutions. As previously discussed, these solutions are better than solutions from the related work. **3)** Lastly, we have adapted the cost-sensitive decision tree so it can be used in the domain of context recognition.

6.3 Limitations and Future Work

The first notable limitation of our methodology is its lack of focus on regression problems (e.g., the location coordinates of the user). Currently, the CS-DT could be easily adapted to using regression trees instead. More substantial work will be required to adapt the SCA and DCA methods for the task. A possible research direction for achieving this task would be to either find a way to discretize continuous context values or to rely more on the feature values (as CS-DTs does). In some cases where contexts of interest are discrete, the proposed methodology still faces problems. An example would be the "fall detection" problem, where we aim to detect someone's fall in order to call for help. In this case we have two different contexts *falling* and *not-falling*, but the former is so short and infrequent that any setting-to-context setup will struggle to find good energy-efficient solutions. On the opposite side of the spectrum, having too many contexts (25+) can make our mathematical models too slow, undermining their effectiveness.

The second limitation observed from the experiments comes from the CS-DT design. It relies entirely on decision trees to make the classification. In many domains this classifier is insufficient and thus the methodology cannot be used. Furthermore, the CS-DT architecture suffers when combined with duty-cycle optimization. A possible research direction to amend both of the problems is to create very shallow trees with an arbitrary classifier recognizing the instances that fall in leaf nodes, but they do so using different settings. This is in a way similar to how linear model trees [112] work – having the tree structure, but then using a linear model in their leaves.

Additional research direction would be trying to adapt to the stability of the context, i.e. change sensing settings based on whether the context is rapidly changing or is being stable for a long time.

Finally, we acknowledge that the research problem is very complex, and is definitely not completely solved by our contribution. There are always additional avenues of exploration and new approaches to the problem, each with its own strength and weaknesses.

References

[1] *Smartwatch popularity*, https://www.hodinkee.com/articles/us-smartwatch-sales-are-soaring-2019.

[2] *Samsung Health*, https://www.samsung.com/us/support/owners/app/samsung-health.

[3] *iOS - Health*, https://www.apple.com/lae/ios/health/.

[4] *Runkeeper*, https://runkeeper.com/.

[5] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh, "A framework of energy efficient mobile sensing for automatic user state recognition," in *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services*, ACM, 2009, pp. 179–192.

[6] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[7] D. Brockhoff, T. Friedrich, and F. Neumann, "Analyzing hypervolume indicator based algorithms," in *International Conference on Parallel Problem Solving from Nature*, Springer, 2008, pp. 651–660.

[8] Ö. Yürür, C. H. Liu, Z. Sheng, V. C. Leung, W. Moreno, and K. K. Leung, "Context-awareness for mobile sensing: A survey and future directions," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 68–93, 2016.

[9] O. D. Lara and M. A. Labrador, "A survey on human activity recognition using wearable sensors," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1192–1209, 2013.

[10] *Heartman*, http://www.heartman-project.eu/.

[11] Ö. Kafalı, S. Bromuri, M. Sindlar, T. van der Weide, E. Aguilar Pelaez, U. Schaechtle, B. Alves, D. Zufferey, E. Rodriguez-Villegas, M. I. Schumacher, *et al.*, "Commodity 12: A smart e-health environment for diabetes management," *Journal of Ambient Intelligence and Smart Environments*, vol. 5, no. 5, pp. 479–502, 2013.

[12] *Running & Jogging*, https://play.google.com/store/apps/details?id=com.zeopoxa.fitness.running/.

[13] M.-H. Park, J.-H. Hong, and S.-B. Cho, "Location-based recommendation system using bayesian user's preference model in mobile devices," in *International Conference on Ubiquitous Intelligence and Computing*, Springer, 2007, pp. 1130–1139.

[14] G. Aldaz, S. Puria, and L. J. Leifer, "Smartphone-based system for learning and inferring hearing aid settings," *Journal of the American Academy of Audiology*, vol. 27, no. 9, pp. 732–749, 2016.

[15] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: Rich monitoring of road and traffic conditions using mobile smartphones," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, ACM, 2008, pp. 323–336.

[16] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt, "Micro-blog: Sharing and querying content through mobile phones and social participation," in *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, ACM, 2008, pp. 174–186.

[17] G. Slapničar, N. Mlakar, and M. Luštrek, "Blood pressure estimation from photoplethysmogram using a spectro-temporal deep neural network," *Sensors*, vol. 19, no. 15, p. 3420, 2019.

[18] M. Gjoreski, H. Gjoreski, M. Luštrek, and M. Gams, "Continuous stress detection using a wrist device: In laboratory and real life," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, ACM, 2016, pp. 1185–1193.

[19] T. K. Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, IEEE, vol. 1, 1995, pp. 278–282.

[20] J. R. Quinlan, "Improved use of continuous attributes in c4.5," *Journal of Artificial Intelligence Research*, vol. 4, pp. 77–90, 1996.

[21] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine Learning*, vol. 6, no. 1, pp. 37–66, 1991.

[22] T. Chen, T. He, M. Benesty, V. Khotilovich, and Y. Tang, "Xgboost: Extreme gradient boosting," *R package version 0.4-2*, pp. 1–4, 2015.

[23] L. Wang, H. Gjoreski, K. Muraom, T. Okita, and D. Roggen, "Summary of the Sussex-Huawei locomotion-transportation recognition challenge," in *Proceedings of the 6th International Workshop on Human Activity Sensing Corpus and Applications*, Springer, 2018, pp. 1521–1530.

[24] E. L. Murnane, D. A. Huffaker, and G. Kossinets, "Mobile health apps: Adoption, adherence, and abandonment," in *Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*, ACM, 2015, pp. 261–264.

[25] T. Rault, A. Bouabdallah, Y. Challal, and F. Marin, "A survey of energy-efficient context recognition systems using wearable sensors for healthcare applications," *Pervasive and Mobile Computing*, vol. 37, pp. 23–44, 2017.

[26] A. Krause, M. Ihmig, E. Rankin, D. Leong, S. Gupta, D. Siewiorek, A. Smailagic, M. Deisher, and U. Sengupta, "Trading off prediction accuracy and power consumption for context-aware wearable computing," in *Ninth IEEE International Symposium on Wearable Computers*, IEEE, 2005, pp. 20–26.

[27] J. Lee and J. Kim, "Energy-efficient real-time human activity recognition on smart mobile devices," *Mobile Information Systems*, vol. 2016, 2016.

[28] H. Junker, P. Lukowicz, and G. Troster, "Sampling frequency, signal resolution and the accuracy of wearable context recognition systems," in *Eighth International Symposium on Wearable Computers*, IEEE, 2004, pp. 176–177.

[29] U. Maurer, A. Smailagic, D. P. Siewiorek, and M. Deisher, "Activity recognition and monitoring using multiple sensors on different body positions," Carnegie Mellon School of Computer Science, Tech. Rep., 2006.

[30] A. Khan, N. Hammerla, S. Mellor, and T. Plötz, "Optimising sampling rates for accelerometer-based human activity recognition," *Pattern Recognition Letters*, vol. 73, pp. 33–40, 2016.

[31] E. Walton, C. Casey, J. Mitsch, J. A. Vázquez-Diosdado, J. Yan, T. Dottorini, K. A. Ellis, A. Winterlich, and J. Kaler, "Evaluation of sampling frequency, window size and sensor position for classification of sheep behaviour," *Royal Society Open Science*, vol. 5, no. 2, p. 171 442, 2018.

[32] V. Q. Viet, H. M. Thang, and D. Choi, "Adaptive energy-saving strategy for activity recognition on mobile phone," in *2012 IEEE International Symposium on Signal Processing and Information Technology*, IEEE, 2012, pp. 95–100.

[33] A. J. Jerri, "The Shannon sampling theorem—its various extensions and applications: A tutorial review," *Proceedings of the IEEE*, vol. 65, no. 11, pp. 1565–1596, 1977.

[34] V. Janko, N. Reščič, M. Mlakar, V. Drobnič, M. Gams, G. Slapničar, M. Gjoreski, J. Bizjak, M. Marinko, and M. Luštrek, "A new frontier for activity recognition: The sussex-huawei locomotion challenge," in *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, ACM, 2018, pp. 1511–1520.

[35] X. Qi, M. Keally, G. Zhou, Y. Li, and Z. Ren, "AdaSense: Adapting sampling rates for activity recognition in body sensor networks," in *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium*, IEEE, 2013, pp. 163–172.

[36] L. Zheng, D. Wu, X. Ruan, S. Weng, A. Peng, B. Tang, H. Lu, H. Shi, and H. Zheng, "A novel energy-efficient approach for human activity recognition," *Sensors*, vol. 17, no. 9, p. 2064, 2017.

[37] Y. Wang, B. Krishnamachari, Q. Zhao, and M. Annavaram, "The tradeoff between energy efficiency and user state estimation accuracy in mobile sensing," in *International Conference on Mobile Computing, Applications, and Services*, Springer, 2009, pp. 42–58.

[38] S. Bosch, R. Marin-Perianu, P. Havinga, and M. Marin-Perianu, "Energy-efficient assessment of physical activity level using duty-cycled accelerometer data," *Procedia Computer Science*, vol. 5, pp. 328–335, 2011.

[39] B. Cvetković, V. Janko, A. E. Romero, Ö. Kafalı, K. Stathis, and M. Luštrek, "Activity recognition for diabetic patients using a smartphone," *Journal of Medical Systems*, vol. 40, no. 12, p. 256, 2016.

[40] O. Yurur, C. H. Liu, X. Liu, and W. Moreno, "Adaptive sampling and duty cycling for smartphone accelerometer," in *2013 IEEE 10th International Conference on Mobile Ad-Hoc and Sensor Systems*, IEEE, 2013, pp. 511–518.

[41] K. K. Rachuri, C. Mascolo, M. Musolesi, and P. J. Rentfrow, "Sociablesense: Exploring the trade-offs of adaptive sampling and computation offloading for social sensing," in *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking*, ACM, 2011, pp. 73–84.

[42] D. H. Kim, Y. Kim, D. Estrin, and M. B. Srivastava, "Sensloc: Sensing everyday places and paths using less energy," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, ACM, 2010, pp. 43–56.

[43] I. Constandache, S. Gaonkar, M. Sayler, R. R. Choudhury, and L. Cox, "Enloc: Energy-efficient localization for mobile phones," in *The 28th IEEE Conference on Computer Communications*, IEEE, 2009, pp. 2716–2720.

[44] M. Azizyan, I. Constandache, and R. Roy Choudhury, "SurroundSense: Mobile phone localization via ambience fingerprinting," in *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking*, ACM, 2009, pp. 261–272.

[45] N. Vallina-Rodriguez and J. Crowcroft, "ErdOS: Achieving energy savings in mobile os," in *Proceedings of the Sixth International Workshop on MobiArch*, ACM, 2011, pp. 37–42.

[46] Y.-S. Lee and S.-B. Cho, "Activity recognition using hierarchical hidden Markov models on a smartphone with 3d accelerometer," in *International Conference on Hybrid Artificial Intelligence Systems*, Springer, 2011, pp. 460–467.

[47] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell, "Sensing meets mobile social networks: The design, implementation and evaluation of the cenceme application," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, ACM, 2008, pp. 337–350.

[48] L. Gao, A. K. Bourke, and J. Nelson, "Activity recognition using dynamic multiple sensor fusion in body sensor networks," in *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, IEEE, 2012, pp. 1077–1080.

[49] H. Noshadi, F. Dabiri, S. Meguerdichian, M. Potkonjak, and M. Sarrafzadeh, "Energy optimization in wireless medical systems using physiological behavior," in *Wireless Health 2010*, ACM, 2010, pp. 128–136.

[50] P. Zappi, C. Lombriser, T. Stiefmeier, E. Farella, D. Roggen, L. Benini, and G. Tröster, "Activity recognition from on-body sensors: Accuracy-power trade-off by dynamic sensor selection," in *European Conference on Wireless Sensor Networks*, Springer, 2008, pp. 17–33.

[51] A. Carroll, G. Heiser, *et al.*, "An analysis of power consumption in a smartphone.," in *USENIX Annual Technical Conference*, Boston, vol. 14, 2010, pp. 21–21.

[52] V. Janko and M. Luštrek, "Using markov chains and multi-objective optimization for energy-efficient context recognition," *Sensors*, vol. 18, no. 1, p. 80, 2018.

[53] Z. Yan, V. Subbaraju, D. Chakraborty, A. Misra, and K. Aberer, "Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach," in *2012 16th International Symposium on Wearable Computers*, IEEE, 2012, pp. 17–24.

[54] J. W. Cooley, P. A. Lewis, and P. D. Welch, "The fast Fourier transform and its applications," *IEEE Transactions on Education*, vol. 12, no. 1, pp. 27–34, 1969.

[55] V. Könönen, J. Mäntyjärvi, H. Similä, J. Pärkkä, and M. Ermes, "Automatic feature selection for context recognition in mobile devices," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 181–197, 2010.

[56] R. Cilla, M. A. Patricio, A. Berlanga, and J. M. Molina, "Creating human activity recognition systems using pareto-based multiobjective optimization," in *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*, IEEE, 2009, pp. 37–42.

[57] D. Anguita, A. Ghio, L. Oneto, F. X. Llanas Parra, and J. L. Reyes Ortiz, "Energy efficient smartphone-based activity recognition using fixed-point arithmetic," *Journal of Universal Computer Science*, vol. 19, no. 9, pp. 1295–1314, 2013.

[58] L. Wang, T. Gu, X. Tao, and J. Lu, "A hierarchical approach to real-time activity recognition in body sensor networks," *Pervasive and Mobile Computing*, vol. 8, no. 1, pp. 115–130, 2012.

[59] V. Srinivasan and T. Phan, "An accurate two-tier classifier for efficient duty-cycling of smartphone activity recognition systems," in *Proceedings of the Third International Workshop on Sensing Applications on Mobile Phones*, ACM, 2012, p. 11.

[60] S. Nath, "ACE: Exploiting correlation for energy-efficient and continuous context sensing," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, ACM, 2012, pp. 29–42.

[61] S. Kang, J. Lee, H. Jang, Y. Lee, S. Park, and J. Song, "A scalable and energy-efficient context monitoring framework for mobile personal sensor networks," *IEEE Transactions on Mobile Computing*, vol. 9, no. 5, pp. 686–702, 2010.

[62] X. Fafoutis, L. Marchegiani, A. Elsts, J. Pope, R. Piechocki, and I. Craddock, "Extending the battery lifetime of wearable sensors with embedded machine learning," in *2018 IEEE 4th World Forum on Internet of Things*, IEEE, 2018, pp. 269–274.

[63] H. Kalantarian, C. Sideris, B. Mortazavi, N. Alshurafa, and M. Sarrafzadeh, "Dynamic computation offloading for low-power wearable health monitoring systems," *IEEE Transactions on Biomedical Engineering*, vol. 64, no. 3, pp. 621–628, 2017.

[64] M. M. Masud, M. A. Serhani, and A. N. Navaz, "Resource-aware mobile-based health monitoring," *IEEE Journal of Biomedical and Health Informatics*, vol. 21, no. 2, pp. 349–360, 2017.

[65] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ACM, 2010, pp. 49–62.

[66] B. Priyantha, D. Lymberopoulos, and J. Liu, "Littlerock: Enabling energy-efficient continuous sensing on mobile phones," *IEEE Pervasive Computing*, vol. 10, no. 2, pp. 12–15, 2011.

[67] H. Lu, A. B. Brush, B. Priyantha, A. K. Karlson, and J. Liu, "SpeakerSense: Energy efficient unobtrusive speaker identification on mobile phones," in *International Conference on Pervasive Computing*, Springer, 2011, pp. 188–205.

[68] *Amazon Alexa*, `https://developer.amazon.com/alexa`.

[69] *OK Google*, `https://ok-google.io/`.

[70] M. B. Kjærgaard, J. Langdal, T. Godsk, and T. Toftkjær, "Entracked: Energy-efficient robust position tracking for mobile devices," in *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services*, ACM, 2009, pp. 221–234.

[71] N. Lokare, S. Samadi, B. Zhong, L. Gonzalez, F. Mohammadzadeh, and E. Lobaton, "Energy-efficient activity recognition via multiple time-scale analysis," in *2017 IEEE Symposium Series on Computational Intelligence*, IEEE, 2017, pp. 1–7.

[72] D. Gordon, J. Czerny, and M. Beigl, "Activity recognition for creatures of habit," *Personal and Ubiquitous Computing*, vol. 18, no. 1, pp. 205–221, 2014.

[73] H. Sagha, S. T. Digumarti, J. d. R. Millán, R. Chavarriaga, A. Calatroni, D. Roggen, and G. Tröster, "Benchmarking classification techniques using the Opportunity human activity dataset," in *2011 IEEE International Conference on Systems, Man, and Cybernetics*, IEEE, 2011, pp. 36–40.

[74] S. Sarker, A. K. Nath, and A. Razzaque, "Tradeoffs between sensing quality and energy efficiency for context monitoring applications," in *2016 International Conference on Networking Systems and Security*, IEEE, 2016, pp. 1–7.

[75] L. K. Au, M. A. Batalin, T. Stathopoulos, A. A. Bui, and W. J. Kaiser, "Episodic sampling: Towards energy-efficient patient monitoring with wearable sensors," in *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, IEEE, 2009, pp. 6901–6905.

[76] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell, "The Jigsaw continuous sensing engine for mobile phone applications," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, ACM, 2010, pp. 71–84.

[77] C. Jiang, "Sampling frequency optimization and training model selection for physical activity classification with single triaxial accelerometer," Master's thesis, Tampere University of Technology, Faculty of Natural Sciences, 2015.

[78] F. Ben Abdesslem, A. Phillips, and T. Henderson, "Less is more: Energy-efficient mobile sensing with senseless," in *Proceedings of the 1st ACM Workshop on Networking, Systems, and Applications for Mobile Handhelds*, ACM, 2009, pp. 61–62.

[79] T. Park, J. Lee, I. Hwang, C. Yoo, L. Nachman, and J. Song, "E-gesture: A collaborative architecture for energy-efficient gesture recognition with hand-worn sensor and mobile devices," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, ACM, 2011, pp. 260–273.

[80] A. Rahmati and L. Zhong, "Context-based network estimation for energy-efficient ubiquitous wireless connectivity," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 54–66, 2011.

[81] J. Paek, J. Kim, and R. Govindan, "Energy-efficient rate-adaptive GPS-based positioning for smartphones," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ACM, 2010, pp. 299–314.

[82] W. Cheng, S. Erfani, R. Zhang, and R. Kotagiri, "Learning datum-wise sampling frequency for energy-efficient human activity recognition," in *Thirty-Second AAAI Conference on Artificial Intelligence*, AAAI, 2018.

[83] L. K. Au, A. A. Bui, M. A. Batalin, and W. J. Kaiser, "Energy-efficient context classification with dynamic sensor control," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 6, no. 2, pp. 167–178, 2012.

[84] G. Thatte, V. Rozgic, M. Li, S. Ghosh, U. Mitra, S. Narayanan, M. Annavaram, and D. Spruijt-Metz, "Optimal allocation of time-resources for multihypothesis activity-level detection," in *International Conference on Distributed Computing in Sensor Systems*, Springer, 2009, pp. 273–286.

[85] X. Chai, L. Deng, Q. Yang, and C. X. Ling, "Test-cost sensitive naive bayes classification," in *Fourth IEEE International Conference on Data Mining*, IEEE, 2004, pp. 51–58.

[86] M. Kukar, I. Kononenko, *et al.*, "Cost-sensitive learning with neural networks," in *European Conference on Artificial Intelligence*, IOS Press, 1998, pp. 445–449.

[87] S. Lomax and S. Vadera, "A survey of cost-sensitive decision tree induction algorithms," *ACM Computing Surveys*, vol. 45, no. 2, p. 16, 2013.

[88] C. X. Ling, Q. Yang, J. Wang, and S. Zhang, "Decision trees with minimal costs," in *Proceedings of the Twenty-First International Conference on Machine learning*, ACM, 2004, p. 69.

[89] V. Janko and M. Luštrek, "Energy-efficient data collection for context recognition," in *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*, ACM, 2017, pp. 458–463.

[90] V. Janko and M. Luštrek, "Markov chain model for energy-efficient context recognition," in *Slovenian Conference on Artificial Intelligence: Proceedings of the 19th International Multiconference Information Society*, Jožef Stefan Institute, 2016, pp. 28–31.

[91] V. Janko and M. Luštrek, "Choosing duty-cycle parameters for context recognition," in *2018 14th International Conference on Intelligent Environments*, IEEE, 2018, pp. 83–86.

[92] *E-Gibalec*, `https://www.e-gibalec.si`.

[93] *Commodity12*, `https://cordis.europa.eu/project/rcn/100711/factsheet/en/`.

[94] H. Gjoreski, M. Ciliberto, L. Wang, F. J. Ordonez Morales, S. Mekki, S. Valentin, and D. Roggen, "The university of Sussex-Huawei locomotion and transportation dataset for multimodal analytics with mobile devices," *IEEE Access*, pp. 42 592–42 604, 2018.

[95] *Sussex-Huawei Locomotion Challenge*, `http://www.shl-dataset.org/activity-recognition-challenge`.

[96] L. Wang, H. Gjoreski, M. Ciliberto, S. Mekki, S. Valentin, and D. Roggen, "Benchmarking the SHL recognition challenge with classical and deep-learning pipelines," in *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, ACM, 2018, pp. 1626–1635.

[97] M. Gjoreski, V. Janko, N. Reščič, M. Mlakar, M. Luštrek, J. Bizjak, G. Slapničar, M. Marinko, V. Drobnič, and M. Gams, "Applying multiple knowledge to sussex-huawei locomotion challenge," in *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, ACM, 2018, pp. 1488–1496.

[98] J. B. Kuipers, *Quaternions and rotation sequences*. Princeton university press Princeton, 1999, vol. 66.

[99] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[100] J. Shawe-Taylor and N. Cristianini, *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press Cambridge, 2000, vol. 204.

[101] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[102] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," Cornell Aeronautical Lab, Tech. Rep., 1961.

[103] *Opportunity*, `https://archive.ics.uci.edu/ml/datasets/opportunity+activity+recognition/`, Accessed: 2017-10-24.

[104] M. Luštrek, B. Cvetković, V. Mirchevska, Ö. Kafalı, A. E. Romero, and K. Stathis, "Recognising lifestyle activities of diabetic patients with a smartphone," in *2015 9th International Conference on Pervasive Computing Technologies for Healthcare*, IEEE, 2015, pp. 317–324.

[105] B. Cvetković, V. Janko, and M. Luštrek, "Demo abstract: Activity recognition and human energy expenditure estimation with a smartphone," in *2015 IEEE International Conference on Pervasive Computing and Communication Workshops*, IEEE, 2015, pp. 193–195.

[106] B. Cvetković, V. Mirchevska, V. Janko, and M. Luštrek, "Recognition of high-level activities with a smartphone," in *Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*, ACM, 2015, pp. 1453–1461.

[107] *Zephyr*, `https://www.zephyranywhere.com/`.

[108] I. Crk, F. Albinali, C. Gniady, and J. Hartman, "Understanding energy consumption of sensor enabled applications on mobile phones," in *Engineering in Medicine and Biology Society (EMBC), Annual International Conference of the IEEE*, IEEE, 2009, pp. 6885–6888.

[109] B. Cvetković, V. Janko, A. Gradišek, M. Luštrek, T. Kajtna, and B. Štrumbelj, "Mobile application to stimulate physical activity in schoolchildren," in *2016 12th International Conference on Intelligent Environments*, IEEE, 2016, pp. 206–209.

[110] V. Janko, B. Cvetković, A. Gradišek, M. Luštrek, B. Štrumbelj, and T. Kajtna, "E-Gibalec: Mobile application to monitor and encourage physical activity in schoolchildren," *Journal of Ambient Intelligence and Smart Environments*, vol. 9, no. 5, pp. 595–609, 2017.

[111] *scikit-learn*, `https://scikit-learn.org/stable/`.

[112] D. Potts and C. Sammut, "Incremental learning of linear model trees," *Machine Learning*, vol. 61, no. 1-3, pp. 5–48, 2005.

Bibliography

Publications Related to the Thesis

Journal Articles

V. Janko, B. Cvetković, A. Gradišek, M. Luštrek, B. Štrumbelj, and T. Kajtna, "E-Gibalec: Mobile application to monitor and encourage physical activity in schoolchildren," *Journal of Ambient Intelligence and Smart Environments*, vol. 9, no. 5, pp. 595–609, 2017.

B. Cvetković, V. Janko, A. E. Romero, Ö. Kafalı, K. Stathis, and M. Luštrek, "Activity recognition for diabetic patients using a smartphone," *Journal of Medical Systems*, vol. 40, no. 12, p. 256, 2016.

V. Janko and M. Luštrek, "Using markov chains and multi-objective optimization for energy-efficient context recognition," *Sensors*, vol. 18, no. 1, p. 80, 2018.

Book Chapters

V. Janko, M. Gjoreski, G. Slapničar, M. Mlakar, N. Reščič, J. Bizjak, V. Drobnič, M. Marinko, N. Mlakar, M. Gams, and M. Luštrek, "Winning the Sussex-Huawei locomotion-transportation recognition challenge," in *Human Activity Sensing*, Springer, 2019, pp. 233–250.

Conference Paper

B. Cvetković, V. Janko, A. Gradišek, M. Luštrek, T. Kajtna, and B. Štrumbelj, "Mobile application to stimulate physical activity in schoolchildren," in *2016 12th International Conference on Intelligent Environments*, IEEE, 2016, pp. 206–209.

V. Janko and M. Luštrek, "Markov chain model for energy-efficient context recognition," in *Slovenian Conference on Artificial Intelligence: Proceedings of the 19th International Multiconference Information Society*, Jožef Stefan Institute, 2016, pp. 28–31.

B. Cvetković, V. Mirchevska, V. Janko, and M. Luštrek, "Recognition of high-level activities with a smartphone," in *Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*, ACM, 2015, pp. 1453–1461.

B. Cvetković, V. Janko, A. Gradišek, M. Luštrek, T. Kajtna, and B. Štrumbelj, "Mobile application to stimulate physical activity in schoolchildren," in *2016 12th International Conference on Intelligent Environments*, IEEE, 2016, pp. 206–209.

V. Janko and M. Luštrek, "Choosing duty-cycle parameters for context recognition," in *2018 14th International Conference on Intelligent Environments*, IEEE, 2018, pp. 83–86.

V. Janko and M. Luštrek, "Cost-sensitive trees for energy-efficient context recognition," in *2019 15th International Conference on Intelligent Environments*, IEEE, 2019.

V. Janko and M. Luštrek, "Energy-efficient data collection for context recognition," in *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*, ACM, 2017, pp. 458–463.

M. Gjoreski, V. Janko, N. Reščič, M. Mlakar, M. Luštrek, J. Bizjak, G. Slapničar, M. Marinko, V. Drobnič, and M. Gams, "Applying multiple knowledge to sussex-huawei locomotion challenge," in *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, ACM, 2018, pp. 1488–1496.

V. Janko, N. Reščič, M. Mlakar, V. Drobnič, M. Gams, G. Slapničar, M. Gjoreski, J. Bizjak, M. Marinko, and M. Luštrek, "A new frontier for activity recognition: The sussex-huawei locomotion challenge," in *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, ACM, 2018, pp. 1511–1520.

Other Publications

Journal Articles

B. Cvetković, R. Szeklicki, V. Janko, P. Lutomski, and M. Luštrek, "Real-time activity monitoring with a wristband and a smartphone," *Information Fusion*, vol. 43, pp. 77–93, 2018.

B. Cvetković, H. Gjoreski, V. Janko, B. Kaluža, A. Gradišek, I. Jurinčič, A. Gosar, S. Kerma, G. Balažič, and M. Luštrek, "E-Turist: An intelligent personalised trip guide," *Informatica*, vol. 40, no. 4, pp. 447–455, 2016.

V. Janko and M. Guid, "A program for progressive chess," *Theoretical Computer Science*, vol. 644, pp. 76–91, 2016.

Conference Paper

V. Janko, A. Stensbye, and M. Luštrek, "Predicting office's ambient parameters," in *Slovenian Conference on Artificial Intelligence: Proceedings of the 20th International Multiconference Information Society*, Jožef Stefan Institute, 2017, pp. 83–86.

V. Janko, N. Mlakar, and B. Jani, "BRISCOLA : Being resourceful in stacking cards - opponent, lament away!" In *Slovenian Conference on Artificial Intelligence: Proceedings of the 21th International Multiconference Information Society*, Jožef Stefan Institute, 2018, pp. 29–32.

M. Luštrek, B. Fele, N. Reščič, and V. Janko, "Counting bites with a smart watch," in *Slovenian Conference on Artificial Intelligence: Proceedings of the 22th International Multiconference Information Society*, Jožef Stefan Institute, 2019, pp. 49–52.

B. Cvetković, V. Janko, and M. Luštrek, "Demo abstract: Activity recognition and human energy expenditure estimation with a smartphone," in *2015 IEEE International Conference on Pervasive Computing and Communication Workshops*, IEEE, 2015, pp. 193–195.

M. Gjoreski, V. Janko, H. Gjoreski, B. Cvetkovć, M. Luštrek, and M. Gams, "Activity and stress monitoring using smartphone and wrist device," in *7th Jožef Stefan international postgraduate school students' conference*, Jožef Stefan Institute, 2016, pp. 154–164.

V. Janko, M. Gjoreski, C. M. De Masi, N. Reščič, M. Luštrek, and M. Gams, "Cross-location transfer learning for the Sussex-Huawei locomotion recognition challenge," in *Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*, ACM, 2019, pp. 730–735.

V. Janko and M. Guid, "Development of a program for playing progressive chess," in *Advances in Computer Games*, Springer, 2015, pp. 122–134.

B. Cvetković, V. Janko, and M. Luštrek, "Demo abstract: Activity recognition and human energy expenditure estimation with a smartphone," in *Pervasive Computing and Communication Workshops*, IEEE, 2015, pp. 193–195.

Biography

Vito Janko was born on December 18, 1990 in Koper, Slovenia. He finished his primary and secondary education in Piran, Slovenia. In 2009, he started his studies at the interdisciplinary program shared between the Faculty of Computer and Information Science and the Faculty of Mathematics and Physics, both part of the University of Ljubljana. In 2012, he obtained a BSc degree by defending his BSc thesis entitled "Aktivno učenje". He continued his Master's studies at the same program, and finished with a theses entitled "1-2-3 Šah". For his Master's thesis he received the Prešeren award.

During his Master's study he started working as a student at the Department of Intelligent Systems at the Jožef Stefan Institute, Ljubljana, Slovenia. There he collaborated on EU-funded projects Commodity12 (COntinuous Multi-parametric and Multi-layered analysis Of DIabetes TYpe 1 & 2) and Fit4Work as well as on national projects e-Turist and e-Gibalec.

After finishing his Master's study in 2015 he continued working at the Jožef Stefan Institute as a young researcher under the supervision of Dr. Mitja Luštrek. At the same time he enrolled in the PhD program entitled "Information and Communication Technologies" at the Jožef Stefan International Postgraduate School, Ljubljana, Slovenia. His PhD thesis topic was selected due to the frequent problem with the battery life of the recording devices in basically all projects where he previously collaborated.

While working on his thesis he still helped with the ongoing context-recognition projects, most importantly being the first author for the submissions at the SHL competition where he achieved second place in 2018 and first place in 2019. Meanwhile he was also active in the Slovenian Artificial Intelligence Society (SLAIS), where he acted as a member of the executive board and a secretary and treasurer in 2018.