MASTER

Frugal learning
applying machine learning with minimal resources

Evchenko, M.V.

Award date:
2016

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
Frugal Learning: Applying Machine Learning with Minimal Resources

Master Thesis

Mikhail Evchenko

Supervisor:

Dr. ir. Joaquin Vanschoren

Assessment committee members:

Dr. ir. Michel Westenberg
Dr. ir. Pierluigi Casale (Philips Lighting Research)

Final version

Eindhoven, August 31, 2016
Abstract

The main contribution of this work is the development of a new metric that can be used for the comparison of machine learning algorithms based on predictive performance and the amount of resources required for the proper working of an algorithm. The name of the new metric is Frugality score. Based on this score, one can choose the most appropriate algorithm for the specific task. This task can be run in different environments with different amounts of available resources, such as CPU performance, available RAM, storage, and battery, that require different algorithms. Therefore, the choice of an algorithm based on such important information can be done in a straightforward manner with Frugality score, which helps with obtaining the best result.

The selection of algorithms that can be used in the application developed for a wearable device with limited computational resources is based on Frugality scores for a number of machine learning algorithms and meta-features obtained from a vast range of data sets. These data sets were divided in two clusters, and Pareto Front analysis helped to identify the best algorithms in terms of AUC and time required to build and test a classifier. With the help of heat maps and Frugality curves, three algorithms with the most attractive properties have been identified. Namely, these algorithms are AdaBoost.M1, A1DE, and the dagging. They were selected based on results collected from more than 500 data sets. The first algorithm shows outstanding performance but requires a significant amount of time to produce a result. In contrast, the dagging is a fast algorithm with lower performance than AdaBoost.M1. The third algorithm, A1DE, is a trade-off between performance and time and can be used in practice for a task in which demand for resources can be traded for a small decrease in performance. These algorithms are used in an Android application running on a smartwatch that performs human activity recognition (HAR), and their results were compared to each other in terms of AUC and battery consumption.

The application for performing machine learning classification on a consumer smartwatch with an Android operating system was developed from scratch. It does not require connection to the Internet and performs activity recognition based on data collected from built-in sensors. All implementations of machine learning algorithms are transferred from the Weka library.

Data about six types of activities were collected from 20 participants. Machine learning classifiers were trained on these data and used for HAR. The A1DE algorithm was trained in the shortest period of time. However, this classifier showed the highest battery consumption while having the worst performance, while the dagging algorithm showed almost the same battery consumption but a considerably better performance as that of A1DE. Finally, AdaBoost.M1 performed the best and used the lowest battery. As a result, AdaBoost.M1 can be recommended for HAR on wearable devices such as smartwatches for achieving optimum results in terms of performance and battery consumption. The dagging could be a proper choice for applications that use activity recognition for a long period of time and can afford inaccuracies for a moderate share of classification results. The third algorithm, A1DE, should be used for tasks in which classifier should be trained in the smallest time, and data are presented by a vector of nominal attributes.
Preface

This work reflects research I conducted in the area of machine learning for my thesis from Eindhoven University of Technology. I want to thank my supervisor, Joaquin Vanschoren, for his support and openness for discussion at any time of the day or night. I also would like to thank Pierluigi Casale for his valuable advice about data collection and activity recognition on wearable devices. The user interface of mobile application follows guidelines from Google but was influenced by my professor in Visualization, Michel Westenberg.

One of the central parts of this project is a data collection experiment. Many people were kind and agreed to spend time performing different physical exercises. I want to express my gratitude to Emin, Dato, Bin, Nan, Ilaha, Sakina, Shamil, Nderim, Rob, Stephanie, Cagil, Kiana, Sanand, Kirsten, Ankur, Hilda, Nikita, Haoshi, and Christopher. Many of you said that lying on a couch was the best that you can imagine in the middle of the day, but no one complained about walking upstairs for a long period of time.

I want to express gratitude to my family who supported me for two years while I was studying in the Netherlands. You are a really important part of my life and encourage me to achieve results and become better every day. I have never felt alone while being so far from home, because you called me and did everything to make me happy. Thank you for your patience and smart ideas while I was busy with the project.
Contents

Contents vii

List of Figures xi

List of Tables xiii

Listings xv

1 Introduction 1

1.1 Frugal learning context ........................................... 1
1.2 Research problem ..................................................... 2
1.3 Document outline ..................................................... 3

2 Preliminaries 5

2.1 Literature study: machine learning in constrained environments .............. 5
2.2 Literature study: metrics for algorithm evaluation .......................... 6
2.3 Literature study: activity recognition .................................... 7
2.4 Data for analysis ......................................................... 9

3 Analysis of algorithms with Frugality score 11

3.1 Definition of a new measure ............................................. 12
3.2 Universal approach for different performance and resource demands evaluations ...... 14
3.3 Clustering ................................................................. 15
  3.3.1 Identifying a structure ............................................. 15
  3.3.2 Compute a number of clusters ................................. 16
| CONTENTS |
|------------------|------------------|
| 3.3.3 Visualizing clustering | 17 |
| 3.3.4 Analyzing features of clusters | 19 |
| 3.3.5 Selecting data sets for studying a frugality of algorithms | 20 |
| 3.4 Pareto analysis | 20 |
| 3.4.1 Explanation of the Pareto Front method | 21 |
| 3.4.2 Pareto Fronts for clusters of data sets | 22 |
| 3.4.3 Results of analysis with Pareto Front method | 23 |
| 3.5 Heat maps and frugality score lines | 24 |
| 3.5.1 SVD | 24 |
| 3.5.2 What information can we see on heat maps | 25 |
| 3.5.3 Introducing Frugality curves and choosing algorithms for evaluation | 26 |
| 4 Activity recognition on a wearable device | 30 |
| 4.1 Smartwatch as an example of a wearable device | 30 |
| 4.2 Details of implementation | 31 |
| 4.3 User interface | 33 |
| 4.4 Collection of data in experiment | 37 |
| 4.4.1 Performed activities | 37 |
| 4.4.2 Data submission and storage once its collected | 37 |
| 4.5 Data preparation | 38 |
| 4.6 Analysis of performance and battery consumption | 42 |
| 5 Conclusions | 46 |
| 5.1 Results and discussion | 46 |
| 5.2 Future work | 47 |
| 5.2.1 Performance optimization | 47 |
| 5.2.2 API for external applications | 48 |
| 5.2.3 Personalization of standard machine learning models | 48 |
| Bibliography | 49 |
| Appendix | 55 |

Frugal Learning: Applying Machine Learning with Minimal Resources
A Data protocol

A.1 Purpose of the data collection ........................................... 55
A.2 Equipment ................................................................. 55
A.3 Environmental setup ..................................................... 56
A.4 Data format ............................................................... 56
A.5 Forms and documents ................................................... 56
  A.5.1 Challenges and solutions .......................................... 57
  A.5.2 Impartiality, privacy and safety .................................. 57

B Algorithms used in the study .............................................. 58

C Missing results for classification tasks per an algorithm .......... 60
## List of Figures

2.1 *ARR* with three different values for AccD (0.2, 0.3 and 0.7) ........................................ 7

2.2 *A3R′* for three different settings for the n-th root (4, 8, and 16) .............................. 7

2.3 Typical Activity Recognition Chain (ARC) to recognize activities from wearable sensors. An ARC comprises of stages for data acquisition, signal preprocessing and segmentation, feature extraction and selection, training, and classification. Raw signals ($D$) are first processed ($D'$) and split into $m$ segments ($W_i$) from which feature vectors ($X_i$) are extracted. Given features ($X_i$), a model with parameters scores $c$ activity classes $Y_i = y_1, \ldots, y_c$ with a confidence vector $p_i$. ........................................ 8

3.1 The example of Frugality curves constructed for LibSVM with radial basis kernel function, decision tree j48 (implementation of C 4.5 in Weka) with confidence threshold for pruning set to 0.25 and minimum number of instances per leaf equal to 2, and Random Forest with 100 trees. Value of $w$ reflects a demand for resources, and a higher value means a more constrained environment. A Frugality score shown on the figure for a particular algorithm is an average of all individual scores computed for this algorithm and every data set in a study. The number before the name of an algorithm in the legend is an ID of the algorithm on OpenML.org website. .......................... 14

3.2 The Silhouette clustering for meta-features of data sets. The highest value $s(i)$ is computed for 2 clusters. ................................................................. 16

3.3 PCA visualization for data sets. ................................................................. 17

3.4 The t-SNE visualization for data sets. ................................................ 18

3.5 The t-SNE visualization with additional noise. ........................................ 18

3.6 Pareto Front with the names of algorithms for the data set “722_pol”. The x axis contains results for AUC, and y axis contains log of build and test time for a classifier. 21

3.7 Pareto Front for the first cluster. .................................................. 22

3.8 Results of RealAdaBoost and AdaBoost.M1 for the first cluster. .................... 23

3.9 Pareto Front for the second cluster. .................................................. 24

3.10 Dendrogram for data sets created with five latent features. ............................. 26
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.11</td>
<td>Frugality scores heat map constructed with $w$ equal to 0.1. This value represents weak resource constraints for a task. Algorithms are sorted based on AUC performance showed in the previous section. The lowest performing algorithms can be found on the left, and the top performers are shown on the right. Data sets follow the order obtained with a dendrogram introduced in the beginning of this section.</td>
</tr>
<tr>
<td>3.12</td>
<td>Frugality scores heat map constructed with $w$ equal to 0.5. This value represents moderate resource constraints for a task. Algorithms are sorted based on AUC performance showed in the previous section. The lowest performing algorithms can be found on the left, and the top performers are shown on the right. Data sets follow the order obtained with a dendrogram introduced in the beginning of this section.</td>
</tr>
<tr>
<td>3.13</td>
<td>Frugality scores heat map constructed with $w$ equal to 1.0. This value represents extended resource constraints for a task. Algorithms are sorted based on AUC performance showed in the previous section. The lowest performing algorithms can be found on the left, and the top performers are shown on the right. Data sets follow the order obtained with a dendrogram introduced in the beginning of this section.</td>
</tr>
<tr>
<td>3.14</td>
<td>Frugality curves for best the performing and the fastest algorithms computed with selected with PAM method data sets.</td>
</tr>
<tr>
<td>4.1</td>
<td>Coordinate system of a mobile device.</td>
</tr>
<tr>
<td>4.2</td>
<td>Introduction screen with two available options for a user to choose from.</td>
</tr>
<tr>
<td>4.3</td>
<td>The main screen of data collection mode.</td>
</tr>
<tr>
<td>4.4</td>
<td>Selection of activity for data collection with icons representing types of activities.</td>
</tr>
<tr>
<td>4.5</td>
<td>Screen of the application during data collection about <em>standing</em> activity.</td>
</tr>
<tr>
<td>4.6</td>
<td>Screen of the application recognizing <em>sitting</em> activity.</td>
</tr>
<tr>
<td>4.7</td>
<td>Confirmation dialog for closing the application.</td>
</tr>
<tr>
<td>4.8</td>
<td>Types of activities performed by the participants of the data collection experiment.</td>
</tr>
<tr>
<td>4.9</td>
<td>Activity recognition workflow in the experiment.</td>
</tr>
<tr>
<td>4.10</td>
<td>General importance of raw features for predicting a target variable calculated by Random Forest.</td>
</tr>
<tr>
<td>4.11</td>
<td>Importance of raw features for predicting each individual target variable computed by the learning vector quantization classifier.</td>
</tr>
<tr>
<td>4.12</td>
<td>Battery consumption measured for the dagging, A1DE, and AdaBoost.M1 classifiers.</td>
</tr>
<tr>
<td>5.1</td>
<td>Doze provides a recurring maintenance window for apps to use the network and handle pending activities.</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Names of the excluded from the study algorithms and the amount of unsuccessful runs per algorithm. ........................................ 9

3.1 Properties of data sets ............................................... 15
3.2 Mean values of features .............................................. 19
3.3 Median values of features ............................................ 20

4.1 Sensors that are used for data collection about human activities . .......... 32
4.2 Correlations between sensors based on raw data, part one. Green cell connotes that two variables have an absolute value of pair-wise correlation higher than 0.75. A light blue cell denotes a feature that should be excluded from the analysis. .... 40
4.3 Correlations between sensors based on raw data, part two. Green cell connotes that two variables have an absolute value of pair-wise correlation higher than 0.75. A light blue cell denotes a feature that should be excluded from the analysis. .... 41
4.4 Performance of classifiers for HAR. ................................. 43
4.5 Confusion matrix for the dagging with 10-fold cross-validation. ............. 43
4.6 Confusion matrix for A1DE with 10-fold cross-validation. ..................... 43
4.7 Confusion matrix for AdaBoost.M1 with 10-fold cross-validation. ............. 44
Listings

4.1 Command to pull data from Android device ............................................. 32
4.2 Receiving a path to a folder with sensor data on external storage. ............. 32
4.3 Setting memory limit for Gradle that allows building project with Weka library. . 33
Scientists have been using machine learning techniques for an increasing number of tasks in recent years. The growth of computational resources and declining prices for hardware offer opportunities for regular customers that earlier were available only for large institutions and well-funded research projects. With advances in algorithms and methodologies, it is now possible to apply machine learning techniques for such sophisticated tasks as autonomously driving cars or identifying objects in pictures and video materials.

1.1 Frugal learning context

Miniaturization allows for making chips smaller and using them in different environments. However, the smallest and cheapest devices have insufficient computational power, a lack of storage resources, small battery capacity, and sometimes limited connection to networks. Despite being less powerful than computers and servers, these devices have the ability to perform machine learning tasks.

Tasks in machine learning are generally divided in three main categories: supervised learning, unsupervised learning and reinforcement learning [64]. This work studies a discovery process for the identification of the optimal algorithm for a classification task. Such a task can be considered a subclass of supervised learning. Considering that a large number of algorithms have been implemented for performing classification in machine learning, identifying the optimal one could be a challenge.

In the literature, algorithms used for classification tasks are mainly compared by performance measurements such as accuracy [51] or area under the ROC curve (AUC) [35]. However, for some tasks a small decrease in performance can be traded for much faster building time for a classifier or economical usage of a battery. Therefore, a new method that takes into account performance and resources will help identify the most appropriate algorithm for every machine learning task accomplished in a constrained environment.

The choice of the best algorithm for a specific task on a constrained device is a two-goal optimization problem. It can be solved by taking into consideration the projected performance of an algorithm and the amount of resources that an algorithm requires for completing a task. Frugality score is a new tool designed to evaluate an algorithm based on both objectives. It can be computed for each algorithm based on a wide range of diverse tasks and allows one to choose the most appropriate algorithms for a task just by looking at the values of this new measurement.
CHAPTER 1. INTRODUCTION

One of the most important features of Frugality score is its ability to work with any combination of performance and resource evaluations. In our study we chose AUC as a performance indicator and time as a resource evaluation measure for calculating a Frugality score, but they can easily be substituted with other measures that are suitable for a specific task. Another advantage of Frugality score is that it can be visualized with Frugality curves, and it is possible to build a curve that shows performance of a specific algorithm for a different set of constraints. In addition, this algorithm can be easily compared with other algorithms by looking at a plot with Frugality curves or comparing Frugality scores values for specific constraints.

When a new measure such as Frugality score is defined and presented, the next step is to prove that it works with a real case study. As a demonstration of our approach, a proof of concept in the form of an Android application was developed. The application is designed for a wearable device such as a smartwatch and performs HAR. This task has been well studied by researchers [2], [11], [16], [47], [70] on smartphones, but to the best of our knowledge no research has been done using a watch that provides extended information due to its position on the wrist and the presence of a heart rate sensor on the back side. The data for training classifiers were collected as a part of this project from a group of 20 people who performed a predefined set of tasks. Participants wore a smartwatch on their wrists, and data from the device’s built-in sensors were written to an embedded file storage. Trained models created with different classifiers were used for activity recognition. The results for each of them are shown in Chapter 4 which presents information about the frugal machine learning application.

1.2 Research problem

The main goal of this work is an analysis of machine learning algorithms from a frugal learning prospective. It means that algorithms should be compared not only by their performance but also by the amount of resources that they require. As a result, two main parts should be completed.

- Creating a formula for Frugality score.
- Ranking of algorithms based on this new measure for a specific set of constraints.

The first goal is research when a new measure should be introduced. The second is to use this measure for analyse results collected from an extensive set of experiments. This analysis should lead to a selection of algorithms that can be used in the application developed as a proof of concept. To achieve these goals, the next set of questions should be answered.

- How can performance and the amount of resources be combined together in one measure?
  The first research question reflects that a new formula that combines performance and the amount of required resources should be defined. This could be done by analyzing related work, identifying drawbacks of existing solutions, and defining a new formula designed for machine learning algorithms with respect to both these criteria.

- What indicator can be chosen for the performance part of Frugality score?
  The second question can be answered by taking into account that many performance measures exist and each of them works for the evaluation of an algorithm for a specific task. However, in general, AUC performs well and allows us to evaluate the performance of an algorithm providing higher discriminancy and consistency than accuracy or other measures [52]. The detailed explanation of choosing AUC as a performance component for the Frugality score formula is given in Section 3.1.
CHAPTER 1. INTRODUCTION

- What resource ought to be taken for the comparison of algorithms based on Frugality score?
  Selection of a resource for the Frugality score formula is an important question. It reflects
  the type of constraints that a researcher wants to evaluate. This work uses the time required
  to train and test a classifier as a resource. A given time allows us to compare algorithms
directly and incorporates some other resources such as CPU or speed of working with RAM
and storage.

- Which methods should be used for the comparison of algorithms?
  To perform a comparison, the data used for the analysis should be obtained and prepared. At
  the first step, data sets were downloaded from the OpenML.org website [73]. They contain
information about different domains ranging from biological data to MBA grades. Data sets
were analyzed and clustered in such a way that new clusters contain data sets with different
properties. Grouping data sets in clusters allows us to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for
each cluster. Algorithms on these plots were used to identify the best algorithms for diverse
sets of tasks with similar characteristics. As a result, a Pareto Front was constructed for

- What types of algorithms can be used for evaluating the correctness of hypotheses in a
  practical task?
  Three algorithms from distinct classes could be selected for this purpose: one with the best
  performance, one with a low demand for resources, and one that presents a trade-off between
  both optimization goals. One from this final class can be considered as the optimal solution.

- Can a proof of concept be developed that shows the accuracy of previous work?
  The development of the application for a HAR task is an important part of this project. As
stated, previous work for this task has been done primarily on smartphones. These devices
have specifications similar to those for desktops several years ago and cannot be considered as
truly constrained devices. The recently introduced type of device is a consumer smartwatch.
It has an energy-efficient CPU, a small amount of RAM, a few GBs of data storage, and,
very importantly, a battery that has only a fraction of a smartphone’s capacity. Therefore,
a smartwatch has all the properties of a constrained device. Algorithms selected based on
Frugality score were ported to a device, and machine learning for HAR showed whether
original assumptions about the suitability of these algorithms for working in constrained
environment were correct.

The scope of the project is relatively broad and covers topics from constructing a new
measure for comparing machine learning algorithms to developing software that can read data
from sensors and perform machine learning with limited resources. Therefore, research questions
should be studied in the right order. As a result, it should allow us to perform a theoretical study
that can be proven with a practical evaluation. The itemized structure of this report is shown in
the next section.

1.3 Document outline

This document is structured as follows. Sections 2.1 and 2.3 give an overview about related
work for performing machine learning in constrained environments and HAR on mobile devices.
Section 2.4 provides information about data sets used for the comparison of algorithms. The data
protocol presented in Appendix A explains how data about activities, used for evaluating the
performance of algorithms with a real machine learning task, were collected from a study group
with help from a developed application that runs on a smartwatch.
CHAPTER 1. INTRODUCTION

The Section 3.1 presents the Frugality score and shows that it can be used in various contexts. Our evaluation of algorithms for different constraints based on the new measure is shown in Sections 3.3-3.5. Information about the application developed for doing HAR on a smartwatch is included in Section 4. Finally, the conclusion summarizes results from previous sections and explains directions of future work.
Chapter 2

Preliminaries

This Section contains all the information that is required for a reader to be prepared for understanding the rest of the thesis. It covers related work in the field of machine learning with scarce resources, presents alternative metrics for a comparison of machine learning algorithms, and gives an overview for HAR. The former is important since resource constraints and ways of doing research in constrained environments closely related to the first project goal: defining the Frugality score formula. Studying metrics that other researchers proposed for evaluation of algorithms is helpful for understanding advantages and drawbacks of existing solutions. The latter presents a number of works about current state-of-the-art in the area of human activity recognition and helpful for developing the application for this task based on the previous research. This application should follow guidelines established by other scientists in this field and use the set of algorithms selected based on Frugality score results.

Information about data used for studying algorithms and calculating Frugality scores for these algorithms is given in the Section 2.4. This part discusses in details the source of data, amount of data sets and properties of data.

2.1 Literature study: machine learning in constrained environments

The problem of scarce or frugal resources has been studied by a number of researchers. It could be seen from different perspectives such as "computational power or memory optimization, insufficient data for building a model, oracle resources required to build a hypothesis, achieving the best performance in a constraint environment, or defining a measure that could help compare algorithms based on more than one parameter" [19]. As stated in [8], changes in every aspect related to frugality have influence on classification results.

As far as this author knows, frugal learning was covered in the literature primarily for supervised machine learning. In addition, HAR performed by the specially developed for this project application on a smartwatch is a classification task. Thus, only supervised algorithms are studied in this and other sections of the report from the perspective of constrained machine learning.

Extended research related to a problem of performance of algorithms for a specific task has been covered in the literature. For instance, [18] provided comparison in terms of performance...
CHAPTER 2. PRELIMINARIES

and resource consumption between several types of decision trees and methods such as bagging and boosting. Authors used decision trees in order to classify types of terrain on satellite images. The metric which they applied for analysis of computational resources is the “amount of basic operations”. In the case of decision trees, the amount of basic operations is the number of decision points in a tree or in a collection of trees in the case of bagging and boosting [18]. They identified that C5.0 type of Decision tree algorithm [57] requires less resources than bagging and significantly less than boosting, while keeping accuracy of a single model just few percent lower than accuracy of ensembles. However, a calculation of the number of decision points is applicable to trees while other types of algorithms use different approaches to build a classification model.

Performance of algorithms such as Bayesian Network [23], Naive Bayes [37] using Discretization (NBD), Naive Bayes with Kernel Density Estimation (NBK) that is able to use quantitative variables even if they are not normally distributed, C4.5 Decision Tree [58] and Naive Bayes Tree [43] for IP traffic flow classification was in [75]. The importance of this work for our study is in the fact that the given paper evaluated algorithms based not only on accuracy in classification but also on computational performance. All models used in the given paper were built with the help of Weka library [28]. Authors used building time and classification speed of the algorithms for performance comparison. According to the obtained results, the fastest algorithm for classification, i.e. the algorithm with the lowest testing time for a classifier, was C4.5 while NBK was the slowest one. However, when researchers investigated the building time for model, they discovered that NBK required less time than other algorithms to build a model, while NBTree required significantly more time for building a classifier.

One of the most intriguing questions for machine learning researchers is how to find the best algorithm for a particular data set [71]. As stated in [51], time required to train a classifier is largely ignored when comparison between machine learning approaches takes place. In this work, Lim et al. published results of experiments where 22 decision tree algorithms, nine classical and contemporary statistical algorithms and two neural networks were examined. In addition, 16 data sets with real data and 16 artificially created data sets with an addition of noise to the original ones were used in this study. According to authors, presented choice of data sets should reduce possible benefits for some algorithms that favor data with a specific set of features. Obtained results stated that mean error rate difference between the best algorithms was less than 0.012 and was insignificant in statistical and, probably, in practical terms. Conversely, the median time for model training varied from seconds to hours. The fastest algorithm in terms of computational time was C4.5 decision tree, while the group of slowest algorithms included POL [45] (POLYCLASS algorithm. As explained in [51], it fits a polytomous logistic regression model using linear splines and their tensor products.), FM2 [32] (flexible discriminant analysis), and RBF [65] (radial basis function network). Two of the slowest algorithms in the study of Lim et al. were spline-based and one, RBF, was a neural network.

2.2 Literature study: metrics for algorithm evaluation

To the best of our knowledge, a few works are related for combining several objectives in one metric that can be used for ranking and comparison of classifiers. A performance indicator that takes into consideration both time and performance was proposed in [10]. It is calculated with the next formula:

\[
ARR_{d_{p}, a_{q}} = \frac{SR_{d_{p}}}{SR_{a_{q}}} \times \frac{1 + AccD \times \log \left( \frac{r_{d_{p}}}{r_{a_{q}}} \right)}{1 + AccD \times \log \left( \frac{r_{d_{p}}}{r_{a_{q}}} \right)}
\] (2.1)
CHAPTER 2. PRELIMINARIES

In Formula 2.1, $SR_{a_p}^{d_i}$ and $T_{a_p}^{d_i}$ represent the success rate and time of algorithm $a_p$ on data set $d_i$, respectively, and $AccD$ represents the relative importance of accuracy and time, as defined by a user [10]. $AccD$ expresses amount of accuracy that user is willing to trade for 10 times a speedup or a slowdown. For instance, $AccD$ equals 10 percent means that the user is going to sacrifice 10 percent of accuracy for 10 times an acceleration or a slowdown. Therefore, it is possible to vary this parameter and rank algorithms according to a specific task.

$$A3R'_{a_p} = \frac{SR_{a_p}^{d_i}}{\sqrt{T_{a_p}^{d_i}}} \quad (2.2)$$

ARR has a shortcoming that the resulting function is not monotonic (Fig. 2.1). This property can lead to unreliable evaluation of an algorithm applicability for a specific task. Solution proposed in [1], [71] is $A3R'$ function defined with Formula 2.2. The plot of this metric is shown in Fig. 2.2. The meanings of $SR_{a_p}^{d_i}$ and $T_{a_p}^{d_i}$ in $A3R'$ are the same as for $AAR$. A newly introduced variable $r$ defines the importance of time for a task. The high value of $A3R'$ means that an algorithm fits right for this specific data set. Authors conducted a series of experiments, and proved that $A3R'$ function steadily declines along $\log RT$ axis. This inference means that new measure satisfies the criteria of monotonicity. Given property suggests that $A3R'$ can be used for a proper ranking of classifiers in a wide range of values for $r$ and overcomes the main drawback of $AAR$ indicator.

Figure 2.1: $ARR$ with three different values for $AccD$ (0.2, 0.3 and 0.7). Adapted from [1]

Figure 2.2: $A3R'$ for three different settings for the n-th root (4, 8, and 16). Adapted from [1]

2.3 Literature study: activity recognition

Human activity recognition has attracted attention in recent years with the increased popularity of smartphones and wearable devices. Many scientists have been trying to propose a framework for algorithm analysis that is precise and fast. Since the price of hardware is decreasing, more data is generated, stored, and this data becomes available for scientific analysis in such fields as security, sports, and healthcare.

Anguita et al. [2] conducted a research [61] that investigates the relation between the number of predictions for HAR made by a classification algorithm and battery life of a smartphone. Each value measured for activity recognition process encompasses all steps from reading sensor data to making a prediction about a type of activity. According to [2], one of the most significant factors that influences the performance and battery life is a format of measurements from sensors: they studied fixed-point results with 8 and 16 bit integers and floating-point values with 32 bits.
CHAPTER 2. PRELIMINARIES

When a custom written HAR application worked with 8 bit integer values, the smartphone (Samsung Galaxy S II) lasted approximately 100 percent longer compared to the same application working on the for floating-point measurements. However, Anguita et al. noted that this result could be hardware dependent, and more observations on different devices were required for better analysis.

Bulling et al. [11] provided an overview of 18 methods that were applied to different data sets with information about activities and identifies Activity Recognition Chain (Fig. 2.3) that is used for creating a model. This framework identifies five steps that should be done from obtaining raw data to building a classifier. Initially, data is collected from sensors. This form of data cannot be used for training a classifier, because it contains noise from sensors and using it can lead to overfitting with training data and low performance with previously unseen by classifier data.

When data is collected, it should be cleaned from noise and segmented in such a way that each new part of data set is likely to contain information about a type of activity. Segments can be chosen with or without overlap between each other. Then, feature extraction is applied to segmented data set, and a model that classifies activities can be built. Dividing data in segments is important since it allows engineering new features such as mean or standard deviation that are useful for distinction of similar activities. In addition, segmentation leads to a reduced amount of data that should be processed by a classifier. For instance, if only one new feature should be constructed from 32 raw measurements collected from one sensor with no overlap between segments, the size of data that should be processed by a classifier reduces 32 times. As can be guessed, given approach requires an extensive amount of raw data if one wants to achieve reasonable classification performance.

Figure 2.3: Typical Activity Recognition Chain (ARC) to recognize activities from wearable sensors. An ARC comprises of stages for data acquisition, signal preprocessing and segmentation, feature extraction and selection, training, and classification. Raw signals \( D \) are first processed \( D' \) and split into \( m \) segments \( W_i \) from which feature vectors \( X_i \) are extracted. Given features \( X_i \), a model with parameters scores \( c \) activity classes \( Y_i = y_1, \ldots, y_c \) with a confidence vector \( p_i \). Adapted from [11]

The combination of eyewear device with sensors and a smartwatch was proposed in [15] as a system that can detect falling activity for elderly. When the system recognizes such type of activity, both devices send information to the paired smartphone. Authors used two-step technique with weighted Extreme Learning Machine (ELM) classifier [81] for activity recognition. Fundamentally, ELM is a feedforward neural network with one “hidden” layer connected with inputs by randomly assigned weights and can be used in classification and regression tasks. Use of this fast, compared to other neural networks [34], algorithm allowed scientists to achieve 95.74 percent precision and 93.67 percent recall.

A number of researchers have been able to correctly classify more than 99 percent of activities. According to the paper [16], accuracy level equal to 99.7 percent was achieved for HAR...
when using Hidden Markov Models (HMM). This algorithm is out of scope for our Frugal learning project since it allows to obtain results that may outperform other classifiers but it is not frugal in terms of resources. In addition, the implementation of HMM in Weka library requires special representation of input data and cannot be used with the same data sets that were utilized for computing Frugality scores of other algorithms. Therefore, it was decided not to include HMM in this study.

Uddin et al. extended Activity Recognition Chain with decision fusion and performance evaluation with two additional steps. Authors of [70] applied a feature selection algorithm, namely, guided Random Forest, and built a classifier with extremely randomized trees that provides recognition rate up to 99.6 percent.

Data obtained from reference Android smartphone was used in [47] for activity recognition. Information from accelerometer was segregated in three dimensions, and authors studied to what degree acceleration along X, Y and Z dimension had influence on the final result. Kwon et al. processed obtained data with the multilayer perceptron classifier. According to the paper, system was able to identify walking, standing, and running activity with more than 99 percent accuracy.

Methods presented in this review that are able to achieve a high classification accuracy tend to consume a vast amount of training data and mainly require significant resources for building a model. The main purpose of doing HAR in our project is not to achieve the highest accuracy but to show that it is possible to perform machine learning on a wearable device with limited resources. In addition, we want to know how evaluation made with Frugality scores computed for general set of tasks matches real data obtained with developed application.

2.4 Data for analysis

A Frugality score is based on two parameters: performance measure and the amount of resources required to perform a task. It can be calculated only on per algorithm basis. However, the number of data sets can vary from one to many. The more data sets are used for calculating a score and more diverse they are, the better approximation of a Frugality score for a various set of tasks can be obtained.

This project uses data from OpenML.org server for algorithm analysis. Given website allows to share data sets among researchers, publish results for available data sets and even organize competitions between members. In other words, it is a reproducible research platform.

Table 2.1: Names of the excluded from the study algorithms and the amount of unsuccessful runs per algorithm.

<table>
<thead>
<tr>
<th>Name of algorithm</th>
<th>Missing values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1109_functions.GaussianProcesses</td>
<td>516</td>
</tr>
<tr>
<td>1072_rules.Prism</td>
<td>491</td>
</tr>
<tr>
<td>1086_trees.Id3</td>
<td>489</td>
</tr>
<tr>
<td>1102_misc.OSDL</td>
<td>486</td>
</tr>
<tr>
<td>1173_bayes.HNB</td>
<td>486</td>
</tr>
<tr>
<td>1181_functions.Winnow</td>
<td>480</td>
</tr>
<tr>
<td>1107_Jazy.LBR</td>
<td>461</td>
</tr>
</tbody>
</table>

The number of data sets that were downloaded from OpenML.org website is 516. They were collected from multiple domains varying from biological data to MBA grades. The size of these data sets differs from 10 to 98528 instances. The target can be a two-class or multi-class
value. A fraction of data sets contain missing values. Therefore, some algorithms had issues while processing them. Consequently, seven algorithms presented in Table 2.1 that could not produce classification results for more than half of tasks were removed from the study, and 96 were used for later stages of analysis.

The full list of algorithms that were not able to process at least ten data set initially was presented in [19] and is given in Appendix C.
Chapter 3

Analysis of algorithms with Frugality score

The selection of a classification algorithm by an expert for a particular data set is not trivial. There are myriads of practical issues to consider. Namely, a fraction of algorithms can work only with binary target variables [42], while others require a series of preprocessing steps [79] to achieve a high level of performance. A data set can be incomplete and have missing values that cause issues for algorithms that cannot make a classification based on an incomplete feature vector; therefore, it could be fruitful to think about the manner of imputation [48] for missing values.

Next, we need to rank algorithms based on how likely it is that they will be useful for the task at hand. Even in a situation where all mentioned issues are solved, there is usually not enough information about a data set to make a choice for the most appropriate machine learning algorithm. Performance ranking is frequently used to select an algorithm for a specific task. In general, performance can be expressed in terms of accuracy, precision, recall, or another measure [68]. However, considering only performance for making a choice about an algorithm for a particular task could be misguided for situations in which the available resources are limited. This is especially true for the growing trend of the Internet of things (IoT) in which devices with very limited capabilities are connected to large-scale distributed systems [3]. Another example of increasingly popular devices that have a limited amount of resources is wearable electronics, or wearables [76]. In most cases, a representative wearable device has a CPU, RAM, persistent storage, screen, and network connectivity. However, these devices tend to lack performance and available resources compared to desktops or servers due to their small form factor and price concerns. Consequently, the amount of accessible assets should be taken into account when performing a machine learning task on such devices.

The new metric that allows measuring both performance and the amount of resources required for performing a task is proposed in the beginning of this section and is called Frugality score. This name emphasizes the importance of a limited amount of resources for algorithms' comparison. What types of measures can be used for a Frugality score for real tasks are presented in Section 3.2.

The main goal of the remainder of this chapter is to study the behaviour of machine learning algorithms for various tasks from different viewpoints. The content of Section 3.3 discusses whether data sets can be organized in clusters. Because the variance between more than 500 data sets downloaded from OpenML.org is high, the frugality of a particular algorithm likely depends on some common properties of those data sets. An algorithm may be frugal for one data set but not
frugal in general. Hence, it makes sense to see whether there are inherent clusters and whether frugality is linked to properties of the data sets. Also, it makes sense to investigate whether some algorithms are never frugal and can, therefore, be removed to make the analysis clearer. When the assumption that data sets can be organized in clusters is verified, the next step is to discover the optimal number of clusters. The final step in this section is analysis of acquired clusters by studying their properties.

Each cluster has its own collection of data sets downloaded from OpenML platform. A Pareto Front [41] that allows one to identify the best set of algorithms in terms of AUC and combined time (this measure is a sum of training and test time for a classifier) can be constructed for each cluster. The given method allows us to find a group of optimal solutions. A solution presented on a Pareto Front cannot outperform another solution on this front for two objectives simultaneously. In other words, optimization of a task with respect to parameter $a$ has a negative outcome for parameter $b$ and contrariwise. If an algorithm appears at least once on Pareto Front for any cluster, it is added to a list of algorithms that can be studied at the later stages of analysis.

These algorithms are further evaluated with the construction of heat maps [4]. They are created for different values of $w$ selected so as to denote low, average, or high constraints in resources. Heat map are illustrations that help to identify changes in different environments while keeping the same set of algorithms and data sets. The second way of studying the frugality of algorithms is by creating Frugality curves. Presenting the frugality of an algorithm in a specified range, they allow identification of crossings and help to make the most appropriate choice while taking into account the behaviour of an algorithm and comparing it with alternative solutions.

Based on the conducted analysis in Sections from 3.3 to 3.5, three algorithms with different properties should be chosen for use in HAR on a wearable device.

### 3.1 Definition of a new measure

One of the main features planned when designing the Frugality score is the ability to directly compare algorithms. Taking this into account, the first effort was to fit the values produced by the new function in a range between $[-1, 1]$. However, researchers can choose any combination of measurements for performance and resources. As a result, we decided that the new metric should reflect a dependency between performance and resources while a range of values can be specific for a combination of measures chosen for a particular task. The formula 3.1 allows combining performance and resource measurement together, and the exact output range depends on the selection of predictive performance and resource usage metrics.

$$Frug_{a_j}^{d_i} = P_{a_j}^{d_i} - \frac{w}{1 + \frac{R_{a_j}}{R_{a_j}^c}} (3.1)$$

The area under the ROC curve or AUC [9] is used in this project as a measure for evaluating the performance of algorithms for a classification task. This measure is used for expressing performance as a $P_{a_j}^{d_i}$ in Formula 3.1. Originally, AUC was suitable only for binary classification tasks; however, a special method proposed in [29] named multi-class AUC overcame this issue. The property of AUC is rather robust against imbalances in the target feature. This characteristic is helpful for the study because a part of the data sets (among more than 500 used in the study) have asymmetry in the target feature.

The actual choice of measure for $P_{a_j}^{d_i}$ in the Formula 3.1 indicator shows the amount of required resources was based on the idea that Frugality score can be especially useful for constraint
devices where lack of battery life is an important concern. If a task requires intensive calculations and the CPU works at a maximum performance level for a long time, then the battery will be drained quickly. Accordingly, training and testing time for a classifier can be used as a resource measure, \( R_{d_{ij}} = T_{\text{train}} + T_{\text{test}} \), in the Frugality score formula for classifications tasks.

The derivation of \( \frac{w}{1 + \exp(\log(R_{d_{ij}}))} \) in Formula 3.1 could be explained in the next manner. Our goal is to fit a consumption of resources in a range between 0 and 1. This can be done by transforming raw value for \( R_{d_{ij}} \) using sigmoid function scaled to a range: \( \frac{1}{1 + \exp(-R_{d_{ij}})} \). However, since building or training time often grow exponentially, we can take a logarithm, yielding \( \frac{1}{1 + \exp(-\log(R_{d_{ij}}))} \). This expression is equivalent to the frugality definition for resources in Formula 3.1. Using AUC and a value for \( w \) between 0 and 1, a frugality score for selected measures will be a value between -1 and 1. Based on choices for \( P_{d_{ij}} \) and \( R_{d_{ij}} \), Formula 3.2 is used for constructing a Frugality score in this study.

\[
Frug_{d_{ij}} = AUC - \frac{w}{1 + \frac{T_{\text{train}} + T_{\text{test}}}{T_{\text{train}} + T_{\text{test}}}} 
\]  

(3.2)

Parameter \( w \) defines the constraints of an environment. If the task is all about performance and does not take into account available resources, \( w \) should be set to 0. When resource constraints are moderate, which can be a case for machine learning task performed on a smartphone or on a wearable device, the value of \( w \) can be set close to 0.5. Finally, when resources are highly limited, and the performance of a classifier is not a primary concern, value of \( w \) can be set to 1 or a higher value.

Dependency between \( w \) and Frugality score is unique for each algorithm. A line that shows the impact of frugal resources on the performance of an algorithm has the name **Frugality curve**. The example of a Frugality curve that connects all Frugality score values in the range \([0, 1]\) for \( w \) for three popular algorithms such as LibSVM [14], decision tree j48 [58], and Random Forest [50] is shown in Fig. 3.1.

Thus, the influence of time constraints on the LibSVM algorithm is significant, and it shows worse results than the two other algorithms for any value of \( w \). An especially interesting property of Frugality curves that can be seen at the Fig. 3.1 is an ability to form crossings between curves. When resource constraints are quite modest, Random Forest shows the best performance. However, starting from the point \( w = 0.3 \), j48 outperforms Random Forest and becomes the most prominent candidate for using in a machine learning task. As intersection point with x axis also could be a point of interest when considering what kind of algorithm to use in an environment with significant constraints. More information about Frugality curves can be found at Section 3.5.
3.2 Universal approach for different performance and resource demands evaluations

Previous section defines Frugality score and presents the idea that it can be used with any combination of performance and resource measures. Tasks targeting the maximum share of correct classifications for a classifier can use precision as the measure of performance. On the other hand, some tasks require proper recognition for a particular class. Therefore, recall can be more suitable for such a type of tasks. Other measures, such as F-measure [72] or Accuracy can also be considered as the performance indicator in Formula 3.1.

A type of a resource to use for the comparison of algorithms should be defined by a problem. Considering a situation where a consumption of the resource has to be optimized, e.g. battery level, measure $R_{di}$ can be expressed as the value of battery change. When resource is available but should be used exceptionally efficiently, e.g. memory storage, RAM-Hours [7], or CPU time, this measure can be expressed in the amount of a resource that an algorithm uses during a classification.

An introduction of a Frugality score enables a new perspective of the selection of algorithms for a specific task based on performance and available resources.
3.3 Clustering

The group of miscellaneous data sets selected for the experiment allows studying frugality of algorithms in diverse domains. The number of data sets in this group is 516: all of them\(^1\) were uploaded by researchers to the OpenML.org platform. The website automatically computes 129 characteristics\(^2\) for each added data set and allows us to view these data in browser or download through API. Analysis of all these characteristics can create additional complexity and may require careful handling of noise without adding a real value. Therefore, based on our assumptions about what types of data set features can be useful for analysis, we chose five for further study. The given set of characteristics is presented in Table 3.1. The following subsections use only these five meta-features for analysis and clustering of data sets.

Table 3.1: Properties of data sets

<table>
<thead>
<tr>
<th>Name of parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumAttributes</td>
<td>The number of attributes that define a target variable.</td>
</tr>
<tr>
<td>ClassEntropy</td>
<td>The amount of information required to correctly classify an instance.</td>
</tr>
<tr>
<td>MaxNominalAttDistinctValues</td>
<td>Maximum number of values for a categorical feature in data.</td>
</tr>
<tr>
<td>MajorityClassSize</td>
<td>The size of the majority class.</td>
</tr>
</tbody>
</table>

3.3.1 Identifying a structure

The first step in clustering analysis is a verification of an assumption that data has an internal structure and can be clustered. Otherwise, it can be that data points are randomly distributed in a feature space, and clustering makes no sense. To avoid that, a number of algorithms can be used for the identification of structure in data.

\[
H = \frac{\sum_{j=1}^{m} u_j^d}{\sum_{j=1}^{m} u_j^d + \sum_{j=1}^{m} w_j^d} \tag{3.3}
\]

In this study, the Hopkins statistic [5] method is used for the identification whether there is a structure in a meta-data feature space of data sets. The method works as following. To begin, it generates uniformly at random new data points in a feature space and computes, as shown in Formula 3.3, how these points fit into existing data structure. When distance between new and existing points is almost the same, the measure is around 0.5. The further a result from 0.5 the more evidently that the data has an internal structure. Experiments showed that for the points representing data sets in meta-data feature space the Hopkins statistic is equal to 0.032. This value shows that data clearly has a structure, and it is possible to proceed to the next step and cluster it.

\(^1\)http://www.openml.org/s/1
\(^2\)http://www.openml.org/search?q=+type%3Adata_quality&type=measure
### 3.3.2 Compute a number of clusters

One of the most well-known algorithms for clustering is K-means\[^{[31]}\]. It can create as many clusters from data points as user had set before running the algorithm. However, the quality of clustering without a prior knowledge about the number of clusters is questionable for our task. A better solution could be to determine and assess a possible number of clusters within the data and perform clustering with this knowledge afterwards.

For the identification of the number of clusters in a data, the Silhouette method \[^{[63]}\] is used in this study. This method computes a value \( s(i) \) that indicates to what extent points fit in the given number of clusters. If the value computed by Silhouette is high, it is expected that data can be separated in this amount of clusters. The highest value should be chosen as a parameter that can be passed to K-means for creating clusters from data.

\[
s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}
\]  \hspace{1cm} (3.4)

The Silhouette method uses two values for computing a final result. The first value, \( a(i) \), is the distance between a point and all other points within a cluster that it belongs to, and the second value, \( b(i) \), is the distance between this point and all other points in the nearest cluster. Value \( s(i) \) should be computed for each point in a data set and averaged. This method can demonstrate a sluggish level of performance for a very large data sets, but in case of clustering around 500 data points, it works with an acceptable efficiency.

![Figure 3.2: The Silhouette clustering for meta-features of data sets. The highest value \( s(i) \) is computed for 2 clusters.](image)

As can be seen from Fig. 3.2, the maximum for \( s(i) \) is equal to 0.579 when \( k \) is equal to 2. Therefore, we can use 2 as value for the \( k \) parameter when clustering data with K-means algorithm. Studying whether these two clusters can be divided further in subclusters, and what are the properties of these small clusters might be particularly interesting. Therefore, application
of the Silhouette method for studying whether two obtained clusters have an inner structure can be considered as a part of a future work.

### 3.3.3 Visualizing clustering

When clusters are constructed with the K-means algorithm, visualization of these clusters can be created for additional insight. A number of methods such as the Principal Component Analysis (PCA) method [39] or t-SNE [53] can be used for this purpose. Given methods work in a different manners and should produce distinctive images that can help us look at data from original perspectives. In addition, the resilience of t-SNE against noise is studied for a verification that the result that this method produces while visualizing our clusters is persistent and does not significantly change with negligible changes in data.

![PCA analysis](image)

**Figure 3.3:** PCA visualization for data sets.

Fig. 3.3 shows the output from the PCA method. The colour of every dot corresponds to a cluster with black for a cluster one and red for a cluster two. According to the figure, it is possible to make a linear separation between red and black dots. Therefore, we can assume that the clustering produced by K-means is meaningful. The distribution of points is also visible on this figure. The first cluster has a larger number of data sets and is denser than the second cluster.

The second visualization constructed with a novel algorithm called t-SNE also supports the idea that two clusters can be visually separated from each other. The big cloud of red dots representing the first cluster is located in the centre of the figure, while blue dots representing the second cluster are located at the top and bottom of the image.

Results for t-SNE are computed in two stages. First, t-SNE constructs a probability distri-
CHAPTER 3. ANALYSIS OF ALGORITHMS WITH FRUGALITY SCORE

Figure 3.4: The t-SNE visualization for data sets.

Figure 3.5: The t-SNE visualization with additional noise.
bution over pairs of high-dimensional objects in such a way that similar objects have a high probability of being picked, whilst dissimilar points have an infinitesimal probability of being picked. Second, t-SNE defines a similar probability distribution over the points in the low-dimensional map, and it minimizes the Kullback-Leibler divergence [46] between the two distributions with respect to the locations of the points in the map as stated in [53].

The method takes as an input a set of parameters such as a number of dimensions (two or three), a maximum number of iterations, and a perplexity value. These parameters can have an impact on the final visualization. In addition, minor changes in data can lead to obtaining a different visualization with the t-SNE method. Therefore, it is worth checking whether the separation between two clusters remains if the underlying data is slightly changed. Fig. 3.5 shows the result of a visualization when a new uniformly at random generated meta-feature was added to each data set. This figure has differences from the original image such as transformed shapes of clusters or new positions of blue data points representing data sets from the second cluster, but the differentiation between clusters can be made while looking at it. As a result, we can conclude that data has an inner structure, splitting of data sets into two clusters is the right choice, and this new clusters are persistent.

3.3.4 Analyzing features of clusters

Every data set on OpenML.org has more than 100 meta-features. As stated in the beginning of this section, only five of these are used as properties of data points for clustering algorithms. Grouping data sets together gives an opportunity to analyze combinations of these features in different ways. In this study, analysis is made for mean and median values for features presented in Table 3.1. Studying both median and mean values is important. A data set can have an outlier that has an impact on the mean value of a feature for a whole cluster but not influence the median value.

<table>
<thead>
<tr>
<th>Name of parameter</th>
<th>Cluster 1</th>
<th>Cluster 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumAttributes</td>
<td>30.457</td>
<td>1625.26</td>
</tr>
<tr>
<td>ClassEntropy</td>
<td>0.885</td>
<td>2.771</td>
</tr>
<tr>
<td>DecisionStumpAUC</td>
<td>0.723</td>
<td>0.73</td>
</tr>
<tr>
<td>MaxNominalAttDistinctValues</td>
<td>36.88</td>
<td>460.14</td>
</tr>
<tr>
<td>MajorityClassSize</td>
<td>1790.292</td>
<td>993.5</td>
</tr>
</tbody>
</table>

Combination of mean values for each cluster is presented in Table 3.2. The average data set in Cluster 1 has a feature vector consisting of 30 items, while the average feature vector for a data set in Cluster 2 has more than 1600 elements. Intuitively, operating in high-dimensional space is more difficult for a machine learning algorithm than in low-dimensional space. Consequently, it makes sense that these data sets were placed in different clusters.

The higher value for ClassEntropy in the second cluster aligns with the definition of this feature. Because entropy is the amount of information required to identify an instance, the higher number of features in Cluster 2 leads to higher entropy. The difference between the performance of the Decision Stump classifier measured in AUC for two clusters is less than 1 percent. It could be that the performance of the Decision Stump is mediocre because of the usage of only a single rule for a classification, even though tasks can be rather complicated or have more than two labels in the target feature. This single rule uses only the most important properties of instances in a data set and does not depend on the variety or the total number of features in the data.

The mean value of the number of values for the most diverse categorical feature is 36.88
for Cluster 1 and 460.14 for Cluster 2. The difference is fairly large, it can be caused by outliers. Analysis of Table 3.3 provides a clearer illustration of this feature. In this table, two clusters have a value for this feature equal to -1. This result means that at least half of data sets have no categorical values. Therefore, our suggestion about outliers that influence the mean value for this parameter is correct.

Table 3.3: Median values of features

<table>
<thead>
<tr>
<th>Name of parameter</th>
<th>Cluster 1</th>
<th>Cluster 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumAttributes</td>
<td>11</td>
<td>63.5</td>
</tr>
<tr>
<td>ClassEntropy</td>
<td>0.983</td>
<td>2.779</td>
</tr>
<tr>
<td>DecisionStumpAUC</td>
<td>0.719</td>
<td>0.721</td>
</tr>
<tr>
<td>MaxNominalAttDistinctValues</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>MajorityClassSize</td>
<td>203</td>
<td>200</td>
</tr>
</tbody>
</table>

The final attribute is the majority class size. Cluster 1 has almost two times larger size of the majority class than Cluster 2. However, Table 3.3 shows that the corresponding values are almost the same. As a result, it can be concluded that the differences in Table 3.2 are also influenced by outliers. This conclusion shows that our decision to study attributes of clusters from different perspectives is correct.

### 3.3.5 Selecting data sets for studying a frugality of algorithms

Computing a Frugality score in a brute force manner for every data set in the study can be time consuming and a more suitable approach should be found. Our suggestion is to select 10 representatives from all data sets that can be useful for studying a Frugality score. A method that can help to identify these data sets is the partitioning around medoids (PAM) [40]. This is an unsupervised method for clustering data. The main difference of this algorithm with K-means is that PAM selects a real point as the center of a cluster for every iteration while K-means chooses an abstract place in a feature space. Therefore, PAM allows to find the data set that is represented by the point in the center of a cluster. Each of these central data sets can be considered as the most typical example of a corresponding cluster. Choosing only these points rather than all data in the study can help to increase a calculation speed while keeping a various set of tasks with different properties. Since the proportion between size of the cluster one and the cluster two is nine to one, the same balance should be kept while selecting data sets for the analysis in Section 3.5 where heat maps and Frugality curves are created. The data sets selected from the first cluster are “980 optdigits”, “844 breastTumor”, “751 fri e4 1000 10”, “831 autoMpg”, “1038 gina agnostic”, “457 prnn cshushings”, “1119 adult-census”, “9 autos”, and “454 analcatdata halloffame”. The data set chosen from the second cluster is “20 mfeat-pixel”. These data sets are used for studying the properties of algorithms with Frugality curve plots presented in Section 3.5.3 of this chapter.

### 3.4 Pareto analysis

A comparison between algorithms in classification tasks is often done in terms of accuracy or AUC. This means that only performance is used for ranking algorithms. Even though performance is used in the Frugality score, this is a metric with two measures. In our study, the second measure, requirement for resources, is shown in terms of the time necessary to build and test a model.
Therefore, the connection between these two measures should be studied to establish a better understanding of the relationship between them.

### 3.4.1 Explanation of the Pareto Front method

A visual image is one the best ways to study a complex concept. A Pareto Front can be considered as a plot where each axis represents a different measure. Every data point should have values of these two measures. In that case it can be mapped to x and y axes and placed on a plot. The next step is drawing a frontier that consists of objects that have such a property that one measure cannot be improved while keeping the same or increasing a value of the second objective. When all points that satisfy this criterion are found, a line connecting all of them should be drawn. This line is the Pareto Front for a set of measurements.

![Pareto Front](image.png)

Figure 3.6: Pareto Front with the names of algorithms for the data set “722_pol”. The x axis contains results for AUC, and y axis contains log of build and test time for a classifier.

The Pareto Front for the results of classifiers collected for the data set “722_pol” is shown in Fig. 3.6. AUC has a range from zero to one, and time is shown on a logarithmic scale. The value $T(i)$ for y axis is computed with the Formula 3.5. On the one hand, algorithms that employ ensemble models, e.g. Rotation Forest [62] and Random Forest [50], show the best performance. Obviously, they work slower than simple but fast algorithms such as REPTree [80] or HyperPipes [77]. The value of AUC measure for a group of top performers on the left part of the graph is close to one. It seems that the dots representing these algorithms have almost the same AUC, but performance of each algorithm differs from others. The main idea of the Pareto Front is that if a machine learning method is presented on it, this classifier has a higher AUC that any other algorithm within the same period of time defined by the y axis.

$$T(i) = \log(t_{\text{train}} + t_{\text{test}})$$  \hspace{1cm} (3.5)

[^3]: [http://www.openml.org/d/722](http://www.openml.org/d/722)
Figure 3.7: Pareto Front for the first cluster.

The analysis of the meta-features in Section 3.3.2 suggested that data sets can be grouped in two clusters. Hence, two Pareto Fronts should be created. Each of them allows us to identify a collection of algorithms where every item has either high AUC or requires less computational time for a classifier. These collections can partially overlap with each other, because some algorithms show similar performance or frugality for both clusters. Obtained collections should be merged in a single group where each algorithm is presented only once. A reason for that is that analysis becomes more straightforward and interpretable when having only one set of notable algorithms.

### 3.4.2 Pareto Fronts for clusters of data sets

The best performing algorithm in Fig. 3.7 is Rotation Forest with 160 iterations. It has an AUC that is equal approximately to 0.87 and a computing time around 3.8. The second best algorithm, Random Forest, has a slightly smaller AUC value than Rotation Forest but works almost two times faster on average. A group of Bayesian methods, such Averaged 1-Dependence Estimator (A1DE) [74] and BayesNet shows AUC performance at around 0.83 and \( T \) value around 1. The faster algorithms with sluggish performance are Decision Stump and HyperPipes.

The presence of RealAdaBoost algorithm on Pareto Front deserves rigorous analysis. A property of this algorithm is that it can handle only two-class classification tasks. A label in the legend of a plot states that this classifier was able to perform classification only for 383 data sets out of 466 in the first cluster. This number of completed tasks is the lowest among algorithms shown in Fig. 3.7. In addition, the activity recognition task chosen for implementation in the application assumes that six types of activities, shown in Section 4.4.1, are performed in the experiment. This contradiction can be solved by including another version of the AdaBoost algorithm that can handle multi-class tasks, namely, AdaBoost.M1, to the list of those selected for detailed analysis algorithms. Results for these two classifiers are shown in Fig. 3.8. Based on the information from
this figure, we suggest that performance and time for AdaBoost.M1 with 10 iterations is close to the values shown by RealAdaBoost, and both of them can be evaluated further with heat maps and Frugality curves analysis.

As can be seen from Fig. 3.9, ensembles have the top performance for data sets in Cluster 2. The best algorithm is Rotation Forest with 40, 20, and 10 iterations. Naive Bayes has AUC and \( \mathcal{T} \) results that are close to those shown by Bayes Net [17]. The algorithms that work in the most constrained environments are RandomTree [80] and HyperPipes. Primarily, the distribution of algorithms for both clusters are similar: ensembles show the best performance but require a lot of time. A method based on Bayesian methods is a compromise between performance and time, and algorithms such as trees and HyperPipes show the best speed.

### 3.4.3 Results of analysis with Pareto Front method

The selection process shown in this section allows for identification the set of 17 noteworthy algorithms. Every classifier presented on Pareto Front should be carefully studied in terms of Frugality score for better understanding of suitability for working in constrained environments. Properties such as high performance or fast classification time can diminish or emerge with changes in the amount of available resources for a task.
3.5 Heat maps and frugality score lines

This section starts with the introduction of the singular value decomposition (SVD) method employed for a transition from the original to a latent feature space. The ensuing part studies three heat maps with Frugality score values constructed for results obtained with parameter $w = 0.1, 0.5, \text{ and } 1$. These settings imitate situations when a demand for resources is low, moderate or high. The Section 3.5.3 explains Frugality curves and their applications for assessing how a Frugality score changes for values of $w$ in a range between 0 and 1 for the studied set of algorithms.

3.5.1 SVD

The original data about AUC and combined time contain results for 103 (96 after removing classifiers that produced too many errors as explained in Section 2.4) algorithms and 516 data sets. Analysis conducted with clustering and Pareto Fronts allowed for reducing the number of potentially interesting algorithms for evaluation with Frugality scores down to 17. Therefore, this section attempts to find similarities in a matrix with 516x17 dimensionality. A value of the Frugality score obtained by an algorithm for a particular data set is considered to be a feature and put in a corresponding place. Exploring similarities and differences for such a big number of algorithms with a lengthy feature vector could be problematic even for an accurate and fast-clustering algorithm such as hierarchical clustering [38]. The reason is that finding common properties for algorithms based on 500 features is computationally expensive and does not always produce meaningful results. A solution for this issue could be in reducing the number of features. This can be done by moving in a low-dimensional space. Singular value decomposition [24] is a method that can help to perform this task.
SVD translates original data to a new latent space with a lower number of dimensions. The original space is described by a production of three matrices $UDV^t$. These elements are derived from the original matrix with Frugality scores obtained with a value for parameter $w$ equal to 0. Even though each of the three matrices has a different dimensionality compared to the original matrix, their production should produce, with some approximation, the source matrix.

The diagonal values of matrix $D$ can be used for evaluation of how many features should be used for constructing a new low-dimensional space. According to data, five latent features describe around 91 percent of a variance in the original data. Based on this information, a new low-dimensional space with five latent features can be created. The first matrix $U$ in this space has the size 516x5 and contains information about the data sets while matrix $V^t$ with dimensions 5x516 includes data about the algorithms. The element $D$ is a diagonal matrix with the size 5x5 that presents information about the importance of each latent feature for original data.

3.5.2 What information can we see on heat maps

Matrices obtained with SVD method at the previous step allow constructing heat maps in low-dimensional space and performing a hierarchical clustering for algorithms. Besides constructing a visual representation of a particular property of data, heat map allows us to create a dendrogram [20]. This data structure merges similar data sets in groups, combines this groups into larger groups, and this process continues until reaching the root. The dendrogram constructed for $w = 0$ is used for acquiring the order of data sets. Fig. 3.10 shows this dendrogram. As can be seen, some features such as four and five partially overlap with each other. On the other hand, features two, three, and four are rather distinct, which allows us to conclude that hierarchical clustering is meaningful for this combination of data sets and latent features.

The set of algorithms used for constructing heat maps originates from Section 3.4 where the best 17 classifiers were selected from two Pareto fronts. Algorithms with low performance measured in AUC can be found on the left on each heat map presented in Fig. 3.11, 3.12, and 3.13. The best performers in terms of AUC and with a high demand for such a resource as computational time are placed on the right side. This organization of algorithms and the order of data sets that is taken from a dendrogram, obtained in the beginning of this section, are used for generating heat maps. In particular, it is interesting to study heat maps created for the values of a parameter $w$ that represent low, medium, and extended constraints for a task.

Fig. 3.11, 3.12, and 3.13 presents heat maps constructed for $w$ equal to 0.1, 0.5, and 1, respectively. The order of rows is fixed according to the dendrogram presented in Fig. 3.10 and the order of columns is sorted with respect to performance of algorithms measured in AUC. The scale for Frugality score values ranges from 0 to 1. A low value for a Frugality score is denoted with light yellow colour while a high value is shown in red colour. The colour schema has been constructed based on YlOrRd palette from ColorBrewer [30] instrument that helps to select a colour encodings for maps.

The advantage of having fixed order for cells is an opportunity to visually compare different heat maps and identify differences between them. Accordingly, the first trend is that ensembles located on the right suffer from increasing constraints more than algorithms located in the middle or on the left. When the value of $w$ is equal to 0, these classifiers have a lot of dark red cells representing a high Frugality score for a particular task. However, with increase of $w$ up to 1, columns with scores for this methods become primary light yellow, meaning that a Frugality score is low and close to 0. Clearly, relatively high values of Frugality scores demonstrated by LogitBoost algorithm [12] with 40 iterations shown in Fig. 3.12 and 3.13 can be considered as an exception.

\[^4\text{http://colorbrewer2.org/}\]
CHAPTER 3. ANALYSIS OF ALGORITHMS WITH FRUGALITY SCORE

The second observation is that RealAdaBoost algorithm [22] emerges from the crowd when value of \( w \) increases. Algorithms that were close on Pareto Front to it such as RandomSubSpace [33] or LogitBoost with 10 iterations have diminishing Frugality scores for \( w \) equal to 0.5 or 1. Therefore, RealAdaBoost might be considered as a good candidate for tasks where performance and time are equally important. This statement should be proved with an extended analysis using Frugality curves in the next section.

Another fact that can be derived from heat maps is that frugality values of relatively simple algorithms, such as HyperPipes, Decision Stump, Random Tree, the dagging [69], and A1DE decrease slower than results of algorithms in the middle or on the right side of a plot. Especially, Fig. 3.12 presents substantial difference in colours where the left part is primarily red or orange and the right part is mostly yellow. These algorithms also should be studied with Frugality curves for the better understanding of their properties and relevant for them constraints.

3.5.3 Introducing Frugality curves and choosing algorithms for evaluation

This part shows how to assess frugality for algorithms. Frugality scores for each algorithm are computed in a range \([0; 1]\) for the \( w \) parameter and connected with a line in Fig. 3.14. This line is a Frugality curve. It presents how an algorithm fits for tasks with different amounts of available resources. Zero for \( w \) means that the performance of the algorithm is the top priority and the amount of resources should not be taken into account. The value of \( w \) equal to 1 means that the environment is constrained and that algorithms should work certainly fast while still producing meaningful results. The value of each Frugality score has been averaged across 10 data sets chosen with PAM methods in Section 3.3.5.
CHAPTER 3. ANALYSIS OF ALGORITHMS WITH FRUGALITY SCORE

Figure 3.11: Frugality scores heat map constructed with $w$ equal to 0.1. This value represents weak resource constraints for a task. Algorithms are sorted based on AUC performance showed in the previous section. The lowest performing algorithms can be found on the left, and the top performers are shown on the right. Data sets follow the order obtained with a dendrogram introduced in the beginning of this section.

Figure 3.12: Frugality scores heat map constructed with $w$ equal to 0.5. This value represents moderate resource constraints for a task. Algorithms are sorted based on AUC performance showed in the previous section. The lowest performing algorithms can be found on the left, and the top performers are shown on the right. Data sets follow the order obtained with a dendrogram introduced in the beginning of this section.
Figure 3.13: Frugality scores heat map constructed with $w$ equal to 1.0. This value represents extended resource constraints for a task. Algorithms are sorted based on AUC performance showed in the previous section. The lowest performing algorithms can be found on the left, and the top performers are shown on the right. Data sets follow the order obtained with a dendrogram introduced in the beginning of this section.

As shown in Fig. 3.14, a group of algorithms has similar Frugality curves in the whole range of values. Therefore, performance and $T$, defined in Formula 3.5, for these algorithms are almost the same. Even though many algorithms follow a different scenario, Frugality curves of these algorithms have a number of crossings. Rotation Forest with 160 iterations performs well in tasks in which resources are not taken into account. When the value of $w$ increases, this classifier starts to lose positions and at the point where $w$ is equal to 0.5, HyperPipes becomes a better choice for performing a classification task.

The best result for an environment where resources are really scarce is shown by the dagging algorithm. The given algorithm separates the input data in non-overlapping regions and builds models on them separately. This means that each model is trained on fewer data points, and the overall training requires less time than learning one model on the complete data set. It is important to notice that the careful attitude to used resources shown by this algorithm differs from other ensemble models such as Rotation Forest, Random Forest, or RandomSubSpace that show top results when $w$ is around 0 but then decrease faster than other algorithms that do not use basic learners.

A different behaviour is shown by A1DE algorithm. This algorithm belongs to the family of Naive Bayes methods and has a serious advantage compared to the original Naive Bayes classifier. It averages results over a set of small models, that allows decreasing independence assumptions between features in data. AUC results for this classifier are close to 0.8 and similar to those shown by a high performing algorithm like Rotation Forest. However, when the value of $w$ increases, it keeps its position in the middle between changing leaders and underperforming algorithms. This property of keeping a position with different constraints makes A1DE a good candidate for a wide range of tasks.
Based on the analysis made in this chapter, the dagging, A1DE, and ensemble AdaBoost.M1 are good candidates for the evaluation part of the project.
Chapter 4

Activity recognition on a wearable device

The main contribution of this chapter is a detailed explanation about design principles and the development process of an application that collects data and performs a machine learning task, activity recognition on a smartwatch. This application should help to evaluate our findings about the frugality properties of algorithms selected in Chapter 3. In the beginning, a general overview about a smartwatch is given. The next important topic for discussion is why this device is a good environment for testing various machine learning algorithms. Details of implementation such as platform, used libraries, and the architecture of the data processing procedure explain the process of creating the application.

When information about the application is presented, the next step is a description of data collection and processing. The group of activities should be predefined before the beginning of every HAR experiment. However, no rule or agreement exists about the standard types of activities that should be used in this task. Hence, a list of actions that a person should perform is subject to change in different projects. The activities used to train our classifiers are shown in Section 4.4.1.

The final section presents experimental results about performance of algorithms and battery consumption.

4.1 Smartwatch as an example of a wearable device

A number of researchers have performed human activity recognition on portable devices [67]. However, they mainly used a smartphone as a target device that collects data related to physical activities and performs recognition of these activities based on trained models. This project attempts to use a smartwatch as a constrained device able to perform machine learning tasks.

The target platform for the application is Android. The choice was made based on data showing that more than 1.4 billion devices use Android as an operating system1. Other important facts that influenced the decision to choose Android as a target system are that it is free, supported by different manufacturers, and used in several categories of consumer devices such as smartphones, wearable electronics, TV, laptops, and cars. With such a wide spectrum of devices that work with

---

1http://www.androidcentral.com/google-says-there-are-now-14-billion-active-android-devices-worldwide
this operating system, it should be possible to develop an application for one device and use it with some adjustments on different equipment.

The choice of a smartwatch as a device that is suitable for HAR and for testing frugality properties of algorithms was made based on next assumptions. The CPU of a mobile equipment with ARM architecture [66] has less computational power that modern x86 type CPU [54] installed on desktops, laptops, or servers [59]. Amount of RAM on smartwatch is quite restrictive. Namely, a fraction of 512 MB RAM is available on the model used in the project. This memory is not occupied by operating system and can be used by applications. Such limited amount of RAM is not typical for machine learning tasks. However, the storage capacity with size around 4 GB is enough to save information and trained models related to our machine learning task. Moreover, a smartwatch has a number of sensors that can produce data that can be utilized by a classifier in HAR. Classification based on this data can be accomplished directly on a device with no need to send data by network or use computational power of paired devices.

The model of smartwatch that is used in this work is LG Watch Urbane. Detailed specifications of the device are shown in Appendix A.2. This device has a number of features including a heart rate monitor, a sufficient amount of storage space to save information from sensors, and uses Android as an operating system. It also has a 1.3” screen and a modern outfit that simplified an interaction with the device for participants who were subjects of HAR experiment.

4.2 Details of implementation

The development process for wearable devices requires a special type of Android called Android Wear. The main difference with the mobile OS is that Wear is developed for working with a small screen, cannot use the Internet directly, and has a certain way of interacting with users. Mobile devices have big screens that enable usage of virtual keyboard input while wearable devices try to receive information from a user with gestures or voice commands.

The device includes a number of sensors. These can be divided in two categories: hardware-based and software-based sensors [25]. The former is a physical component that captures real world data. The latter is a software emulation of a sensor. A typical software-based sensor reads data from physical sensors, makes calculations based on this data, and shares a result of these calculations with the operating system. A linear acceleration sensor may be considered an example of a software-based sensor (a number of products have it as a hardware-based sensor). It computes linear acceleration based on the information about gravity collected from an accelerometer sensor.
The list of sensors used for collecting data for HAR is shown in Table 4.1. The descriptions for the majority of sensors are taken from this list \[25\].

Table 4.1: Sensors that are used for data collection about human activities

<table>
<thead>
<tr>
<th>Name of sensor</th>
<th>Dimensions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>3</td>
<td>Measures the acceleration force in ( \text{m/s}^2 ) that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.</td>
</tr>
<tr>
<td>Gyroscope</td>
<td>3</td>
<td>Measures a device’s rate of rotation in ( \text{rad/s} ) around each of the three physical axes (x, y, and z).</td>
</tr>
<tr>
<td>Gravity</td>
<td>3</td>
<td>Measures the force of gravity in ( \text{m/s}^2 ) that is applied to a device on all three physical axes (x, y, z).</td>
</tr>
<tr>
<td>Linear Acceleration</td>
<td>3</td>
<td>Measures the acceleration force in ( \text{m/s}^2 ) that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.</td>
</tr>
<tr>
<td>Rotation vector</td>
<td>4</td>
<td>Measures the orientation of a device by providing the three elements of the device’s rotation vector.</td>
</tr>
<tr>
<td>Heart rate</td>
<td>1</td>
<td>Measures a heart rate of a user in beats per minute [27].</td>
</tr>
<tr>
<td>Pressure</td>
<td>1</td>
<td>Measures the ambient air pressure in hPa or mbar.</td>
</tr>
<tr>
<td>Magnetic field</td>
<td>3</td>
<td>Measures the ambient geomagnetic field for all three physical axes (x, y, z) in T.</td>
</tr>
</tbody>
</table>

Information collected from sensors is stored in a file with 16 Hz frequency. The rate of 16 samples per second was chosen based on the assumption that more intensive reading from sensors and storing data locally could lead to considerable power consumption. In addition, not all sensors provide information faster than 16 times per second, and collecting data more often would not bring any additional value for HAR.

```
adb pull full_path_to_file_or_directory
```

Listing 4.1: Command to pull data from Android device

When data for all participants was recorded, it was uploaded to a computer with Android Debug Bridge (ADB) tool\(^2\). Command `adb pull`, entered in this tool, allows getting data from a device through command-line interface (CLI). Important to notice, that data should be stored to virtual external storage of smartwatch. The path to this storage can be obtained with a function shown in Listing 4.2. Otherwise, by default, data is stored in a protected folder that can be accessed only by the application that generated data, and retrieving data from this directory to an external device with ADB or another tool requires root rights for a user. Obtaining root rights means replacing particular protective mechanisms of Android and can lean to unexpected consequences. As a result, usage of virtual external storage was chosen as the main way of retrieving data from the device.

```
public static String getSensorStorageDir(String folderName) {
    File file = new File(Environment.getExternalStoragePublicDirectory(folderName),
                          toString());
    if (!file.mkdirs()) {
        Log.i(TAG, "Directory not created");
    }
    return file.toString();
}
```

Listing 4.2: Receiving a path to a folder with sensor data on external storage.

\(^2\)https://developer.android.com/studio/command-line/adb.html
Several processing steps were made for cleaning, merging, and creating new features in data. This steps are important since making classification of activities based on raw data can lead to unsatisfactory results. More information about this process is given in Section 4.5.

Three models of classifiers were trained in Windows’ version of Weka: dagging, A1DE, and AdaBoost.M1. These models were serialized to files, stored in a res/raw directory of the application, and transferred to the smartwatch. Models created with Weka work only with a specific data format called Instance. A set of Instances can be obtained from the ARFF file or from individually composed in operating memory during runtime Instances. Consequently, data collected from sensors should be processed and converted to this data structure. When new data are presented as Instances within the application, a deserialized classifier performs activity recognition.

Listing 4.3: Setting memory limit for Gradle that allows building project with Weka library.

A machine learning task on the device is made with the help of Weka library. This application in the form of a weka.jar file must be placed into a libraries folder of the project and added to dependencies in a build.gradle file where all important properties required for building the project are present. It is necessary to remove or comment lines in Weka that use GUI components and compile it from sources. Otherwise, using some tools such as filters can lead to an inability to run an application on an Android device. Versions of Weka compiled by developers specially for Android can be found on the Internet. However, they usually use an older version or have limited functionality since every developer has chosen what components of Weka he or she needed for a project. Therefore, it is meaningful to build one’s own version of the library with necessary functionality and based on the most recently updated code.

Three models with classifiers were created with Weka on a desktop machine. This software supports AdaBoost.M1 from the box, and A1DE and the dagging objects were created using AnDE.jar and dagging.jar libraries, respectively. Weka.jar, AnDE.jar, and dagging.jar should be placed in a libraries folder and included as dependencies for building a project. Note that the level of available memory for the build automation system Gradle that is used in Android Studio should be increased up to 3GB or more if an Android project uses Weka. This can be done by configuring “gradle.properties” file with options shown in Listing 4.3. The given amount of memory was sufficient to make a build while a default amount of memory was not enough for assembling the application.

4.3 User interface

The developed application has two modes. The first one is designed for data collection, and the second one is used for activity recognition. When the application starts, a user can see an introduction screen shown in Fig. 4.2. Pressing a button leads to a transition to a corresponding mode. Switching to another mode can be done by restarting the application.
CHAPTER 4. ACTIVITY RECOGNITION ON A WEARABLE DEVICE

Figure 4.2: Introduction screen with two available options for a user to choose from.

The data collection mode, shown in Fig. 4.3, allows collecting data about activities to a file. A new file is created every time by starting the application in this mode. A user can press a red button with a hand sign and choose one out of six types of activities from a menu.

Figure 4.3: The main screen of data collection mode.

The view of a menu with icons\(^8\) that depict \textit{walking}, \textit{going upstairs}, \textit{going downstairs}, \textit{sitting}, \textit{standing}, and \textit{lying} is shown in Fig. 4.4. In general, design of the application was developed in accordance with Material Design principles \cite{55} proposed by Google. The main feature of this style is creating consistent interfaces with standard layouts across a wide range of devices. It has the goal of presenting information in a concise form and providing a unified set of control elements.

\footnote{Icons used in this project were downloaded from \textit{Material Icons} page of Google design website, available by address \url{https://design.google.com/icons/} and accessed on the 16th of May, 2016.}

Frugal Learning: Applying Machine Learning with Minimal Resources
that can be easily recognized and used by a majority of users.

Figure 4.4: Selection of activity for data collection with icons representing types of activities.

Information shown on the screen while performing data collection is basic. As can be seen from Fig. 4.5, the name of an activity is shown at the top and a pause button is at the bottom. The idea behind this is that a user should not be distracted by a timer with time shown since the start of an activity or a blinking icon related to performing the task. As a result, a user selects a desirable task by pressing a button in the main menu and performs the activity without looking at the screen. This helps us to avoid unnecessary noise related to moving a hand and looking at the information on the screen that users of the first prototype performed often.

Figure 4.5: Screen of the application during data collection about *standing* activity.

When the application is about to close, a file with data is closed for writing. Therefore, if the information about one person should be saved to one file, a dialog, shown in Fig. 4.7,
confirmation about closing the application can help preventing undesirable quitting.

Files with data about users are stored to a virtual external storage as explained in Section 4.2. Every file has a name with the date and time when data collection started. As can be expected, a file with data about activities is not created in recognition mode.

A classification of activities is done with one of selected algorithms. The chosen algorithm should be selected in the source code before the building of the application, and a file with the corresponding model will be automatically loaded by the operating system. The screen presents the name of activity that a classifier recognized and the pause button. When a user presses this button, the same dialog about closing the application, shown in Fig. 4.7, appears.
4.4 Collection of data in experiment

This section describes the data collection process with the developed application. Specifications of hardware, data format, forms signed by participants, and information about environmental setup are shown in Appendix A. Moreover, description about how data was collected and stored, and details about the environment in which the experiment took place are also shown there.

4.4.1 Performed activities

The number of people that participated in the experiment is 20. Every participant was asked to complete the set of six activities.

![Activity Images]

Figure 4.8: Types of activities performed by the participants of the data collection experiment.

The length of each performed activity is one minute after filtering first five seconds in the beginning and in the end of every action. Given terms allowed to collect 120 minutes \((1 \times 6 \times 20)\) of data that was used to train classifiers.

4.4.2 Data submission and storage once its collected

Individual file with data created for every participant. It remains on the device on a persistent storage while collecting data for other participants within the same day. When all participants completed set of tasks for a particular day, data is uploaded on a computer for cleaning, merging and used in training classifiers. The final model, used for activity recognition evaluations, that is constructed for every classifier using all data collected from all participants.
CHAPTER 4. ACTIVITY RECOGNITION ON A WEARABLE DEVICE

4.5 Data preparation

Data produced by a sensor can be 1-dimensional or multi-dimensional. It depends on whether a signal can be determined in one or more directions by built-in equipment in the smartwatch. A sensor such as an accelerometer produces data that is recorded for x, y, and z components. Fig. 4.1 shows information about the Android coordinate system. Some data is plain and has only one dimension. An example of such a sensor is a barometer. It measures air pressure and produces a single value as a result of a measurement. Detailed information about the number of dimensions available for every sensor is shown in Table 4.1.

The first step in data processing is cleaning. We have removed five seconds from the start and the end of every activity. This was necessary because, while starting and finishing an activity, participants pressed a touch screen button on the device. The combination of actions required to press a button on a small touch screen requires rotating a hand, moving it closer to a face, and using another hand for tapping an activity’s icon. This set of movements takes around two seconds, but for the elimination of noise not related to performing activities the interval filtered out was increased up to five seconds. Cleaned data for 20 subjects of the experiment were merged in a single file.

The next step in our data processing pipeline after filtering noise related to tapping a screen is an identification and removing of highly correlated features. Using these features can lead for decreasing performance of classifiers. Correlation matrix was constructed in R with “Caret” package9. This matrix allows identify features with high correlations. All combinations of features with correlation higher than threshold equal to 0.75, that means a high correlation between two values, are coloured in green and shown in Tables 4.2 and 4.3. Feature with the highest mean correlation with other features from such a pair is excluded from the study. While doing this, it can help improve model accuracy by removing noise and irrelevant information.

---

9https://cran.r-project.org/web/packages/caret
CHAPTER 4. ACTIVITY RECOGNITION ON A WEARABLE DEVICE

Figure 4.10: General importance of raw features for predicting a target variable calculated by Random Forest.

Figure 4.11: Importance of raw features for predicting each individual target variable computed by the learning vector quantization classifier. Codes of activities: X0 - Walking, X1 - Walking upstairs, X2 - Walking downstairs, X3 - Sitting, X4 - Standing, X5 - Lying.

classification accuracy should increase. Based on correlations, $RotVecZ$ (Rotation vector, z axis), $MagFielX$ and $MagFielZ$ (Magnetic field, x and z axes) should be removed from the list of features.

Section 4.2 stated that data were collected for many sensors and for all dimensions available for these sensors. However, whether all information is important for a proper activity classification
### Table 4.2: Correlations between sensors based on raw data, part one.

Green cell connotes that two variables have an absolute value of pair-wise correlation higher than 0.75. A light blue cell denotes a feature that should be excluded from the analysis.

<table>
<thead>
<tr>
<th></th>
<th>AccelX</th>
<th>AccelY</th>
<th>AccelZ</th>
<th>GyroX</th>
<th>GyroY</th>
<th>GyroZ</th>
<th>GravityX</th>
<th>GravityY</th>
<th>GravityZ</th>
<th>LinAccelX</th>
<th>LinAccelY</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccelX</td>
<td>1</td>
<td>0.17</td>
<td>0.02</td>
<td>0.02</td>
<td>0</td>
<td>0.03</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
<td>-0.01</td>
</tr>
<tr>
<td>AccelY</td>
<td>0.17</td>
<td>1</td>
<td>-0.02</td>
<td>-0.01</td>
<td>-0.02</td>
<td>-0.02</td>
<td>0</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>-0.03</td>
</tr>
<tr>
<td>AccelZ</td>
<td>0.02</td>
<td>-0.02</td>
<td>1</td>
<td>-0.02</td>
<td>-0.02</td>
<td>0.01</td>
<td>0</td>
<td>-0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.04</td>
</tr>
<tr>
<td>GyroX</td>
<td>0.02</td>
<td>-0.01</td>
<td>-0.02</td>
<td>1</td>
<td>0.07</td>
<td>-0.35</td>
<td>-0.01</td>
<td>0.01</td>
<td>-0.01</td>
<td>-0.06</td>
<td>-0.13</td>
</tr>
<tr>
<td>GyroY</td>
<td>0</td>
<td>-0.02</td>
<td>-0.02</td>
<td>0.07</td>
<td>1</td>
<td>0.48</td>
<td>0</td>
<td>0.01</td>
<td>0</td>
<td>0.09</td>
<td>-0.01</td>
</tr>
<tr>
<td>GyroZ</td>
<td>0.03</td>
<td>-0.02</td>
<td>0.01</td>
<td>-0.35</td>
<td>0.48</td>
<td>1</td>
<td>0</td>
<td>-0.01</td>
<td>0</td>
<td>0.15</td>
<td>0.08</td>
</tr>
<tr>
<td>GravityX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-0.01</td>
<td>0</td>
<td>0.01</td>
<td>1</td>
<td>-0.1</td>
<td>0.56</td>
<td>0.21</td>
<td>0.1</td>
</tr>
<tr>
<td>GravityY</td>
<td>0</td>
<td>0.01</td>
<td>-0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>-0.01</td>
<td>-0.1</td>
<td>1</td>
<td>0.07</td>
<td>0.02</td>
<td>-0.32</td>
</tr>
<tr>
<td>GravityZ</td>
<td>0</td>
<td>0.01</td>
<td>-0.01</td>
<td>0.01</td>
<td>0</td>
<td>0.01</td>
<td>0.07</td>
<td>1</td>
<td>0.05</td>
<td>0.05</td>
<td>0.1</td>
</tr>
<tr>
<td>LinAccelX</td>
<td>0.01</td>
<td>0</td>
<td>0.01</td>
<td>-0.06</td>
<td>0.09</td>
<td>0.15</td>
<td>0.21</td>
<td>0.02</td>
<td>0.05</td>
<td>1</td>
<td>-0.07</td>
</tr>
<tr>
<td>LinAccelY</td>
<td>-0.01</td>
<td>-0.03</td>
<td>0.04</td>
<td>-0.13</td>
<td>-0.01</td>
<td>0.08</td>
<td>0.1</td>
<td>-0.32</td>
<td>0.1</td>
<td>-0.07</td>
<td>1</td>
</tr>
<tr>
<td>LinAccelZ</td>
<td>0</td>
<td>0.01</td>
<td>-0.03</td>
<td>-0.08</td>
<td>0.05</td>
<td>0.04</td>
<td>0.17</td>
<td>0.12</td>
<td>0.11</td>
<td>0.32</td>
<td>-0.3</td>
</tr>
<tr>
<td>RotVecX</td>
<td>0</td>
<td>0</td>
<td>-0.01</td>
<td>0.02</td>
<td>0.01</td>
<td>0.04</td>
<td>0.29</td>
<td>0.13</td>
<td>-0.05</td>
<td>-0.09</td>
<td>-0.09</td>
</tr>
<tr>
<td>RotVecY</td>
<td>0</td>
<td>-0.01</td>
<td>-0.01</td>
<td>-0.02</td>
<td>0.02</td>
<td>0.03</td>
<td>-0.54</td>
<td>-0.1</td>
<td>-0.45</td>
<td>-0.12</td>
<td>-0.03</td>
</tr>
<tr>
<td>RotVecZ</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
<td>0.01</td>
<td>-0.01</td>
<td>-0.02</td>
<td>0.06</td>
<td>0.13</td>
<td>-0.05</td>
<td>-0.07</td>
<td>-0.03</td>
</tr>
<tr>
<td>RotVecS</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
<td>0.01</td>
<td>-0.01</td>
<td>-0.02</td>
<td>0.06</td>
<td>0.13</td>
<td>-0.05</td>
<td>-0.07</td>
<td>-0.03</td>
</tr>
<tr>
<td>AiPreVal</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.06</td>
<td>0.04</td>
<td>-0.04</td>
<td>-0.02</td>
<td>-0.06</td>
<td>0.06</td>
</tr>
<tr>
<td>MagFieldX</td>
<td>-0.01</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
<td>-0.01</td>
<td>-0.01</td>
<td>-0.79</td>
<td>0.04</td>
<td>-0.53</td>
<td>-0.21</td>
<td>-0.08</td>
</tr>
<tr>
<td>MagFieldY</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
<td>-0.02</td>
<td>0.01</td>
<td>0.04</td>
<td>0.01</td>
<td>0.15</td>
<td>-0.01</td>
<td>-0.16</td>
<td>0.16</td>
</tr>
<tr>
<td>MagFieldZ</td>
<td>0</td>
<td>-0.01</td>
<td>-0.01</td>
<td>0</td>
<td>-0.02</td>
<td>-0.01</td>
<td>-0.58</td>
<td>0.04</td>
<td>-0.01</td>
<td>-0.14</td>
<td>-0.07</td>
</tr>
<tr>
<td>HeartRateVal</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
<td>0.01</td>
<td>-0.33</td>
<td>0.11</td>
<td>-0.17</td>
<td>-0.18</td>
<td>-0.03</td>
<td>-0.03</td>
</tr>
<tr>
<td></td>
<td>LinAccelZ</td>
<td>RotVecX</td>
<td>RotVecY</td>
<td>RotVecZ</td>
<td>RotVecS</td>
<td>AiPreVal</td>
<td>MagFielX</td>
<td>MagFielY</td>
<td>MagFielZ</td>
<td>HeartRateVal</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td>AccelX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AccelY</td>
<td>0.01</td>
<td>0</td>
<td>-0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-0.01</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AccelZ</td>
<td>-0.03</td>
<td>-0.01</td>
<td>-0.01</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
<td>-0.01</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>GyroX</td>
<td>-0.08</td>
<td>0</td>
<td>-0.02</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>-0.02</td>
<td>0</td>
<td>0.01</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>GyroY</td>
<td>0.03</td>
<td>0.02</td>
<td>0.02</td>
<td>-0.01</td>
<td>-0.01</td>
<td>0</td>
<td>-0.01</td>
<td>0.01</td>
<td>-0.02</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>GyroZ</td>
<td>0.04</td>
<td>0.01</td>
<td>0.03</td>
<td>-0.02</td>
<td>-0.02</td>
<td>0</td>
<td>-0.01</td>
<td>0.04</td>
<td>-0.01</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>GravityX</td>
<td>0.17</td>
<td>0.04</td>
<td>-0.54</td>
<td>0.06</td>
<td>0.06</td>
<td>-0.79</td>
<td>0.01</td>
<td>-0.58</td>
<td>-0.33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GravityY</td>
<td>0.12</td>
<td>0.29</td>
<td>-0.1</td>
<td>0.13</td>
<td>0.13</td>
<td>0.04</td>
<td>0.04</td>
<td>-0.7</td>
<td>0.04</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>GravityZ</td>
<td>-0.11</td>
<td>0.13</td>
<td>-0.45</td>
<td>-0.05</td>
<td>-0.05</td>
<td>-0.04</td>
<td>-0.53</td>
<td>-0.15</td>
<td>-0.81</td>
<td>-0.17</td>
<td></td>
</tr>
<tr>
<td>LinAccelX</td>
<td>0.32</td>
<td>-0.05</td>
<td>-0.12</td>
<td>0.07</td>
<td>0.07</td>
<td>-0.02</td>
<td>-0.21</td>
<td>-0.01</td>
<td>-0.14</td>
<td>-0.18</td>
<td></td>
</tr>
<tr>
<td>LinAccelY</td>
<td>-0.3</td>
<td>-0.09</td>
<td>-0.03</td>
<td>-0.03</td>
<td>-0.03</td>
<td>0.06</td>
<td>-0.08</td>
<td>0.16</td>
<td>-0.07</td>
<td>-0.03</td>
<td></td>
</tr>
<tr>
<td>LinAccelZ</td>
<td>1</td>
<td>0.01</td>
<td>-0.04</td>
<td>0.05</td>
<td>0.05</td>
<td>-0.06</td>
<td>-0.17</td>
<td>-0.09</td>
<td>-0.1</td>
<td>-0.08</td>
<td></td>
</tr>
<tr>
<td>RotVecX</td>
<td>0.01</td>
<td>1</td>
<td>0.1</td>
<td>-0.49</td>
<td>-0.49</td>
<td>0.07</td>
<td>-0.03</td>
<td>-0.29</td>
<td>-0.36</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>RotVecY</td>
<td>-0.04</td>
<td>0.1</td>
<td>1</td>
<td>-0.42</td>
<td>-0.42</td>
<td>-0.02</td>
<td>0.34</td>
<td>0.22</td>
<td>0.36</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td>RotVecZ</td>
<td>0.05</td>
<td>-0.49</td>
<td>-0.42</td>
<td>1</td>
<td>1</td>
<td>-0.03</td>
<td>0.25</td>
<td>-0.04</td>
<td>0.15</td>
<td>-0.06</td>
<td></td>
</tr>
<tr>
<td>RotVecS</td>
<td>0.05</td>
<td>-0.49</td>
<td>-0.42</td>
<td>1</td>
<td>1</td>
<td>-0.03</td>
<td>0.25</td>
<td>-0.04</td>
<td>0.15</td>
<td>-0.06</td>
<td></td>
</tr>
<tr>
<td>AiPreVal</td>
<td>-0.06</td>
<td>0.07</td>
<td>-0.02</td>
<td>-0.03</td>
<td>-0.03</td>
<td>1</td>
<td>0.04</td>
<td>0.05</td>
<td>0.06</td>
<td>0.21</td>
<td></td>
</tr>
<tr>
<td>MagFielX</td>
<td>-0.17</td>
<td>-0.05</td>
<td>0.34</td>
<td>0.25</td>
<td>0.25</td>
<td>0.04</td>
<td>1</td>
<td>0.09</td>
<td>0.5</td>
<td>0.28</td>
<td></td>
</tr>
<tr>
<td>MagFielY</td>
<td>-0.09</td>
<td>-0.29</td>
<td>0.22</td>
<td>-0.04</td>
<td>-0.04</td>
<td>0.05</td>
<td>0.09</td>
<td>1</td>
<td>0.08</td>
<td>-0.04</td>
<td></td>
</tr>
<tr>
<td>MagFielZ</td>
<td>-0.1</td>
<td>-0.36</td>
<td>0.36</td>
<td>0.15</td>
<td>0.15</td>
<td>0.06</td>
<td>0.5</td>
<td>0.08</td>
<td>1</td>
<td>0.21</td>
<td></td>
</tr>
<tr>
<td>HeartRateVal</td>
<td>-0.08</td>
<td>0.02</td>
<td>0.17</td>
<td>-0.06</td>
<td>-0.06</td>
<td>0.21</td>
<td>0.28</td>
<td>-0.04</td>
<td>0.21</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.3: Correlations between sensors based on raw data, part two. Green cell connotes that two variables have an absolute value of pair-wise correlation higher than 0.75. A light blue cell denotes a feature that should be excluded from the analysis.
or some part of it can be removed is a question that should be considered. A number of algorithms that can calculate the significance of every feature exists [13].

Using Random Forest to evaluate the importance of each feature for a proper classification, a plot that shows the feature's significance can be constructed. However, Random Forest identifies only the general importance of each feature. Conversely, the algorithm for learning vector quantization [44] allows us to create a plot in which each activity has its own order for the importance of features. Both plots are presented in Fig. 4.10 and 4.11. As can be seen, they have a number of similarities.

Surprisingly, an accelerometer and gyroscope sensors are amongst the least significant for a proper classification. These results were obtained with 10-fold cross-validation that reduced the chance that given observations are influenced by outliers. The explanation could be that raw values from these sensors do not have a significant impact on a classification but new features engineered based on them can be valuable for HAR. Moreover, a number of researchers [21, 49, 60] evaluated data from these sensors and found them useful for HAR. As a result, information from these sensors is left for further analysis.

When the set of features that remain in the data set was identified, a data preparation process could be advanced to the next stage in which new features based on available information could be created. Based on the work done in the HAR area by other researchers [49, 21], min, max, mean values, range (i.e. the difference between max and min), and standard deviation were selected as features helpful for increasing the value of correct evaluations by classifiers. Consequently, these values were calculated for every feature. The window used for constructing new features was 2 seconds with 50 percent overlap. The given value for overlap was used by a number of scientists and allowed them to achieve a reasonable performance [6, 56, 78]. The number of attributes in a feature vector after this operation increased up to 91 \( ((21 - 3) \times 5 + 1) \) from original 22 \((21 + 1)\) in the original with raw data.

Once the data are cleaned and processed, the final step is to convert them to a format that Weka can read and use for building and testing classifiers. This format is Attribute-Relation File Format (ARFF), a standard format that used by this software. All features in our data that are collected from sensors or computed at a later stage should be treated as double values, and the target feature should be transformed into a categorical variable with six values. The number of categories in the target feature corresponds to the number of activities that participants performed during the experiment.

### 4.6 Analysis of performance and battery consumption

Three algorithms were selected based on their Frugality score for using on a wearable device. Namely, they are the dagging, A1DE, and AdaBoost.M1. The latter is an option of AdaBoost algorithm that can work with multi-class classification tasks. Since AdaBoost.M1 and the dagging use ensembles of weak learners for determining a class of the target feature, a base classifier has to be chosen for them. In this set of experiments, J48 was chosen as a weak learner. It is important to notice that A1DE works faster and requires a smaller file size for a serialized classifier model when processes categorical features instead of numerical. As a result, data were preprocessed for this classifier with Discretize filter\(^{10}\) in Weka. Other parameters were not changed and used with default settings.

A number of experiments were designed for the application. While performing an activity recognition task, information about battery level and classification performance was recorded. An interval for storing information about features was set to 2 seconds. The time window for

\(^{10}\)http://weka.sourceforge.net/doc.dev/weka/filters/unsupervised/attribute/Discretize.html
computing new features is also 2 seconds. A classifier evaluates features every second and shows the result on a screen. As was mentioned in Section 4.5, the overlap between windows is 50 percent. Data about battery level were collected at 30-seconds intervals. A relatively long period for a measurement of battery state was chosen since checking a battery level more often can have more impact on it than performing an activity recognition task.

Table 4.4: Performance of classifiers for HAR.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Accuracy</th>
<th>AUC</th>
<th>Build Time, sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>The dagging</td>
<td>87.45</td>
<td>0.982</td>
<td>1.17</td>
</tr>
<tr>
<td>A1DE</td>
<td>81.08</td>
<td>0.975</td>
<td>0.28</td>
</tr>
<tr>
<td>AdaBoost.M1</td>
<td>95.37</td>
<td>0.997</td>
<td>52.75</td>
</tr>
</tbody>
</table>

The performance of algorithms was evaluated on a desktop computer based on data collected and processed in the study. According to the results, AUC for algorithms varies from 0.975 for A1DE to 0.997 for AdaBoost.M1. Other information about Accuracy, AUC, and build time is shown in Table 4.4. As can be seen, A1DE has lower AUC and Accuracy than the dagging but can be created in 0.28 seconds, which is fast enough compared to 1.17 seconds for the dagging. AdaBoost.M1 has the highest Accuracy and AUC but was significantly slower than other algorithms. These values align with data obtained from Frugality score results. Confusion matrices for three algorithms are presented in Tables 4.5, 4.6, and 4.7. As a reminder, with taking into account results for AUC and combined time for building and testing a classifier, the dagging was considered as an algorithm that can perform well in constrained environments; A1DE is a universal choice for situations in which performance and time are equally important. Finally, AdaBoost.M1 is a meaningful choice for situations in which performance is a priority.

Table 4.5: Confusion matrix for the dagging with 10-fold cross-validation.

<table>
<thead>
<tr>
<th>Walking</th>
<th>Upstairs</th>
<th>Downstairs</th>
<th>Sitting</th>
<th>Standing</th>
<th>Lying</th>
<th>Classified as</th>
</tr>
</thead>
<tbody>
<tr>
<td>1057</td>
<td>153</td>
<td>119</td>
<td>3</td>
<td>15</td>
<td>3</td>
<td>Walking</td>
</tr>
<tr>
<td>147</td>
<td>1219</td>
<td>174</td>
<td>0</td>
<td>8</td>
<td>1</td>
<td>Upstairs</td>
</tr>
<tr>
<td>137</td>
<td>220</td>
<td>928</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>Downstairs</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>11</td>
<td>1531</td>
<td>19</td>
<td>17</td>
<td>Sitting</td>
</tr>
<tr>
<td>14</td>
<td>17</td>
<td>32</td>
<td>21</td>
<td>1392</td>
<td>2</td>
<td>Standing</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>12</td>
<td>14</td>
<td>0</td>
<td>1969</td>
<td>Lying</td>
</tr>
</tbody>
</table>

Table 4.6: Confusion matrix for A1DE with 10-fold cross-validation.

<table>
<thead>
<tr>
<th>Walking</th>
<th>Upstairs</th>
<th>Downstairs</th>
<th>Sitting</th>
<th>Standing</th>
<th>Lying</th>
<th>Classified as</th>
</tr>
</thead>
<tbody>
<tr>
<td>1064</td>
<td>180</td>
<td>37</td>
<td>2</td>
<td>53</td>
<td>14</td>
<td>Walking</td>
</tr>
<tr>
<td>243</td>
<td>1041</td>
<td>214</td>
<td>3</td>
<td>45</td>
<td>3</td>
<td>Upstairs</td>
</tr>
<tr>
<td>215</td>
<td>170</td>
<td>852</td>
<td>9</td>
<td>46</td>
<td>1</td>
<td>Downstairs</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1461</td>
<td>7</td>
<td>114</td>
<td>Sitting</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>3</td>
<td>187</td>
<td>1183</td>
<td>89</td>
<td>Standing</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>83</td>
<td>16</td>
<td>1905</td>
<td>Lying</td>
</tr>
</tbody>
</table>

In addition to performance evaluation, resource consumption is an important parameter in a Frugality score. We measured it in terms of battery life. Classifiers that deliver high performance should work a reasonable amount of time on a device with limited access to energy.

In the experiment about changes in battery level caused by three classifiers, each algorithm ran the same set of activities. Before starting a new evaluation, the device was fully charged. This action is important since the battery level decreased at different rates depending on the level

Frugal Learning: Applying Machine Learning with Minimal Resources  43
of charge left. Consequently, experiment conditions for all classifiers should be the same. This condition can help to achieve more reliable and fair results for every algorithm in the experiment.

Table 4.7: Confusion matrix for AdaBoost.M1 with 10-fold cross-validation.

<table>
<thead>
<tr>
<th></th>
<th>Walking</th>
<th>Upstairs</th>
<th>Downstairs</th>
<th>Sitting</th>
<th>Standing</th>
<th>Lying</th>
<th>Classified as</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walking</td>
<td>1244</td>
<td>60</td>
<td>38</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>Walking</td>
</tr>
<tr>
<td>57</td>
<td>1412</td>
<td>76</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td>Upstairs</td>
</tr>
<tr>
<td>52</td>
<td>91</td>
<td>1148</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Downstairs</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1573</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>Sitting</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>1450</td>
<td>0</td>
<td>0</td>
<td>Standing</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2002</td>
<td>Lying</td>
</tr>
</tbody>
</table>

The watch was placed on the wrist, and the application was run in active mode\(^{11}\) with a constantly working screen. This helped to run the application seamlessly and avoid the battery optimizers in Android that decrease CPU speed and put additional resource constraints on running tools. A special optimization for the Android platform that can increase running time for the application and allow the use of our tool as a service available for other applications is a part of a future work as described in section 5.2.

Figure 4.12: Battery consumption measured for the dagging, A1DE, and AdaBoost.M1 classifiers.

Experiments revealed that the power consumption certainly depends on a algorithm. As can be seen in Fig. 4.12, the AdaBoost.M1 has the lowest power consumption among studied algorithms except for the period of 2-16 minutes. The worst performer, A1DE, is plotted with a green line. This line tends to be lower or at the same level as other algorithms, meaning that this algorithm consumes more energy than other classifiers. The reason could be that a discretization filter used by this classifier adds the data preparation overhead leading to more active power

---

\(^{11}\)http://developer.android.com/training/wearables/apps/always-on.html
consumption. The behaviour demonstrated by the dagging is relatively stable. In most situations, the battery level of this algorithm is between A1DE and AdaBoost.M1 in the same period of time. To conclude, AdaBoost.M1 and the dagging show the power consumption similar to expected, and A1DE consumes the highest amount of energy. Small deviations in energy consumptions, especially in the first minutes, can be caused by other applications or an operating system that tries to optimize work for all programs and starts or stops additional services, which slightly influence battery level.

Battery consumption and performance results for studied classifiers mainly align with data collected from OpenML.org data sets. Consequently, the behaviour of algorithms evaluated with a Frugality score is correct, and real observations support inferences made in the algorithms selection process.
Chapter 5

Conclusions

This thesis introduces a new perspective on the analysis of machine learning algorithms in terms of their frugality. The Frugality score combines performance with the amount of resources required for a classifier to perform a task. The formula presented in Section 3.1 shows how this new metric can be used for evaluation of algorithms based on the mentioned objectives. Another important property of a Frugality score is that it can be computed with various combinations of measurements for performance evaluation and resource consumption.

5.1 Results and discussion

Application of a Frugality score to an algorithm selection process for tasks with different constraints is shown in Chapter 3. Using data from OpenML.org platform, and Weka as machine learning environment used for classification, a number of metrics such as AUC, building time, and training time were collected and used for computing this new measure for algorithms. A fraction of algorithms could not correctly perform a classification task due to inner limitations or another type of error. Information about these missing values was obtained for a paper [19] and is given in Appendix C.

Data sets were clustered into two groups. Use of clustering allowed us to avoid situations in which an algorithm shows frugal behaviour for a particular data set but is not frugal for a wide range of projects. One cluster contained data about tasks with a smaller number of attributes than those in another group. This difference was one of the most significant between clusters. More information about numerous properties of data sets in clusters is shown in Section 3.3.4. Two different approaches: PCA and t-SNE were used for the visualization of clusters. Fig. 3.3 and 3.4 created with these visualizations support the idea that clusters have reasonable differences and can be separated.

The selection of algorithms studied in the Frugality score analysis section was made using the Pareto Front method. The given approach allowed for creating plots that presented all algorithms with outstanding performance in classification task or that consumed a modest amount of resources. AUC and combined time consisting of train and test time were used as objectives for Pareto Front analysis. Overall, 16 algorithms were chosen for the next stage of the study.

Data sets that contained information about Frugality scores for 16 algorithms and 103 algorithms contained noise and were rather large for valid hierarchical clustering. Using the SVD approach for transferring this data set to a low-dimensional space with five latent features helped to...
perform hierarchical clustering and create a dendrogram that arranged data sets based on latent features. Heat maps were used as a tool to compare algorithms in environments with different constraints. Algorithms for these visualizations were sorted by the average performance that they showed on Pareto Front figures. Three heat maps created with frugality weight \( w \) equal to 0.1, 0.5, and 1, and these correspond to low, average, and high demand for resources. Based on this information, properties of algorithms for different constraints were studied. Finally, construction of Frugality curves allowed us to choose three algorithms that have compelling properties useful for running a machine learning task such as HAR on a wearable device. These algorithms the dagging, A1DE, and AdaBoost.M1.

Information about the application created for this project is shown in Chapter 4. Device properties and used sensors are discussed together with a description about what kind of information was collected from 20 participants in the experiment. The procedure of data processing is divided into cleaning, feature selection, and feature engineering. Details of battery consumption and evaluation of accuracy shown by classifiers allowed us to compare expectations obtained with Frugality curves and real data from the application.

## 5.2 Future work

This study introduced a Frugality score and showed how to compute it for a number of machine learning algorithms from the Weka library. The obtained results allowed us to find algorithms used in the application developed for HAR. The given application allowed us to mostly verify assumptions about performance and battery consumption made with a Frugality score. However, a number of improvements to this application should be made, so it can be used by a wide range of users for performing activity recognition on wearable devices.

### 5.2.1 Performance optimization

The battery of a wearable device has a limited capacity due to physical size constraints. The operating system used on such devices attempts to maintain a balance between usability and battery consumption. As a result, Android tries to switch off a screen after a short period of time or force applications to work in “doze” mode [26]. This mode is used for doing background tasks in short intervals and keeps a device idle for the majority of the time.

Figure 5.1: Doze provides a recurring maintenance window for apps to use the network and handle pending activities. Adapted from [26].

The application used in this project may be optimized by doing a little amount of work...
in this mode and still provide an accurate classification. Values between active periods can be 
interpolated based on the context and limited amount of information that is available for collecting 
in time periods between active segments. These improvements in power management require a 
detailed study of available for developers opportunities and numerous tests and can be incorporated 
into a future work.

Another topic that relevant for performance of machine learning algorithms on constrained 
devices is an influence of floating-point precision on classification speed and battery consumption. 
As was mentioned in Section 2.3, research conducted for a smartphone [2] revealed that using 
integer values for features instead of floating-point variables can increase battery life up to 100 
percent. To the best of our knowledge, an extensive survey of such a type that encompasses a 
wide range of devices and various classification algorithms does not exist in the literature. This 
survey might bring a new perspective on performance of machine learning algorithms not only for 
constrained devices but also for a wider range of equipment.

5.2.2 API for external applications

Accurate classification of activities can be useful in many domains. Therefore, it can be helpful 
for developers to use predictions from this app in their applications. Moreover, developers choose 
what is more important for them: higher performance or modest battery consumption. These 
choices are made by picking one of the available classifiers or downloading additional components 
from a repository. Such a functionality can be provided in a form of API with all necessary 
functions.

5.2.3 Personalization of standard machine learning models

Data that are used by a classifier can change over time. Those uploaded to device models used for 
HAR are universal and do not adapt to a particular user. With a concept drift approach enabled 
for the application, it should be possible to achieve higher performance with the same level of 
battery consumption. Moreover, depending on a user’s behaviour, the application can adjust a 
time window that defines how long software should keep sensor data that is used by a classifier 
for a particular user. This personalization can improve battery life and application performance.
Bibliography


50 Frugal Learning: Applying Machine Learning with Minimal Resources


[57] Ross Quinlan. Data mining tools see5 and c5.0. 2004. 6


[59] Nikola Rajovic, Pall Carpenter, Isaac Gelado, Nikola Puzovic, and Alex Ramirez. Are mobile processors ready for hpc? In *Supercomput. conf., barcelona supercomputing center, Denver*, 2013. 31

[60] Attila Reiss, Gustaf Hendeby, and Didier Stricker. A competitive approach for human activity recognition on smartphones. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2013)*, 24-26 April, Bruges, Belgium, pages 455–460. ESANN, 2013. 42


[77] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005. 21


Appendix A

Data protocol

This section describes information about data that were collected for HAR task. Data protocol is a document that defines a data collection, sets up rules for processing, and describes the format of data. Moreover, it ensures that the research can be reproduced, if necessary, by other parties. Data were collected from 20 participants with the average length around one minute per activity for every person who engaged in our experiment.

A.1 Purpose of the data collection

Collect data about activities performed by people for studying in the frugal learning project.

A.2 Equipment

Smartwatch with the next specifications:

- Display 1.3 Full Circle P-OLED, 320 x 320 pixels
- Dimensions 45.5 x 52.2 x 10.9mm
- Weight 66.5g
- Total Internal Memory 512MB RAM
- 4GB eMMC
- Bluetooth 4.1 LE
- Strap Stitched Leather, Standard 22mm
- Dust & Water Resistance IP67
- OS Android Wear
- Processor Qualcomm Snapdragon 400, 1.2 GHz
- Sensors 9-Axis, PPG
• Battery 410 mAh

More information is available on the official LG website\(^1\).

### A.3 Environmental setup

This part presents information about the subjects of the data collection experiment and what observation sites were used for collecting a data.

Participants were selected from students of Eindhoven University of Technology with age ranging from 20 to 30 years. They declared their age, height, weight, daily activity level and what sport exercises they do.

Activities that were tracked during the experiment have different requirements for space and accessories. Details about every location that was used are given next.

• Office room in which information about standing activity was collected.
• 50 meters long hall used to collect data for walking activity.
• Staircase with 25 stairs between floors. It connects three floors of a building and used for walking upstairs and downstairs activities.
• Two meter long couch for lying activity.
• Office chair for sitting activity.

### A.4 Data format

CSV file with the measurements from an accelerometer, a gyroscope, a gravity, a linear acceleration, a rotation vector, a magnetic field, an air pressure, and a heart rate sensors. Min, max, mean, range, and standard deviation features are computed for every axis available for a sensor.

### A.5 Forms and documents

The questionnaire has the next set of questions.

• Age;
• Height;
• Weight;
• Daily activity level;
• Please indicate what kind of sport activities you perform (if applicable).
• Consent statement for every participant with permission to use collected data in a study.

\(^1\)http://www.lg.com/us/smartwatch/urbane
A.5.1 Challenges and solutions

Data collected for activities are recorded from sensors in real-time manner. Therefore, it is important to extract only meaningful information that can be used to train classifiers. When a user starts or completes performing activity, she or he presses a button on the wearable device. Pressing a button requires to move a hand with the device closer to the head and rotate the wrist. This action could take around two seconds on average based on data collected from all participants. As a result, we decided to remove five seconds in the beginning and in the end of each activity in order to ignore noise related to pressing a button.

A.5.2 Impartiality, privacy and safety

All data collected with the device is anonymized and cannot be traced to a particular individual. Questionnaire has no fields about a name, an address or similar questions. It consists only from items specified in the section A.5 about the description of the data collection document.
## Appendix B

### Algorithms used in the study

Algorithms and used parameters, blank space in parameters means default values

<table>
<thead>
<tr>
<th>ID and a name of Algorithm on OpenML</th>
<th>Used parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1068.trees.J48</td>
<td></td>
</tr>
<tr>
<td>1070.rules.Ridor</td>
<td></td>
</tr>
<tr>
<td>1071.rules.OneR</td>
<td></td>
</tr>
<tr>
<td>1072.rules.Prim</td>
<td></td>
</tr>
<tr>
<td>1073.rules.PART</td>
<td></td>
</tr>
<tr>
<td>1074.rules.OLM</td>
<td></td>
</tr>
<tr>
<td>1075.rules.NNge</td>
<td></td>
</tr>
<tr>
<td>1076.trees.SimpleCart</td>
<td></td>
</tr>
<tr>
<td>1077.trees.REPTree</td>
<td></td>
</tr>
<tr>
<td>1078.trees.RandomTree</td>
<td></td>
</tr>
<tr>
<td>1079.trees.RandomForest</td>
<td></td>
</tr>
<tr>
<td>1080.trees.NBTree</td>
<td></td>
</tr>
<tr>
<td>1082.trees.LMT</td>
<td></td>
</tr>
<tr>
<td>1084.trees.J48graft</td>
<td></td>
</tr>
<tr>
<td>1086.trees.Id3</td>
<td></td>
</tr>
<tr>
<td>1087.trees.HoeffdingTree</td>
<td></td>
</tr>
<tr>
<td>1088.trees.FT</td>
<td></td>
</tr>
<tr>
<td>1089.trees.ExtraTree</td>
<td></td>
</tr>
<tr>
<td>1090.trees.DecisionStump</td>
<td></td>
</tr>
<tr>
<td>1091.trees.BFTree</td>
<td></td>
</tr>
<tr>
<td>1094.rules.JRip</td>
<td></td>
</tr>
<tr>
<td>1096.rules.DecisionTable</td>
<td></td>
</tr>
<tr>
<td>1098.rules.ConjunctiveRule</td>
<td></td>
</tr>
<tr>
<td>1099.misc.CHIRP</td>
<td></td>
</tr>
<tr>
<td>1100.misc.HyperPipes</td>
<td></td>
</tr>
<tr>
<td>1101.misc.InputMappedClassifier</td>
<td></td>
</tr>
<tr>
<td>1102.misc.OSDL</td>
<td></td>
</tr>
<tr>
<td>1103.misc.VFI</td>
<td></td>
</tr>
<tr>
<td>1104.lazy.IB1</td>
<td></td>
</tr>
<tr>
<td>1105.lazy.IBk</td>
<td>K1, K3, K5</td>
</tr>
<tr>
<td>1106.lazy.KStar</td>
<td></td>
</tr>
<tr>
<td>1107.lazy.LBR</td>
<td></td>
</tr>
</tbody>
</table>
Algorithms and used parameters, blank space in parameters means default values

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lazy.LWL</code></td>
<td></td>
</tr>
<tr>
<td><code>functions.GaussianProcesses</code></td>
<td></td>
</tr>
<tr>
<td><code>functions.KernelLogisticRegression</code></td>
<td></td>
</tr>
<tr>
<td><code>LibLINEAR</code></td>
<td><code>E0.001, E0.01</code></td>
</tr>
<tr>
<td><code>functions.Logistic</code></td>
<td></td>
</tr>
<tr>
<td><code>functions.RBFClassifier</code></td>
<td></td>
</tr>
<tr>
<td><code>functions.SGD</code></td>
<td></td>
</tr>
<tr>
<td><code>bayes.NaiveBayes</code></td>
<td></td>
</tr>
<tr>
<td><code>meta.AttributeSelectedClassifier</code></td>
<td></td>
</tr>
<tr>
<td><code>bayes.A1DE</code></td>
<td></td>
</tr>
<tr>
<td><code>bayes.A2DE</code></td>
<td></td>
</tr>
<tr>
<td><code>bayes.BayesNet</code></td>
<td></td>
</tr>
<tr>
<td><code>functions.LibSVM</code></td>
<td></td>
</tr>
<tr>
<td><code>bayes.HNB</code></td>
<td></td>
</tr>
<tr>
<td><code>functions.KernelLogisticRegression</code></td>
<td></td>
</tr>
<tr>
<td><code>functions.SimpleLogistic</code></td>
<td></td>
</tr>
<tr>
<td><code>functions.SMO</code></td>
<td></td>
</tr>
<tr>
<td><code>functions.SMO</code></td>
<td></td>
</tr>
<tr>
<td><code>functions.SPegasos</code></td>
<td></td>
</tr>
<tr>
<td><code>functions.Winnow</code></td>
<td></td>
</tr>
<tr>
<td><code>AdaBoostM1</code></td>
<td><code>I10, I20, I40, I80, I160</code></td>
</tr>
<tr>
<td><code>meta.AttributeSelectedClassifier</code></td>
<td></td>
</tr>
<tr>
<td><code>Bagging</code></td>
<td><code>I10, I20, I40, I80, I160</code></td>
</tr>
<tr>
<td><code>meta.Dagging</code></td>
<td></td>
</tr>
<tr>
<td><code>meta.Decorate</code></td>
<td></td>
</tr>
<tr>
<td><code>meta.END</code></td>
<td></td>
</tr>
<tr>
<td><code>LogitBoost</code></td>
<td><code>I10, I20, I40, I80, I160</code></td>
</tr>
<tr>
<td><code>MultiBoostAB</code></td>
<td><code>I10, I20, I40, I80, I160</code></td>
</tr>
<tr>
<td><code>meta.RacedIncrementalLogitBoost</code></td>
<td></td>
</tr>
<tr>
<td><code>RandomSubSpace</code></td>
<td><code>I10, I20, I40, I80, I160</code></td>
</tr>
<tr>
<td><code>meta.RealAdaBoost</code></td>
<td><code>I10, I20, I40, I80, I160</code></td>
</tr>
<tr>
<td><code>RotationForest</code></td>
<td><code>I10, I20, I40, I80, I160</code></td>
</tr>
<tr>
<td><code>rules.FURIA</code></td>
<td></td>
</tr>
<tr>
<td><code>trees.ADTree</code></td>
<td></td>
</tr>
<tr>
<td><code>trees.LADTree</code></td>
<td></td>
</tr>
<tr>
<td><code>meta.FilteredClassifier</code></td>
<td></td>
</tr>
</tbody>
</table>
Appendix C

Missing results for classification tasks per an algorithm

Names and the amount, more than 10, of unsuccessful runs per an algorithm

<table>
<thead>
<tr>
<th>Name of an algorithm</th>
<th>Missing values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1109_functions.GaussianProcesses</td>
<td>516</td>
</tr>
<tr>
<td>1072_rules.Prism</td>
<td>491</td>
</tr>
<tr>
<td>1086_trees.Id3</td>
<td>489</td>
</tr>
<tr>
<td>1102_misc.OSDL</td>
<td>486</td>
</tr>
<tr>
<td>1173_bayes.HNB</td>
<td>486</td>
</tr>
<tr>
<td>1181_functions.Winnow</td>
<td>480</td>
</tr>
<tr>
<td>1107_lazy.LBR</td>
<td>461</td>
</tr>
<tr>
<td>1089_trees.ExtraTree</td>
<td>249</td>
</tr>
<tr>
<td>1111_functions.KernelLogisticRegression</td>
<td>121</td>
</tr>
<tr>
<td>1174_functions.KernelLogisticRegression</td>
<td>121</td>
</tr>
<tr>
<td>1180_functions.SPegasos</td>
<td>111</td>
</tr>
<tr>
<td>1195_meta.RealAdaBoost.1160</td>
<td>111</td>
</tr>
<tr>
<td>1120_functions.SGD</td>
<td>110</td>
</tr>
<tr>
<td>1195_meta.RealAdaBoost.110</td>
<td>110</td>
</tr>
<tr>
<td>1195_meta.RealAdaBoost.120</td>
<td>110</td>
</tr>
<tr>
<td>1195_meta.RealAdaBoost.140</td>
<td>110</td>
</tr>
<tr>
<td>1195_meta.RealAdaBoost.180</td>
<td>110</td>
</tr>
<tr>
<td>1199_trees.ADTree</td>
<td>110</td>
</tr>
<tr>
<td>1166_bayes.A2DE</td>
<td>80</td>
</tr>
<tr>
<td>1099_misc.CHIRP</td>
<td>60</td>
</tr>
<tr>
<td>1075_rules.NNge</td>
<td>30</td>
</tr>
<tr>
<td>1196_RotationForest.1160</td>
<td>22</td>
</tr>
<tr>
<td>1187_meta.Decorate</td>
<td>20</td>
</tr>
<tr>
<td>1196_RotationForest.180</td>
<td>20</td>
</tr>
<tr>
<td>1165_bayes.AI1DE</td>
<td>18</td>
</tr>
<tr>
<td>1114_functions.Logistic</td>
<td>14</td>
</tr>
<tr>
<td>1160_meta.AttributeSelectedClassifier</td>
<td>11</td>
</tr>
</tbody>
</table>