

Investigation of Machine Learning Applicability on Mobile Device Diagnostics for Quality Prediction

Edin Daganan & Rasmus Josefsson

Master Thesis
Sony Mobile Communications, Lund University
LTH

June 15, 2016

Abstract

Sony Mobile Communications (SoMC) perpetually collect large amounts of prototype device data into their database which is used for analysing their devices. Different areas are analysed individually using this data but no general model exists which encapsulates the overall performance. A desired model would include information from the different key performance areas and utilise machine learning with the objective to prognosticate the prototype device quality.

This thesis investigates the possibility of using machine learning tools to generalise the data analysis and build a predictive quality model based on historical data. It consists of extracting and processing data from the company database, finding a relevant feature representation and evaluating different models based on both supervised and unsupervised learning methods. The study builds on the assumption that performance is related to the device software and its development, through which conclusions about quality could then be made.

The results show that finding relevant features that distinguish different device software is very difficult, rendering an unsupervised approach inadequate. Apart from deficient features, the supervised methods suffer from the unreliability of using survey data as labels. In conclusion, no explicit model is recommended and different approaches are preferable where another feature representation and more qualitative labels are the main requirements.

Acknowledgements

Thanks to Magnus Oskarsson (LTH) for consistent supervision and valuable inputs.

We want to thank SoMC supervisors Clas Thurban, Mikael Berglund and Andreas Larsson for great support and an inspiring, educating working process. Moreover, we also want to thank Jens Gulin and all other SoMC colleagues who have given us valuable feedback during this thesis.

Contents

1	Introduction	1
1.1	Background	1
1.2	Problem Formulation	2
1.3	Outline	2
2	Theory	5
2.1	Data Processing	5
2.1.1	Database	5
2.1.2	Probes	6
2.1.3	Data Preprocessing	6
2.1.4	Feature Extraction	8
2.1.5	Data Normalisation	9
2.2	Machine Learning	9
2.2.1	Supervised Learning	10
2.2.2	Unsupervised Learning	20
2.2.3	Model Evaluation	22
2.3	Quality	25
2.3.1	Customer Satisfaction	25
3	Methodology	29
3.1	Feature Extraction	29
3.1.1	Precomputed Statistics	29
3.1.2	Raw Data Processing	29

CONTENTS

3.1.3	Employer Surveys	32
3.2	Statistical Analysis	33
3.2.1	Unsupervised Model	33
3.3	Model Implementation	34
3.3.1	Supervised Model Based on Employer Survey Data	34
3.3.2	Supervised Model Based on Software Version	35
3.3.3	Software	36
4	Result	37
4.1	Unsupervised Analysis	37
4.2	Employer Surveys	40
4.3	Supervised Learning Models	40
4.3.1	Survey Model	40
4.3.2	Software Version Model	43
5	Discussion	49
6	Conclusion	53
A	Appendix	55
A.1	Correlation	55
A.2	Additional Results	57

Chapter 1

Introduction

1.1 Background

Sony Mobile Communications (SoMC) are collecting extensive amounts of data from their prototype devices around the clock. Through different so-called probes, events and statistics are logged and reported back to the company database where information is stored. Through the immense size of accumulated data, SoMC wish to examine the possibility of capitalising on machine learning (ML) tools in studying the quality of their devices. This study will be based on several performance focus areas which are used within SoMC for measuring the device quality. Principally being considered independently by different company divisions, this study will interconnect the different areas to evaluate the quality altogether. From regular internal surveys, concerning software quality and covering the areas of interest, labels are provided to enable analysis of the data in a supervised setting.

The data used in this study comes from the internal database where raw data from prototype devices is being stored. To allow for statistical analysis and implementation of ML tools, relevant features need to be extracted from the raw data corresponding to the areas of interest. Furthermore, there is no generic measure of quality established within SoMC which calls for a thorough analysis of the labels that are to be used. The outline and purpose of the data analysis is to determine whether it is possible or not to create a prognostic model which predicts if prototype development conform to the predetermined market release date. More specifically, this project will address the following problems

- Get acquainted with the collected data.
- Exploratory analysis of the dataset.
- Data preprocessing.

- Extract features from the dataset.
- Apply and evaluate several ML algorithms, both supervised and unsupervised, to analyse the data and create a model.
- Train and test the predictability of the model.
- Conclude and compare the most promising models to predict the quality of a device before released.

1.2 Problem Formulation

Given this very broad problem, certain delimitations will be required to render the project feasible within the scope of a master thesis. The main delimitations are: studying devices only of the same platform, considering attributes whose corresponding probes are a mere subset of the entirety and building the labels on survey data, being one out of many SoMC quality measures. Moreover, the project needs methodological restrictions like how feature extraction should be performed or what ML algorithms should be considered.

In order to successfully approach and solve the problem, we will work around the following set of questions

- What data can be extracted and what features capture well the device performance?
- What ML algorithms are appropriate for the data analysis? Is there any suitable model which could be used to build a prognostic model?
- Are the extracted features good and general enough to build a prognostic model upon?
- How reliable are the labels used in the supervised learning setup?

1.3 Outline

The report starts by introducing SoMC concepts such as probes, data warehousing and employee surveys which are presented in sufficient detail for the reader to understand the process of data extraction and the potential difficulties that come along with it. The introduction is followed by a section discussing data preprocessing, including data cleaning and feature extraction. Subsequently, different ML concepts and relevant theory is described from a mathematical point of view.

The ensuing section presents the different approaches to the problems, following the order presented in the background, explaining everything from database

querying to evaluation of implemented ML algorithms. Thereafter, the results from every procedure is presented before drawing conclusions in an individual section. Lastly, the results are thoroughly discussed along with the methods used and suggestions for future studies are presented.

Chapter 2

Theory

2.1 Data Processing

2.1.1 Database

For data storage SoMC utilises Apache Hadoop as a local cluster which originates from Google research projects "The Google File System" paper and "MapReduce: Simplified Data Processing on Large Clusters" published in 2003 and 2004 respectively [27]. It is a cluster for distributed storage and processing of large data sets of different structure. Data is stored in blocks and distributed over the whole cluster which provides significant scalability. The key routine on Hadoop is called MapReduce which assigns tasks to the servers on the cluster where the data in question is being stored [14]. MapReduce is an umbrella term for the two functions performed sequentially on each server, i.e. the map and the reduce task. The map task is exactly like a map in the mathematical sense that its output data is some transformation of the input data. The reduce task then receives the output from the map task and performs some operations, e.g. aggregation and/or statistical computation [13]. In order to access and analyse the stored data, the Apache Hive infrastructure running on top of Apache Hadoop is used, supplying query and analysis functionality. It employs an SQL-based language called HiveQL offering certain table-related extensions, while being limited in other areas compared to standard SQL [28].

The HiveQL language provides a useful function called *join* which operates on two tables in the database, generating its intersection based on a certain key. Multiple join operations can be applied simultaneously such that an arbitrary number of tables can be joined together. Furthermore HiveQL provides extensions to the join operation, e.g. left/right/full (outer) join, which, topologically speaking, generates different compositions of unions and intersections. A broad set of user defined functions (UDFs) exists to facilitate the data extraction as

well as computing basic statistics directly on the cluster such as mean and variance estimators [12].

2.1.2 Probes

SoMC has an internally developed framework which continuously collects data from the devices and is stored on, among others, the Hadoop cluster. Depending on legal prescriptions, certain types of information may or may not be collected from the devices. The prototype devices, used by e.g. the SoMC staff, collect the widest range of data. The different types of data is logged by different, so-called probes. The most general probes log events, consisting of e.g. a time stamp and some measured value, while more specific versions may report already computed statistics. The logging frequency of the probes is strongly linked to the generality of the probe, meaning that a probe collecting information about a very common event will report more frequently than those with precomputed statistics.

The information collected from different probes is stored sequentially in a file locally on the device and based on some trigger, multiple files are then sent concurrently to the cluster. Almost every probe reports along with an event or some reported data, a time stamp based completely on the local time settings of the given device which render comparative time analysis difficult. Upon fatal crashes, certain system updates or similar, the time setting occasionally fails to be restored and instead resets to the UNIX origin, i.e. 1970-01-01 00:00:00 [32]. That implies that events being logged before recuperating a proper time setting will be reported together with an incorrect time stamp. However, data is always stored sequentially such that every event becomes chronologically comparable. Through parameters "import ID", "file number" and "sequence number", the order of occurrence of two arbitrary events can be ordered irrespective of the current time setting of the device. The additional variable "time uploaded" is the time for when an import, typically containing multiple files from numerous devices, is transmitted to the cluster. By definition, this time stamp is an upper bound (in local time of the cluster) for when included events have occurred.

The active probes that are studied within the projects collect information about e.g. battery stamina, system and application crashes, temperature, usage statistics and network stability [23].

2.1.3 Data Preprocessing

The data of interest extracted from the cluster is likely to require different kinds of preprocessing before statistical analysis can be applied. In order to create predictive models with pertinent accuracy the data quality will be of great importance. That includes factors such as accuracy, completeness and consistency which usually are inadequate properties in real-world databases [11].

Inadequacy arise for different reasons, e.g. missing data or incompleteness occurs because data is not always recorded and therefore not available. Such cases occur frequently when working with time series where aggregated statistics are computed for a time frame which is sufficiently small not to capture infrequent events. Data inaccuracy instead depends on factors such as measurement errors, caused by e.g. faulty equipment, incorrect measuring methods or short numerical precision [11]. Inconsistency occurs from e.g. data duplication, changing or using multiple denominations of an attribute or typographical errors. [10].

Four important preprocessing tasks are cleaning, integration, reduction and transformation of the data. Data cleaning consists of removing dirty data, inferring values to null attributes, identifying and treating potential outliers. [11]. The data cleaning procedure is time-consuming but is highly necessary to allow for successful data mining. When possessing a data set which is too large for a complete analysis one could instead sample instances and examine them carefully [10]. It is central, but also difficult, to have a data mining procedure which is robust to incorrect and incomplete data.

A situation that arises frequently in data mining is missing values or null values. Two ways of treating such cases are the following:

- **Ignore the sample:** Measure typically applied to samples with missing labels when the task involves classification.
- **Replace missing values:** What value to impute is unique to each situation. Sometimes values are missing because an event did not occur and in case the variable is a counter, a natural replacement is zero. In other cases a variable may have an underlying tendency in which case statistics like the mean or median is appropriate to replace the missing values with. Yet another way is to use regression, inference measures or decision trees on existing data for replacements of missing values.

Sometimes one might need to merge data from different sources which is when the issue of data integration [11] or data warehousing arises. Different databases will typically record data differently, use different main keys and employ different conventions. Data warehousing is the term for using one general database integration from which all data can be accessed and the absence of it may require considerable amounts of work before the mining process can start. Externally acquired data is sometimes called overlay data and need, just like internal data, undergo a cleaning process before being integrated with the latter [10].

Data reduction consists of dimensionality reduction and numerosity reduction. In the former procedure, data size is reduced while attempting to retain the most relevant information. Examples of dimensionality reduction are principal component analysis (PCA) and manifold learning where data is compressed according to their respective mathematical formulation while attribute subset selection and attribute construction are strategic ways of selecting the most relevant attributes. In the latter procedure, numerosity reduction, data is replaced

by parametric representations like regressions and non-parametric models like histograms, clusters, etc.

Finally, data transformation plays an important role if e.g. distance-based algorithms are going to be applied as they generally perform better when the data is normalised. In methods like discretisation and concept hierarchy, raw data is replaced by intervals or categories [11].

Unreliable result, e.g. caused by over-fitting, is a direct consequence of poor data mining. Reasons for over-fitting could be duplication since repetition of data in many machine learning algorithms would increase the influence of the duplicated data [11, 10].

2.1.4 Feature Extraction

Deciding how the data is appropriately prepared and what features to extract depends on the type of the problem, what tools that will be applied and what variables that exist. The input typically consists of several variables, some of which are numeric, some that are categorical (nominal) and some that are missing/null like mentioned before. The numeric variables can be either discrete or continuous variables as well as the different modelling tools which partition data either discretely or continuously, e.g. binary classification vs linear regression. The difficulty is to find a representation of the data, suitable for the statistical tools applied to the problem.

By using insights about the data and what information is important, one can improve the model or spare significant complexity. A simple example is that if the ratio between variables explain some dependence, it could improve the model by using that particular ratio rather than building the model upon the two (or more) variables used to compute the ratio in question. Specific domain knowledge, can besides increasing performance, also help to cut development time by e.g. reducing the number of variable or choosing a simpler, more appropriate model. Another aspect of the data determining both method and feature selection is the granularity of the data. Granularity is the level of detail in the data set, which can differ between variables and usually needs at least some level of aggregation in order to serve as a feature. The granularity of the input data sets a lower bound on what level of detail the output will have.

There is no general rule for how features are supposed to be selected and it is therefore a decision left to the user. The features extracted will be based on observations and ideas of underlying dependencies and what features that could successfully capture these and help improving the modelling performance [18].

2.1.5 Data Normalisation

Many, but not all, tools and machine learning methods require data to be normalised before training in order to obtain desirable performance. Methods which do not necessarily require normalisation could still benefit from having normalised data [18]. The reason why tools tend to fail with non-normalised data is because attributes with a larger range can have a higher influence on the model, and vice versa. For example, changing the measurement unit, e.g. from hours to minutes, is one way the range of the attribute changes and could introduce bias in a model. In order to prevent this, different methods of normalisation can be applied, among others, min-max normalisation and zero-mean normalisation (also called z -score normalisation). Zero-mean normalisation normalises a variable X based on the variable mean μ_X and variable standard deviation σ_X such that a value $x_i \in X$ is mapped to \hat{x}_i through

$$\hat{x}_i = \frac{x_i - \mu_X}{\sigma_X}, \quad i = 1, \dots, n. \quad (2.1)$$

Zero-mean normalisation is suitable when the actual minimum and maximum of the variable is unknown.

Min-max normalisation is a linear map such that $[m_X, M_X] \ni x_i \mapsto x'_i \in [m_{X'}, M_{X'}]$ where $m_X = \min_i x_i$ and $M_X = \max_i x_i$, $x_i \in X$. The mapping is defined by the following formula

$$x'_i = \frac{x_i - m_X}{M_X - m_X}(M_{X'} - m_{X'}) + m_{X'}. \quad (2.2)$$

There are also variants of zero-mean normalisation where the standard deviation is replaced by other measures such as the mean absolute deviation which renders the estimation more robust to potential outliers [11].

2.2 Machine Learning

Machine learning is an umbrella term for methods that can *learn* to make predictions, estimations or identify certain patterns from data. Instead of being explicitly programmed to perform a task, the computer utilises algorithms that can draw conclusions from past experiences and improve its accuracy throughout the process. A more formal and widely quoted definition of the learning process has been provided by Tom M. Mitchell,

“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T , as measured by P , improves with experience E ”. [17].

A typical machine learning problem consists of an outcome measurement, either quantitative (house prices) or categorical (dog/cat), which one wish to predict given a set of features (such as historical prices or an image). The goal of the algorithm will be to build a model from measured inputs and observed outputs which can predict the output of a new input. Depending on the nature of the problem and the availability of data, the ML problems are often categorised into two groups:

- **Supervised learning:** The algorithm has access to both example inputs and desired outputs which is used to build a predictive model.
- **Unsupervised learning:** There exists no outputs for the example inputs and the algorithm tries to find patterns or dependencies hidden in the data.

The interest in machine learning has literally exploded the past years and these algorithms can now be found in numerous applications, like pattern recognition, image classification, user recommendation systems and self-driving vehicles.

2.2.1 Supervised Learning

A supervised learning task can roughly be described as follows: given a set of observations (\mathbf{x}_i, y_i) , $i = 1, \dots, N$, of inputs (\mathbf{x}) and outputs (y) , train a model to predict the output of a new set of inputs with as high accuracy as possible. There are numerous approaches to solve this task and depending on the nature of the output, the problems are divided into two groups: *regression* when the goal is to predict quantitative outputs and *classification* when the goal is to predict qualitative outputs, referred to as labels or classes. The two groups are closely related and in many cases the model gives a quantitative output although the problem counts as a classification which is solved by mapping the output to a qualitative output. The input can also vary in type, and there can be both quantitative and qualitative input variables for a problem.

The conventional way of denoting the input and output variables are by the symbols X and Y respectively. This uppercase notation refers to the generic aspect of the variables whereas each individual observation, also referred to as *sample*, is denoted with lowercase letters, x_i and y_i , where the subscript i refers to the i^{th} sample. In the most general case, the input and the output are represented by vectors but throughout this thesis only a scalar output will be considered which is common in the case of a classification problem. All the samples are usually collected into two sets, \mathcal{X} and \mathcal{Y} , which are described by

$$\mathcal{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_i \\ \vdots \\ \mathbf{x}_N \end{bmatrix} = \begin{bmatrix} (x_{1,1}, x_{1,2}, \dots, x_{1,j}, \dots, x_{1,p}) \\ \vdots \\ (x_{i,1}, x_{i,2}, \dots, x_{i,j}, \dots, x_{i,p}) \\ \vdots \\ (x_{N,1}, x_{N,2}, \dots, x_{N,j}, \dots, x_{N,p}) \end{bmatrix}, \mathcal{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_N \end{bmatrix} \quad p, N \in \mathbb{R}, \quad (2.3)$$

where each row corresponds to a sample. In the example above, the sets contains of N samples where the input for each sample has the dimension p . Commonly the set of observations is split into three separate sets, a training set, a test set and a validation set denoted as $(\mathcal{X}_{tr}, \mathcal{Y}_{tr})$, $(\mathcal{X}_{test}, \mathcal{Y}_{test})$ and $(\mathcal{X}_{val}, \mathcal{Y}_{val})$ respectively. The two first sets will be used to train and tune the model whilst the validation set will be left untouched until the model is considered to be finished. Then, as the self-explanatory name states, the model performance is evaluated using the validation set.

In the training of the model, the algorithm is fed with input samples, x_i , from the training set and will from them predict outputs, \hat{y}_i . These outputs are then compared with the true outputs, y_i , via a loss function $\mathcal{L}(y_i, \hat{y}_i)$ and the goal of the training is to minimise the loss of all the samples in the training set w.r.t the model parameters θ , i.e

$$\theta_{\min} = \arg \min_{\theta} \sum_i \mathcal{L}(y_i, \hat{y}_i). \quad (2.4)$$

Above, is the loss function defined for each individual sample, but more common is to define the loss function to be the collective loss of all samples in the training set, i.e. $\mathcal{L}(\mathcal{Y}_{tr}, \hat{\mathcal{Y}})$, where $\hat{\mathcal{Y}}$ refers to a matrix with all the predicted outputs. The choice of loss function depends on the application and method, but common choices are the Summed Squared Error (SSE) loss function

$$\mathcal{L} = \sum_i (y_i - \hat{y}_i)^2$$

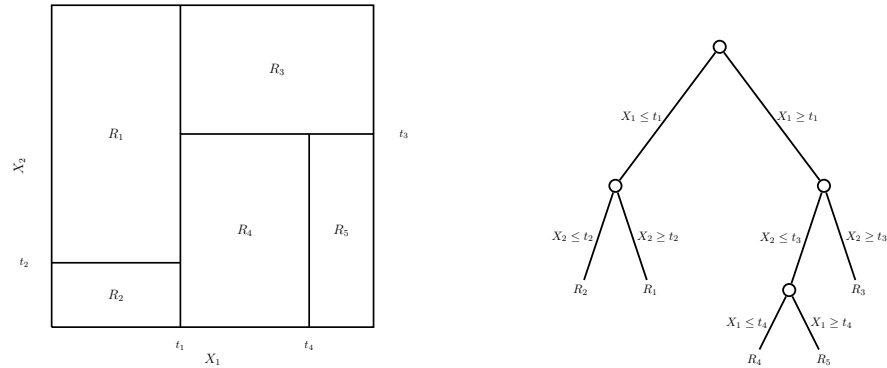
and the cross-entropy loss function (used in logistic regression for binary classification) [30]

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^N (y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)).$$

[9]

Decision Tree

Tree-based methods can be viewed as a machine learning implementation of the parlour game *Twenty questions* [26], in which a player thinks of a subject (or object) and the rest of the players are trying to come up with an answer with the help of 20 "Yes or No"-questions. Like in twenty questions, the tree-based methods are dividing the feature set into smaller sets by applying simple conditions in a step-wise manner. The simple conditions applied will partition the set into rectangles and then each rectangle is fit with a simple model, e.g. a constant or majority voting. Depending on the type of input the conditions applied in each step (each node) may look different, but the common factor is that these conditions split the set into two or more subsets. Example of conditions are; if it is raining or if it is sunny, $x \leq k$ or $x > k$ and $a \in A_i$, $i = 1, \dots, n$. There exists a few different approaches, which all is very similar, but in this paper the CART (Classification and Regression Trees) will be described.



(a) A partitioning of a feature space, done by recursive binary splitting.

(b) The tree corresponding to the partitioning in 2.1a.

Figure 2.1: The partitioning and the corresponding tree to the example explained in the section.

The method is best explained and visualised by using an example. Let Y be the continuous output to a regression model with input variables X_1 and X_2 , each taking values in a closed interval on \mathbb{R} . In this example, only recursive binary partition are considered where the set is divided by inserting border lines parallel to the coordinate axes, i.e. $x \leq t$ or $x > t$. To get a partitioning as the one in figure 2.1a, the set is first divided by the line $X_1 = t_1$ and the two subsets $X_1 \leq t_1$ and $X_1 \geq t_1$ is then further divided by the lines $X_2 = t_2$ and $X_2 = t_3$ respectively. Finally the subset $\{X_1 \geq t_1, X_2 \leq t_3\}$ is split by the line $X_1 = t_4$, which gives a result with 5 regions, displayed in the figure. The choice of variable and splitting point is assumed to be chosen so that the best possible fit is achieved.

Another way of representing the model is by printing out the tree structure, as in figure 2.1b, where the whole data set lies atop of the tree and each branch correspond to each split. The tree is terminated in the regions R_1, \dots, R_5 , when a pre-defined condition is fulfilled. After the partitioning is done and the regions are defined the outcome Y is predicted with a constant c_i in region R_i , i.e

$$\hat{f}(X_1, X_2) = \sum_{i=1}^5 c_i I\{(X_1, X_2) \in R_i\}, \quad (2.5)$$

where $I\{\cdot\}$ is the identity function.

Turning to the question of how the tree should be grown, the algorithm needs to automatically determine the splitting variable and split points for each partition as well as the topology (shape) of the tree. First, consider a training set consisting of N samples, (x_i, y_i) , $i = 1, \dots, N$, from which the algorithm should create a regression tree and an initial partition of M regions, R_1, \dots, R_M . As in the example above, each region is modelled with a constant c_m , i.e. with the function \hat{f} in (2.5). The main goal of the algorithm is now to minimise a loss function and here the analysis is restricted to the case of the sum of squares, i.e. $\sum_i (y_i - \hat{f}(x_i))^2$. For this choice of cost function, the determination of constants c_m becomes simple and the best choice are just the arithmetic mean of y_i in each region R_m ,

$$\hat{c}_m = \text{mean}(y_i | x_i \in R_m). \quad (2.6)$$

The most straightforward way to continue would be to find the binary partition that minimises the sum of squares, but it is generally infeasible. Instead, a greedy algorithm is used where the sum of squares is minimised for each partition. Starting with an initial split defined by a splitting variable j and split point s , the whole data set is divided into two regions

$$R_1(j, s) = \{X | X_j \leq s\} \quad \text{and} \quad R_2(j, s) = \{X | X_j > s\}. \quad (2.7)$$

The algorithm is then searching for the variables (j, s) that minimises the sum of squares over these regions,

$$\min_{j,s} \left(\min_{c_1} \sum_{x_i \in R_1} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2} (y_i - c_2)^2 \right), \quad (2.8)$$

where minimisation over the constants c_i is solved by (2.6), independently of the choice of the variables (j, s) . The outer minimisation is easily done since the determination of an optimal split point for each splitting variable goes fast and scanning through all the j 's can be done in a feasible time. When the optimal pair, (j, s) , is found, the set is split and the procedure is repeated on the two

new subsets. The splitting will continue until a stopping criteria is met, which gives rise to a new question: How large should the tree be? A too large tree will overfit the data and a too small tree will have trouble capturing the essential structures in the data.

A common way of determining the tree size is to create a large tree, T_0 , and then prune this tree according to a method called *cost-complexity pruning*. The initial tree is grown with some simple stopping criteria, like a minimal number of samples in each node. From this tree it is then possible to create subtrees, $T \subset T_0$ by pruning, i.e. remove any number of non-terminal nodes. The terminal nodes in T_0 will be denoted with m , and the number of terminal nodes of each subtree will be denoted $|T|$. The function of interest when pruning the tree is the so-called *cost complexity criterion*

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|, \quad (2.9)$$

where $N_m = |x_i \in R_m|$ is the number of observations in rectangle R_m , \hat{c}_m as in (2.6) and $Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$, referred to as the node impurity. The goal is to find the subtree that minimises this function for a given α . The size of the parameter α will govern the optimal size of T , if $\alpha = 0 \rightarrow T = T_0$ and for larger values on α the tree size will shrink. To find the subtree that minimises (2.9) for a given α the initial tree is pruned using *weakest link pruning* where the internal node that produces the smallest per-node increase in cost is removed and this is repeated until there is only one node left (the root). This creates a sequence of subtrees and of these it is possible to prove that there exists a unique tree, T_α , that minimises the cost function in (2.9) [6]. The estimation of α can then be done and evaluated with k -fold cross-validation, explained in section 2.2.3.

The classification tree, with discrete outputs, is trained in a similar manner and only the function \hat{f} , fitted to the final regions and the node impurity Q_m , need to be modified. Instead of modelling the output with constants as in the regression case, the class proportion is calculated for each terminal region

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k), \quad (2.10)$$

where k refers to each class ($k = 1, \dots, K$) and m to each final node with corresponding regions R_m and number of observations N_m . Each node is then classified according to the majority class in respective node, i.e. $k(m) = \arg \max_k \hat{p}_{mk}$.

There exist different measures for the node impurity but three common choices are,

$$\begin{aligned}
\text{Misclassification error:} & \quad \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}, \\
\text{Gini index:} & \quad \sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk}), \\
\text{Cross-entropy:} & \quad - \sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}.
\end{aligned} \tag{2.11}$$

All three have similar behaviour, but the last two are differentiable which make them more advantageous when working with numerical data. The Gini index and cross-entropy are also more sensitive to changes in node probabilities and endorse pure nodes, i.e. the node contains only one class. This last ability makes them more preferable to use when growing the tree whereas for the pruning either one of the three in (2.11) can be used. [9]

Random Forest

Decision trees are preferred for their simplicity and generality, i.e. its possibility to handle different types of data and problems, but they often lack an important property, namely stability (i.e. high variance) [9]. Even the smallest difference in the data can lead to completely different splits which makes it hard to draw any conclusions from the results. The large variance is due to the hierarchical structure of the tree and an error from early splits is propagated down and growing throughout the tree. A solution to this was presented by L. Breiman, 2001, [5] called *random forests*, which is based on a method called *bagging* (**B**ootstrap **a**ggregating) [4]. Bagging is a method that reduces the variance of a prediction by combining the predictions generated from randomly chosen training sets from the original data set. In the case of random forests, a large number of decision trees are generated and the resulting prediction is achieved by averaging over the trees. The idea is pretty simple and straightforward, and can be summarised with the following algorithm, taken from *The Elements of Statistical Learning*, Friedman, 2001 [9]

Random Forest algorithm

Training:

1. For $b = 1$ to B :
 - (a) Draw a bootstrap sample \mathbf{Z}^* of size M from the training data [29]
 - (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until a stopping criteria is met (minimum number of nodes n_{\min})
 - i. Select m variables at random from p variables ($p =$ input dimension)
 - ii. Pick the best variable/split point among the m chosen variables and split the node according to those.
2. Output the ensemble of trees $\{T_b\}_1^B$

Prediction

Regression: $\hat{f}_{rf}^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x)$

Classification: Each tree makes a *vote*, $\hat{C}_b(x)$, for a class and the final predicted class is achieved by taking the majority of these votes, $\hat{C}_{rf}^B(x) = \text{majority vote}\{\hat{C}_b(x)\}_1^B$

All the randomly generated trees, T_b , are identically distributed which implies that the variance of the average of B trees can be expressed by

$$\text{Var}(\bar{T}_B) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2, \quad (2.12)$$

where \bar{T}_B is the average over B trees, ρ the correlation between each tree and σ^2 the variance for each individual tree. It is evident that the variance will decrease with an increasing B , but there will be a lower bound that depends on the variance and the correlation of the trees. Hence, to further increase the effects of bagging, the random forest algorithm is trying to decrease the correlation between the trees by randomly choose a small set of the input variables, $m \leq p$, to take part in each splitting process of a tree, see step 1-a)-i in the algorithm description above. Typically, the value of m is chosen to be either \sqrt{p} (classification) or $p/3$ (regression), but the optimal value of the parameter will depend on the application.

A common way to evaluate the performance of a random forest is to use *out-of-bag* (OOB) samples. Since all trees are created with bootstrapping, where only a subset of the whole training set is utilised to generate each tree, the

observations not belonging to this subset can be used to evaluate each tree and then taking the average over the prediction rate will give an estimation of the performance. The estimated precision received for the OOB samples is almost equivalent to the precision that can be obtained from N -fold cross-validation, see section 2.2.3. Hence, the random forest can be fit and evaluated in one sequence instead of being trained N times as in the case of N -fold cross-validation. [9]

For further analysis and examples regarding decision trees and random forests see *Ch. 8-10 & 15* in *The Elements of Statistical Learning*, Friedman, 2001 [9].

Support Vector Machines

The support vector machine (SVM), or support vector classifier (SVC), originates from the maximum margin classifier which is an optimally separating hyperplane in the case of a dataset with two linearly separable classes. The SVM is the generalisation to the case where classes are not necessarily linearly separable, i.e. the classes are overlapping. This can be extended to find nonlinear decision boundaries by introducing nonlinear kernel functions and mapping the problem to a higher-dimensional space. Moreover, the two-class problem can be translated to a multi-class problem through a one-vs-all approach which will be introduced in the ensuing subsection. SVMs are suited for solving classification and regression problems and the task of determining model parameters is a convex optimisation problem, meaning that a local optimum is a global optimum.

Before explaining the most general cases we demonstrate the two-class problem in the separable case. Assuming that the training set comprises N vectors $\{(\mathbf{x}_i, y_i), \dots, (\mathbf{x}_N, y_N)\}$ with $(\mathbf{x}_i, y_i) \in \mathcal{X}_{tr} \times \mathcal{Y}_{tr}$, where $\mathcal{X}_{tr} \subseteq \mathbb{R}^d$ and $\mathcal{Y}_{tr} = \{-1, 1\}$. A hyperplane is defined by

$$\{\mathbf{x} \in \mathbb{R}^d : f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = 0\}, \quad (2.13)$$

where \mathbf{w} is the vector of hyperplane parameters having unit length, i.e. $\|\mathbf{w}\| = 1$. Having defined a hyperplane the classification rule is given by

$$G(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x} + w_0), \quad (2.14)$$

where $|\mathbf{w}^T \mathbf{x} + w_0|$ is the perpendicular distance of \mathbf{x} from the hyperplane in (2.13). The separability of the training set implies that $y_i f(\mathbf{x}_i) > 0, \forall i$, i.e. the sign of the mapping of a training vector and its label correspond. In other words, the linear mapping f of a training vector will lie on the side of the decision boundary which corresponds to its class label. The spawning optimisation problem is to find the hyperplane parameters maximising the margin M from the decision boundary to the training points of both classes, which mathematically can be expressed as

$$\begin{aligned} & \arg \max_{\mathbf{w}, w_0, \|\mathbf{w}\|=1} 2M \\ \text{s.t. } & y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1, \quad i \in \{1, \dots, N\}, \end{aligned} \quad (2.15)$$

which is a maximisation problem requiring the quantity M to be maximised. M being inversely proportional to $\|\mathbf{w}\|$, this is equivalent to instead minimising $\frac{1}{2}\|\mathbf{w}\|^2$ which along with discarding the norm constraint on the hyperplane parameters \mathbf{w} gives the following minimisation problem

$$\begin{aligned} & \arg \min_{\mathbf{w}, w_0} \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{s.t. } & y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1, \quad i \in \{1, \dots, N\}. \end{aligned} \quad (2.16)$$

Since the problem is to minimise a quadratic function subject to linear constraints this is indeed a convex optimisation problem. Now having seen how the optimisation problem is established in the case of a dataset with linearly separable classes, we can continue with the extension to the non-separable case before commenting on the actual solving of the optimisation problem and furthermore finding nonlinear decision boundaries based on kernels.

The approach in the non-separable case is to allow for training vectors to be mapped onto the wrong side of the decision boundary, i.e. being in the wrong class. This approach build on slack variables $\xi = (\xi_1, \dots, \xi_N)$ and modifying the constraints in (2.16) to allow for occasional overlapping. The constraint would intuitively take the form $y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq M - \xi_i, \forall i, \sum_{i=1}^N \xi_i \leq K \in \mathbb{R}$ because it measures the absolute distance overlap of the margin from its proper class. This way of modifying the constraints leads however to a non-convex optimisation problem, which can be avoided by instead measuring the relative distance overlap of the margin giving the following constraint

$$\begin{aligned} & y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq M(1 - \xi_i), \quad i \in \{1, \dots, N\}, \\ & \sum_{i=1}^N \xi_i \leq K \in \mathbb{R}. \end{aligned} \quad (2.17)$$

Again defining $M = 1/\|\mathbf{w}\|$, dropping the parameter norm constraint, including the slack variables and rephrasing it appropriately, results in the following optimisation problem

$$\begin{aligned}
& \arg \min_{\mathbf{w}, w_0} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i \\
& \text{s.t. } \begin{cases} y_i(\mathbf{w}^T \mathbf{x}_i + w_0) & \geq M(1 - \xi_i) \\ \xi_i & \geq 0 \end{cases}, \quad i \in \{1, \dots, N\},
\end{aligned} \tag{2.18}$$

where C is a penalty parameter replacing the parameter K that previously bounded the number of misclassifications. Note that misclassifications occur in this setup when $\xi_i > 1$. This formulation covers both the non-separable case ($C < \infty$) and the separable case ($C = \infty$) and solving the constrained optimisation problem is done by considering the corresponding Lagrangian. Without explaining the technique of Lagrangians in detail, it essentially consists of translating (2.18) into an equivalent optimisation problem which can be solved through applying the theory of Lagrangian optimisation and Karush-Kuhn-Tucker conditions. In applying this technique for solving the optimisation problem, so-called support vectors appear which have given name to the classifier.

Finally, attention can be turned towards creating nonlinear classifiers based on SVMs and kernels. The idea is quite simple and consists of first enlarging the original feature space by the application of kernels and subsequently training an SVM, creating a linear classifier in the enlarged space which correspond to a nonlinear classifier in the original space. The enlarged space is typically much larger, possibly infinite, which allow the data to be mapped into a space where it is linearly separable. Before the introduction of kernels, the input features \mathbf{x}_i , $i = 1, \dots, N$ are mapped using basis functions h_m , $k = 1, \dots, M$, i.e. the feature transformation is of the form $h : \mathcal{X} \rightarrow \mathcal{X}$, $h(\mathbf{x}_i) = (h_1(\mathbf{x}_i), \dots, h_M(\mathbf{x}_i))$ and the enlarged feature space $\mathcal{X} \subseteq \mathbb{R}^M$.

In the expression of the resulting separating hyperplane and the so-called Lagrangian dual, being part of the mentioned optimisation technique, the feature transformation h only appears in the form of scalar products $(h(\mathbf{x}_i), h(\mathbf{x}_j))$ in the enlarged space. The property of a kernel is that it can be expressed completely in terms of the scalar product of the enlarged space and the basis functions, i.e. $K(\mathbf{x}, \mathbf{x}') = (h(\mathbf{x}), h(\mathbf{x}'))$. Hence, working with kernels make it possible to compute corresponding scalar products without ever having to explicitly define the enlarged feature space. Some common choices of kernels are

$$\begin{aligned}
K(\mathbf{x}, \mathbf{x}') &= (1 + (\mathbf{x}, \mathbf{x}'))^d, \\
K(\mathbf{x}, \mathbf{x}') &= \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2), \\
K(\mathbf{x}, \mathbf{x}') &= \tanh(\kappa_1 (\mathbf{x}, \mathbf{x}') + \kappa_2),
\end{aligned} \tag{2.19}$$

which are d th-degree polynomial, radial basis and hyperbolic tangent kernels respectively [9, 2].

One vs All Approach

The theory of SVMs presented treats the case of binary classification while many problems require multiple classes. Creating an SVM with $K > 2$ classes can be done in many different ways among which the one-vs-all (or one-vs-the-rest) approach is common. It essentially consists of solving K two-class problems where upon training the k th model, the data points corresponding to the k th class is assigned positive labels while the remaining data points are assigned negative labels. By repeating for all K classes, decision boundaries for each class can thereby be obtained. Note that the approach is possible for all types of SVM, e.g. nonlinear kernels with overlapping classes [9, 2].

2.2.2 Unsupervised Learning

When there are no, or very few examples of labels present in the dataset, there are no longer a way for the supervised learning algorithms to evaluate their predictions which render them impossible to train. Instead unsupervised algorithms are needed which are able to extract structures, patterns and dependencies from the input data. Due to the absence of labels these algorithms have to use other measures to evaluate their performance.

Clustering

Clustering is the problem of trying to divide the data points into two or more clusters which could be thought of as distinct groups. Depending on the clustering algorithm, the way data points are assigned to different clusters are very different. A cluster could be thought of as a subset whose internal distances in some metric are small while distances to points belonging to other clusters are large [2]. Concentrating on unsupervised clustering, the concept is perhaps best explained with a common algorithm called k -means which consists of partitioning data into k predetermined number of clusters by minimising the in-cluster variance.

Assuming a set of unlabelled observations, \mathcal{X} , we denote by $\hat{\mathcal{Y}}$ the corresponding set of predicted cluster belongings. The method seeks to find the partition $S = \{S_i\}_{i=1}^k$ of the full data set which minimises the variance, i.e.

$$\arg \min_S \sum_{i=1}^k \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2, \quad (2.20)$$

where $\boldsymbol{\mu}_i$ is the i th cluster mean. Given an initial set of cluster means, the algorithm alternates between the following steps

1. Given the set of cluster means, minimise (2.20) by assigning each point $\mathbf{x}_j \in \mathcal{X}$ to the closest cluster mean, i.e.

$$C_j = \arg \min_i \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2. \quad (2.21)$$

2. For the given data partition, a new set of cluster means minimising (2.20) are computed

These two steps are then iterated until no point is reassigned to a new cluster. The k -means algorithm is guaranteed to converge but only to a local minimum [9, 2, 22]. Providing k -means as a motivational example, we turn to clustering methods more suitable in this context by emphasising the individual characteristics and giving some brief mathematical introduction to each of them.

We start with a widely used technique called agglomerative hierarchical clustering (AHC) which have some connection to decision trees. In AHC, every data point is initially an individual cluster and then each cluster is iteratively merged together with its closest counterpart. The successive merging of clusters is repeated until only one cluster remains. Obviously, the notion of "closest" imply the need of a proximity measure between clusters. The choice of proximity measure is what defines the certain AHC technique and simple examples make use of usual distances between the closest and furthest points between two clusters, respective. These methods are referred to as single and complete linkage, respectively [22]. In the spirit of k -means, we will employ a different linkage called Ward linkage which in every iteration merges together clusters such that the increase of within-cluster variance is minimal [25].

Another AHC related method called BIRCH (balanced iterative reducing and clustering using hierarchies) is a clustering method working well on very large datasets because it is computationally cheap as it only needs and stores information about the individual clusters rather than information about every single point comprised in it. Being built around a concept called clustering feature (CF) that stores certain cluster information, clusters can be merged iteratively without having to recalculate the CF using every individual data point of the new cluster. The CF is a triple (x, y, z) containing the number of points in the cluster, the linear sum and the square sum of the corresponding points. Considering two clusters defined by $\mathbf{CF}_1 = (x_1, y_1, z_1)$ and $\mathbf{CF}_2 = (x_2, y_2, z_2)$ respectively, merging them results in a new cluster with cluster features $\mathbf{CF}_3 = (x_3, y_3, z_3)$ computed through $\mathbf{CF}_3 = \mathbf{CF}_1 + \mathbf{CF}_2 = (x_1 + x_2, y_1 + y_2, z_1 + z_2)$. The decisions used for the hierarchical clustering is based on other quantities called the centroid, the radius and the diameter of the cluster which can all be computed from the CF. The simplest case which is considered in this work is that the two clusters whose centroids are the closest to each other are merged together to form a new cluster. The iterative clustering structure is stored as a so-called CF tree where each node contains information about the disjoint subclusters constituting the cluster corresponding to the given node. BIRCH leaves the option to the user of defining the desired number of clusters [33].

2.2.3 Model Evaluation

When training a method in a supervised setting the data is conventionally split into a training set and a test set such that the model can be trained on the prior and the prediction error can be estimated from the latter. Since many models have tunable parameters, the procedure is repeated until an optimal choice of parameters is obtained. If the size of the data is large, a validation set is normally set aside and used only for final model evaluation. For example, a 60-20-20 percental partition of the data into a training, testing and validation set respectively could be used if the data size allow for it.

Cross-Validation

When the data size is limited however, it may not be desirable to leave out a part for validation as a reduced training size may drastically affect the model performance. Then, data is instead split only into a training and a testing set, respectively. In this setup, consider cross-validation (CV) to be an estimator of the prediction error of a fitted classification/regression model. Irrespective of the randomness in splitting the data into a training set and a test set, using only one partition for fitting and evaluation will introduce bias in the estimator. Therefore it makes sense to train and evaluate a model multiple times on different training and test sets. Obviously, the test and training sets need to be partitioned disjointly, i.e. any element may not appear in both sets, since the prediction error should be estimated from unseen data. In order to reduce bias and thereby obtain a better estimate of the average prediction error, a strategy called K -fold CV can be applied.

In K -fold CV, the dataset is split into K proportionate, disjoint subsets. The method consists of, in each fold, training a model on the union of $K - 1$ subsets constituting the training set and evaluating it on the remaining subset being the test set. This procedure is repeated K times such that each subset will have been set aside for testing once, or equivalently, each subset is used for training $K - 1$ times. By doing so, the model performance is measured by the average prediction error over the K folds. This reduces the bias of the CV estimator, but increases the variance. The case of training and evaluating the model performance once as described above, corresponds to 1-fold CV. The other extremity of K -fold CV is called leave-one-out cross-validation and is equivalent to N -fold CV, i.e. leaving one sample out in each fold and repeating the procedure N times. Not only does this approach increase the variance of the CV estimator but it may also computationally be the most expensive case of CV. Consequently, selecting K is a bias-variance trade-off for the estimated prediction error but also concerns the computational cost. A convention is to take $K = 10$, i.e. a 10-fold CV [2, 9].

Homogeneity, Completeness & V-Measure

Having class labels available after having clustered data, the V-measure is a way of evaluating the resulting clustering. The measure is the harmonic mean between the measures of homogeneity and completeness just as the common F-measure is the harmonic mean between the quantities precision and recall. Homogeneity measures to what extent the class distribution within an obtained cluster correspond to the actual class distribution. Completeness measures the opposite, making it symmetric to homogeneity, i.e. it measures the rate of how the cluster assignments for a certain class correspond to the actual class distribution. For mathematical formulations and details on the respective measures, see [19].

Silhouette Coefficient

When performing clustering in an unsupervised learning setting, only the properties of the resulting partition can be used in order to evaluate the cluster quality. A measure for this purpose can be obtained from so-called silhouettes which try to quantify the cluster quality by, for each point, computing the average dissimilarity with both its own cluster and the other clusters. A simple way of measuring dissimilarity is by constructing a usual distance matrix using standard Euclidean norm. More precisely, it computes a quantity a_j being the average dissimilarity of a point \mathbf{x}_j to all other points in the same cluster C_i , i.e.

$$a_j = \frac{1}{N_i} \sum_{\mathbf{x}_i \in C_i} d(\mathbf{x}_i, \mathbf{x}_j), \quad (2.22)$$

with $d(\mathbf{x}, \mathbf{y})$ being the dissimilarity of points \mathbf{x} and \mathbf{y} respectively. Moreover, the quantity b_j defines the minimum average dissimilarity over the other clusters, i.e.

$$b_j = \min_k \frac{1}{N_k} \sum_{\mathbf{x}_i \in C_k} d(\mathbf{x}_i, \mathbf{x}_j). \quad (2.23)$$

From these quantities, the resulting silhouette coefficient s_j is composed as follows

$$s_j = \frac{b_j - a_j}{\max(a_j, b_j)}, \quad (2.24)$$

having the property $s_j \in [-1, 1]$. To give meaning to the silhouette coefficient s_j , a few critical values in its range are considered closer

- $s_j \sim 1$: In this case, a_j is small which means that the average dissimilarity within the cluster is considerably smaller than the dissimilarity to the nearest cluster, b_j . This implies that \mathbf{x}_j seems to have been assigned to the correct cluster.
- $s_j \sim 0$: For s_j to be around zero, $a_j \sim b_j$ which means that the dissimilarity to its assigned cluster is approximately the same as the dissimilarity to another cluster. Hence the point \mathbf{x}_j is close to the decision boundary between two clusters.
- $s_j \sim -1$: This is essentially the opposite of the first case, because then the dissimilarity to its own cluster is instead much larger than that of another cluster. This can be considered as having assigned a point to the inappropriate cluster.

Furthermore, the average silhouette coefficient computed as $s = \frac{1}{N} \sum_{j=1}^N s_j$ where N is the number of data points, can be used as an objective function in finding the optimal number of clusters [20].

Dimensionality Reduction

Visualisation is an important part of getting acquainted with a data set. Therefore it is desired to reduce high-dimensional data and find a low-dimensional representation such that it can be displayed in two or three dimensions, e.g. with scatter plots. The difficulty in reducing the dimensions is to preserve underlying structures or features that are essential in further analysing the data. For example, clusters that are perfectly separated in high dimensions are desirably separable, or at least distinguishable, in a lower-dimensional representation. Naturally, it is specific for each data set how dimensions should be reduced in order to preserve relevant underlying structures. There are many existing methods, some of which we will consider more thoroughly.

A common method of reducing dimensions is to apply principal components analysis (PCA). In PCA orthogonal components are obtained from computing the eigenvectors of the correlation matrix. The dimensionality reduction is then the restriction to the subspace spanned by the eigenvectors corresponding to the largest eigenvalues. Or in algebraic terms, the PCA performs a linear coordinate transformation whose principle components are the pairwise orthogonal directions with largest variance. As in the case of SVMs when dealing with nonlinear behaviours, normal PCA can be extended by first applying a nonlinear map and then performing a PCA on the transformed feature space. The method is called kernel PCA since the inner product between the transformed features constitutes the kernel [9, 21].

For mathematical formulations and further explanations of PCA and kernel PCA, consult *A Tutorial On Principal Component Analysis. Derivation, Dis-*

ussion and Singular Value Decomposition, Shlens, 2003 [21] and *Ch. 14.5.4 in The Elements of Statistical Learning*, Friedman, 2001 [9].

The following methods belong to the family of manifold learning which can be viewed as a nonlinear generalisation of linear techniques such as PCA. The name manifold learning refer to the attempt of finding a lower-dimensional manifold in the original feature space. A manifold could be thought of as a structure of lower dimensionality than the original space which the data follows. Manifolds are found in an unsupervised manner, so structures are learnt from the existing data according to some given method.

A method called Spectral Embedding or Laplacian Eigenmaps, uses a graph to describe neighbourhood information and finds an embedding by a spectral decomposition of the so-called graph Laplacian. The method essentially consists of constructing a weighted connection graph and then posing a closely related eigenvalue problem involving the graph Laplacian which is a symmetric, positive semidefinite matrix. The quantity to be optimised for the neighbourhood structure to be preserved in the embedding is such that the eigenvectors corresponding to the smallest eigenvalues will constitute the mapping [1, 9].

Lastly, we consider a technique with a stochastic approach, called *t*-SNE (*t*-distributed stochastic neighbour embedding), which is based on measuring similarities through conditional probabilities of a student *t*-distribution, both in the original space and the sought space. The method tries, like the previous embedding method, to retain lower-dimension embeddings in a higher-dimensional space. The map from high to low dimensions, i.e. that describing the dimensionality reduction, is found by minimising a symmetric version of the so-called Kullback-Leibler divergence over all points in the dataset. This divergence describes the mismatch between the conditional probabilities in the original and transformed feature space, respectively [24].

For thorough, mathematically rigorous explanations of the different dimensionality reduction techniques we refer to the respective sources.

2.3 Quality

The notion of quality is somewhat vague and could be defined in many different ways. Despite there being absolute measures of device quality defined by SoMC they might not correspond to the quality experienced by the customer.

2.3.1 Customer Satisfaction

SoMC is a large global company that operates in a highly competitive market. In such an environment, it is extremely important to be able to keep up with the changeable customer needs, and a way for the companies to do this is by

measuring the *customer satisfaction*. This measure indicates how well the company's products/services are satisfying the expectations of the customers and there exists a formal definition stated by the Marketing Accountability Standards Board

The number of percentage of customers whose reported experience with a firm, its products, or its services (ratings) exceeds specified satisfaction goals. [3]

In comparison to the absolute measures, like sales and market shares which show how well the company is currently performing, the customer satisfaction can indicate how likely the customers are to continue purchasing the company's products/services and hence it is a measure that in some sense can give a hint about how the company will perform in a near future. A dip in customer satisfaction may end up in a decrease in sales and revenues for the company, but it should not be seen as the absolute truth. It should rather be seen as a tool that indicates how the customers' expectation changes over time and it can also highlight certain issues that will help the company further develop their product/service.

There exist various approaches to analysing the customer satisfaction, but a common and acknowledged model was presented by the Japanese professor Noriaki Kano in 1984 [15] which categorises the attributes of a product into 5 groups

- **Must-be Quality:** Attributes that are taken for granted, and the customer will only notice if they aren't fulfilled, for example when a user can't make calls from their smart phone.
- **One-dimensional Quality:** Attributes that can result in both satisfaction and dissatisfaction. These are the attributes that the companies communicate to the customers. An example could be the battery life of a phone, a company promises at least 24 h of battery life, but if the customers experience less it will lead to dissatisfaction or if they experience better it will lead to satisfaction.
- **Attractive Quality:** Attributes that provide satisfaction when fulfilled, but will not bring dissatisfaction when they are not fulfilled. Often attributes that aren't expected of the customers, for example a new update of the software on the phone that will make it possible for the user to unlock their phone with face recognition.
- **Indifferent Quality:** Attributes that don't influence the customer satisfaction, but they can still be crucial for the company. An example could be a chip in the phone which can be produced by numerous manufacturers and for the company it may be important which manufacturer they use

for some reason, e.g. economical, whereas this decision will not affect the customers satisfaction.

- **Reverse Quality:** Attributes that refers to the fact that not all customers are alike, for example will some customers like high-tech product whereas others prefer simple low-tech ones.

This framework can be used to identify important attributes of products and especially the *Must-be* category is of main interest. As the name proposes, the attributes in this category has to be available and if they are not, the product will soon leave the market due to dissatisfaction. The four first categories are often visualised with the graph in figure 2.2

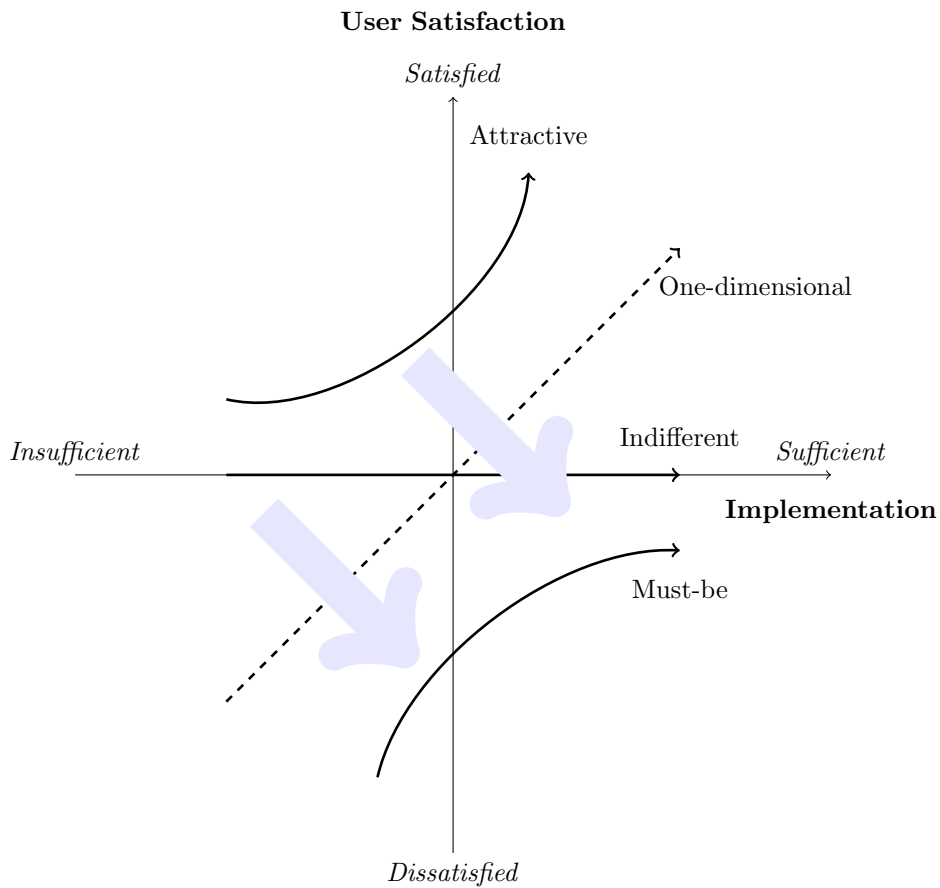


Figure 2.2: An illustration of the Kano model for customer satisfaction.

The big blue arrows in the figure describe how the attribute changes over time. At the beginning, many new features are unexpected by the customers and

will gain high customer satisfaction without working that well. After a while, when other companies have started to implement the feature, the customers will increase their expectations and thereby require higher quality. The whole cycle ends when the feature has become a *Must-be* attribute. Of course all attributes do not follow this cycle, but it is important to remember that the term quality is under constant transformation and something considered good quality yesterday may be considered bad quality tomorrow.

The usual way of measuring the customer satisfaction is by the use of surveys. The surveys could be constructed in numerous ways, but they will often contain questions where the customer is asked to rate certain attributes of the product/service and some questions where the customer gets the chance to leave comments regarding each attribute. [8]

For further descriptions and discussions about customer satisfaction and the Kano model see *How to make product development projects more successful by integrating Kano's model of customer satisfaction into quality function deployment*, Matzler, 1998 [16] and the website of *Marketing Accountability Standards Board* [3].

Chapter 3

Methodology

3.1 Feature Extraction

3.1.1 Precomputed Statistics

Some probes provide precomputed statistics which require none, or very little, processing before qualifying as valid features. An example of such probes contain data like number of installed packages and application usage which update roughly every day depending on the feasibility of the device, e.g. network access. The number of installed packages is directly applicable as feature whereas the probe logging the application usage consists of how many times and the total time that different applications have been placed in the foreground, therefore requiring some treatment before serving as a feature.

3.1.2 Raw Data Processing

Most of the probes comprise raw data, like logs or single events, e.g. crash logs, report of current battery level, user-performed actions, connection or disconnection of the power cable, turning the screen on or off. In these cases considerable data processing is necessary to extract relevant features corresponding to the different areas of interest. Given large datasets the computations are performed on the cluster utilising the MapReduce functionality of Apache Hive. For almost every job, the map task is a unit mapping leaving the originally stored data intact before aggregating and performing computations during the reduce task. For the reduce task, data is distributed over the cluster servers by a set of variables specified in the database query. Depending on what to be analysed the devices will furthermore be divided into different e.g. time periods and software versions such that the desired analysis can be performed. Moreover, keeping

data sorted is essential for the features to be extracted correctly in the reduce task when distributed over the cluster. In order to do so, the self-explanatory variables `time_uploaded` (mappable to `import_id`), `file_nr` and `seq_nr` explained in the probe section, help to keep data chronologically ordered in a simple manner.

Depending on the probe and the complexity of the feature to be extracted, it may be done with or without utilising the MapReduce functionality. A rule of thumb is that tasks requiring conditional statement (e.g. if-else), local variables for tracking or some non-trivial mathematical function need to employ the MapReduce in order to be computed. Another example faced in this project requiring a MapReduce task is when the computation of a feature depend conditionally on data from several probes, e.g. the computation of the average battery consumption takes the screen state, power cable connection and shutdowns into account which all come from different probes. However, simpler tasks based on counting, summing or similar aggregative operations can be performed directly on the cluster by using user-defined functions (UDFs) defined in HiveQL. The UDFs include basic statistical estimators such as mean and variance which can be directly applied on the underlying data if the feature itself is based on numerical data and hence can be directly computed on the cluster. In some situations the combination of MapReduce and aggregative operations/UDFs directly on the cluster is the best alternative to extract a given feature.

With the different functionality and mentioned approaches, we explain in detail how our features are extracted. Different time resolutions will be considered in the project but the resolution in the feature set will always be consistent, meaning that different features originating from time series will be computed over the same period of time.

The feature corresponding to the battery stamina is based upon an earlier internal project of SoMC where the average battery percentage drop over a certain time period is measured while simultaneously computing the rate the screen has been turned on and checking when the device has been connected/disconnected to/from a power source. Altogether it requires data from five different probes and computations are performed using a MapReduce task. We have experimented with different conditions for measurement validity based on stipulations of the original project. This essentially consists of choosing a threshold being the minimal amount of consecutive time, without connection to power source or shutdown, that the battery consumption has to be measured. Depending on the considered time resolution this condition induce a trade-off between data completeness and accuracy. This becomes evident when studying the daily usage (i.e. 24h time resolution) since putting a high threshold, e.g. 14 hours, could generate vast incompleteness as many devices are plugged to a power source more frequently than every 14 hours, thereby rendering the measurements invalid. Choosing a lower threshold, e.g. 1 hour, could instead cause inaccuracy since a time period where the device is static may be measured and therefore not correspond to the usage captured in the other features. It is believed that the original SoMC stipulation, a 14 hour threshold of minimal consecutive mea-

suring without charging, would reflect a daily average battery consumption as it likely captures the routine usage and measures long enough to eliminate short-term noise. However, from initial experiments considering daily usage it gives too much incompleteness and therefore the threshold was modified to obtain more complete data. Mathematically, the feature corresponding to the battery consumption is computed by

$$x = \frac{\sum_{i=1}^n \Delta b_i}{\sum_{i=1}^n \Delta T_i}, \quad (3.1)$$

where Δb_i and ΔT_i is the change in battery level and measured time respectively for measurement i . Note that $\Delta T_i > T_m$, where T_m is the predefined threshold corresponding to some minimal time of consecutive measurement without charging or shutdown.

Moreover, features corresponding to the application usage are extracted based on a probe reporting daily aggregated statistics comprising, for each application used, the number of times and total duration that it has been in the foreground. We do not distinguish between the different applications but instead aggregate these statistics to obtain both the total and average number of times and duration respectively that applications have been in the foreground, hence constituting four features in total. Likewise, these features are computed employing the MapReduce functionality. The features corresponding to the total number of times and total duration are straightforward while the average number of times and average duration is computed according to the formulae

$$m = \frac{1}{n} \sum_{i=1}^n n_i, \quad (3.2)$$

$$x = \frac{1}{n} \sum_{i=1}^n T_i, \quad (3.3)$$

where n_i is the total number of times and T_i is the total duration that applications have been in the foreground for time i and n is the total number of days. Note that since the originating probe provides pre-computed daily statistics we cannot get a higher time resolution.

The features corresponding to application start-up times are computed by average after numerical values are extracted from a probe reporting these times along with related information in a nested structure. Depending on the time resolution we consider either a single feature comprising all applications or we distinguish between applications and compute a unique feature corresponding to the start-up time of each of the most popular applications. The number of applications whose start-up times are to be computed depend on the resulting

completeness, which just like before diminishes with increasing time resolution (i.e. shorter period of measurement). Therefore, over a longer time-span more applications are likely to have been started so that start-up times exist and therefore a larger number of corresponding, unique features can be used with an acceptable level of completeness. Irrespective of the time resolution, the values of interest can be extracted and mean or variance estimations can be done directly on the cluster. In the case where start-up times of certain applications each correspond to a unique feature they are computed as the mean of each start-up time respectively. In the case of higher time resolution where all start-up times constitute a single feature, it is computed in two steps as follows. First the relative standard deviation in start-up time from the mean of the whole population (i.e. zero-mean normalisation) is computed for each application and observation before taking the mean over all observations for each application, i.e.

$$y_i = \frac{1}{\sigma(X_i)} \sum_{j=1}^n \frac{x_i^j - \mu(X_i)}{n}, \quad (3.4)$$

where x_i^j is a single observation of the start-up time for application i and n is the total number of observations. The final feature is obtained by simply taking the average of all these quantities computed for a device, i.e

$$y = \frac{1}{m} \sum_{i=1}^m y_i. \quad (3.5)$$

Finally, features concerning the crashes and the network stability are computed by simply counting the number of occurrences of these events over the measured period of time. The features corresponding to the crash frequency are split into four individual features depending on the origin of the crash, which in our case could be either the system, the modem, any of the SoMC applications or in any third party application. In the case of the network stability, only the availability to make calls is considered which is expressed by two features, one that is the total number of phone calls that has been conducted and another one that is the number of times an active phone call has been disrupted. Both the crash and the network features are computed with the predefined functions that the Hive interface provide.

3.1.3 Employer Surveys

SoMC has a test program for its employees where they receive mobile devices which they are supposed to use regularly. Regular surveys, typically distributed upon new software releases, are also part of the program where users get to

quantify the performance and perceived quality of the device. The surveys which have the highest participation are corresponding to contemporary projects.

The surveys cover a wide range of questions, around 20 in total, concerning both hardware and software quality. The questions can differ between different surveys within a project and added questions usually relate to changes from a previous software version. Only a subset of questions covering the perceived quality (e.g. battery performance, responsiveness, network stability) will be of interest for this study and are usually consistent as they are core questions throughout projects. The surveys corresponding to the projects considered in this study are released approximately every fourth week. In nearly all questions the user is asked to quantify the perceived quality on a discrete scale and has the options of leaving additional comments. Surveys are taken in by a third party software under the employment identity which does not allow direct connection to the device of the user and its data. Through the company intranet employment identities can be linked to the IMEIs (International Mobile Station Equipment Identity; unique cell phone identity tag [31]) of the devices used by that employee, in order to connect the survey responses with the device data.

The subset of responses taken into consideration are those from questions having the best correspondence with the focus areas of the study, e.g. battery performance, stability. The optional text answers are omitted as they appear very sporadically and are difficult to process and eventually analyse, instead focus is turned towards the quantitative responses (numerical values) which are complete and simpler to build a model upon.

3.2 Statistical Analysis

Before identifying any characteristics in the feature representation of the data, some fundamental treatment needs to be done. This includes identifying and removing faulty or corrupt data and obvious outliers that may have an important impact on the overall statistics.

3.2.1 Unsupervised Model

In trying to find patterns and structures in the extracted data, we will apply several unsupervised learning models. The aim of the unsupervised approach is to examine if our features follow any pattern related to different software branches or software versions through which anticipatory improvement would show. Consequently, the data considered in this setup are observations with features being statistics aggregated over each software version such that there may exist multiple observations for each device, but corresponding to different software versions. The feature space is 11-dimensional and each feature is either precomputed statistics or computed according to the section 3.1.2. Before

turning to the clustering, we start by filtering out incomplete data, observations with faulty values and apparent outliers.

We then scrutinise the data through scatter plots between the different pairs of features. From these observations we then narrow down the set of clustering algorithms to those that are likely to suit our data. We first apply AHC with Ward linkage and optimise the number of clusters using the average silhouette coefficient as objective. Subsequently, we visualise the data by using several dimensionality reduction techniques. We then repeat the process instead applying BIRCH.

Lastly, we examine if the obtained clusters are pertinent to the different software branches by using evaluation measures homogeneity, completeness and the V-measure score.

3.3 Model Implementation

As a first attempt to create a predictive model for the quality, a supervised learning model is created in which the responses collected in the employer surveys are used as labels. In addition to this model, another supervised learning model with labels based on the software version is tested in an attempt to investigate how well our features represent the development of the software versions.

3.3.1 Supervised Model Based on Employer Survey Data

In the following supervised learning setup, the survey responses regarding the overall perceived quality will constitute our labels. The training data will be based upon the daily aggregated statistics, either collected directly from the probes according to subsection 3.1.1 or computed as described in subsection 3.1.2. Hence, each observation in the dataset characterises the daily usage of a device. This means that the data set will contain multiple observations corresponding to the same device, but each observation will correspond to different days. An intuitive reason for choosing daily statistics to compose our features is that device usage is likely to be periodic with a 1 day period. Another motivation for choosing a daily time frame is that occasional incompleteness, i.e. observations having features with missing values, is not a crucial issue. Discarding an observation corresponding to the daily usage of a device, due to incompleteness, would not necessarily create any problems as there would in most cases be enough observations left corresponding to that same device. We have 9 different features in our feature set, comprising the different attributes of interest. We decide to filter out those samples that have three or more missing/faulty values and those samples missing only one or two values will be replaced by the mean value of that feature for the corresponding device.

We start by letting each observation in the data set constitute a training pair

(\mathbf{x}, y) with $\mathbf{x} \in \mathcal{X}_{tr}$ being the observation and $y \in \mathcal{Y}_{tr}$ being the label of the device corresponding to that sample. That is, we have training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ built on the n observations where y_i is the label of the device corresponding to observation \mathbf{x}_i . We then train a random forest classifier on the data and evaluate the model. We will perform the evaluation by a slightly modified variant of cross-validation since randomly partitioning the dataset would be fundamentally wrong. In doing so, observations corresponding to a certain device may occur in both the training and the test set. Therefore we partition the set of devices into a train and test set and subsequently fit the model on training pairs corresponding to the training devices and finally we evaluate the model on the test set. Note that we use the model to predict the label of all observations corresponding to the devices belonging to the test set, but we are explicitly interested in predicting the perceived quality of those devices individually. So for each device, we set the perceived quality to be the most common label predicted from its observations, i.e. a majority vote.

We stick to the same methods but perform a variant of data augmentation in an attempt to increase model accuracy. The motivation behind the approach is that multiple experiences may altogether determine the perceived quality. Moreover, the internal order in which those experiences occurred may not be of importance. Having this idea in mind we form a new, larger feature space with 27 dimensions by concatenating 3 observations from the original feature space, corresponding to a certain device. Having posed the assumption that the internal order of observations is insignificant, we create all possible permutations of observations corresponding to a certain device, thereby increasing the number of training pairs for each device by a factor $\frac{1}{m} \binom{m}{3}$, with $m > 3$ being the number of observations corresponding to that device. The model evaluation is performed similarly, i.e. creating a partition based on the devices giving a training and a test set whose corresponding observations are then used to train and evaluate the model, respectively.

3.3.2 Supervised Model Based on Software Version

Instead of trying to predict the quality of each individual device in the data set, as in the model above, another model is considered where the behaviour of each software version is of interest. The daily aggregated features used in the model above are still considered, but the representation of the observation pairs (\mathbf{x}_i, y_i) in the data set are altered. Rather than connect each observation to a certain device and its grade from the surveys, an observation will be represented by its software version and all information regarding from which device the data comes from is omitted. The decoupling from the survey data makes it possible to extend the numbers of observations in our data set with data collected at a later date. However, only the software versions corresponding to the original branch and software versions that are represented by at least 5 observations are considered which will make the gain of including more data less noticeable.

In the initial setup of the model, the software versions were intended to be used as labels for the observations but due to the high number of software versions and the lack of examples of each software version other representations are needed. Instead, two other representations are used, one where the observations will be divided according to the major branches of the software tree and an even more coarse partitioning where two major branches are merged together into a single class. With these two representations established, a random forest classifier is trained and the performance of the classifier is evaluated with both OOB samples and 10-fold cross-validation, for more details regarding the methods see section 2.2.1. To improve the precision of the classifier, some of the parameters of the classifier are tuned until optimal values are obtained.

A great attribute of the random forest classifier is the possibility to get an estimate of how much each input variable (feature) influenced the classification. Based on this estimation, some of the features are studied more closely with focus on the correlation between them and the class belongings. To further investigate the impact of each feature on the resulting classification, new models are created where some or many of the features are omitted before a random forest classifier is applied. The performance of each model is evaluated in the same manner as before, i.e with OOB samples and 10-fold cross-validation.

3.3.3 Software

Most of the statistical analysis and all the machine learning implementation is done in *Python*, or more specifically *Python 2.7.11*. The code is written in an *iPython notebook*, which we ran locally on our computers at SoMC. For preprocessing and visualisation of the data, common libraries like *numpy*, *matplotlib*, *scipy* were utilised whereas all of the machine learning was implemented with the packages available in the *scikit-learn* library [7].

Chapter 4

Result

4.1 Unsupervised Analysis

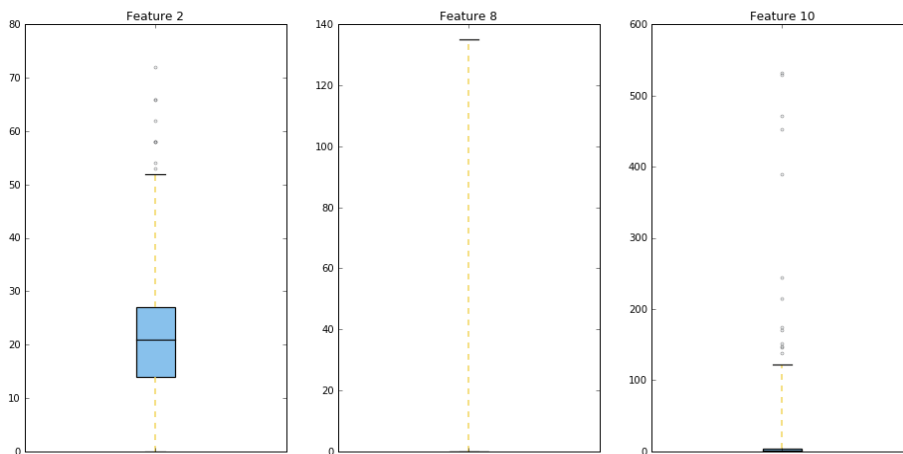


Figure 4.1: Boxplots displaying statistics for feature 2, 8 and 10 respectively. The whiskers display the $\alpha/2$ and $1 - \alpha/2$ quantiles with $\alpha = 0.01$ and the dots are values outside the whiskers, which we will classify as outliers. Variable 8 is a discrete variable with median 0 and maximum/minimum values corresponding to the respective quantiles. Variable 10 is also a discrete variable which by contrast has many distant outliers.

We start in an unsupervised setup by considering the basic statistics and try to examine potential structures of the data. In this case, the dataset consists of devices whose features are based on aggregated statistics over a certain software

version. Note that the same device ID may occur more than once in the dataset, but then with a different software version. The data has 11 features, meaning that each data point is represented by an 11-dimensional vector in the feature space. When extracting features according to the given criteria (e.g. minimal time of measurement), 2766 devices with given software meet the qualifications. Before individually visualising the statistics of each feature, we filter out devices with apparent inaccuracies such as negative values for features corresponding to positively defined attributes or features defined by proportions whose values are larger than 1. These issues arise for reasons such as resetting of device timing, causing feature extraction of temporal kind to generate faulty values. For the dataset, being relatively large, we decide to filter out all devices with missing values to not introduce bias in a model by imputing identical values for one or multiple feature across the whole dataset. The remainder after having filtered these devices out is a dataset consisting of 2558 devices. We scrutinise the statistics by producing boxplots corresponding to different attributes, see figure 4.1

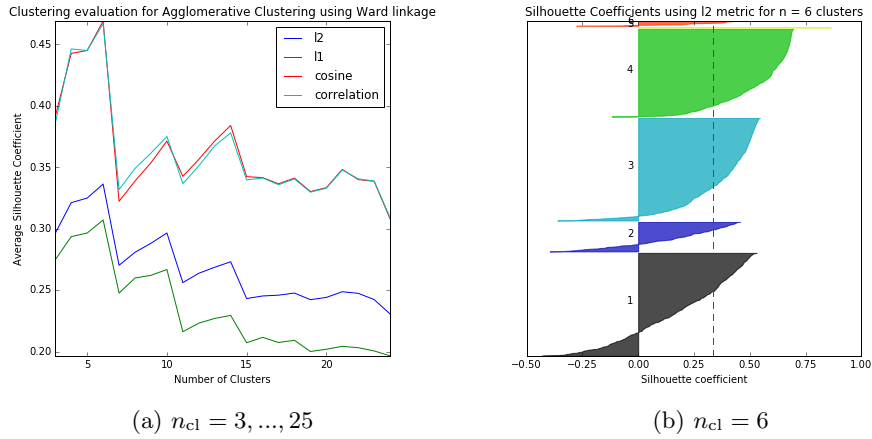


Figure 4.2: Evaluation of the AHC (Ward linkage) algorithm by calculation of the silhouette coefficients. In the left figure is the average of the silhouette coefficients plotted as a function of the number of clusters and the different colours represent different measures used in the calculations of the silhouette coefficients. The right figure displays the silhouette coefficients for each sample obtained with $n_{cl} = 6$, for which the highest average of the silhouette coefficients was received. Note that there are two very small classes.

We proceed by filtering out the devices having at least one feature outlier, i.e. a variable with value outside the whiskers as shown in figure 4.1. After having filtered out these outliers we possess a set of 2428 devices used for further analysis. In figure A.1 of the appendix we illustrate the final feature set through scatter plots for all combinations of features together with the corresponding correlation matrix, also presented in equation (A.1) of the appendix.

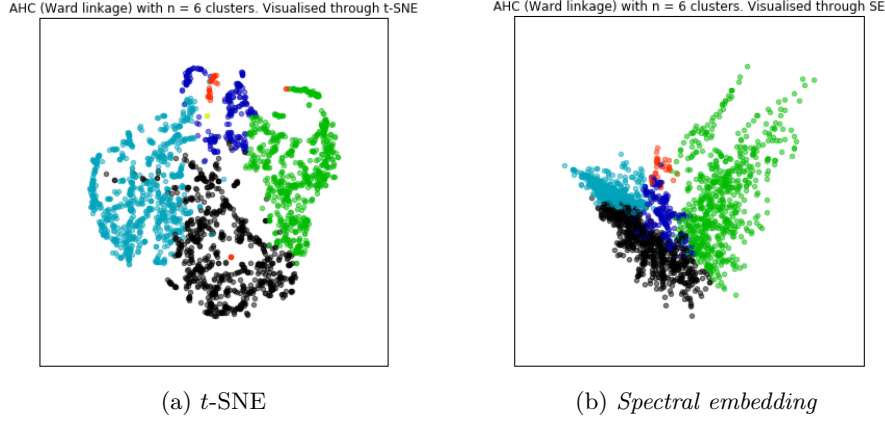


Figure 4.3: The clusters computed with the AHC (Ward linkage) algorithm ($n_{cl} = 6$), see figure 4.2b, visualised with the help of two dimensionality reduction techniques, *t-SNE* and spectral embedding.

Homogeneity score	0.022
Completeness score	0.019
V-measure score	0.020

Table 4.1: Different classification scores used to evaluate the relevance between the clusters and the software branches.

After having a processed dataset, we attempt to cluster the data using different methods. First, we apply AHC with Ward linkage which needs the pre-determined parameter n_{cl} which is the number of clusters. We use the average silhouette coefficient as objective function with several metrics in optimising the parameter n_{cl} . The resulting optimisation is displayed in figure 4.2a along with the silhouette coefficient for each sample for the optimal number of clusters, shown in figure 4.2b.

We attempt to illustrate the clustering and the data in general by performing several dimensionality reductions. We apply *t-SNE* and spectral embedding respectively and display the resulting 2D scatter plots with cluster belongings in 4.3.

A corresponding analysis yielding similar results using BIRCH is displayed in the appendix. Once having performed the unsupervised clustering, we wish to examine if the obtained clusters have any relevance to the device software. There are 7 software branches (one being almost negligible) and we associate each data point with its software branch and calculate several classification scores. The scores are displayed in table 4.1.

4.2 Employer Surveys

In the surveys there are nine questions that concerned attributes of the device related to the software, e.g the camera, battery life, software ease of use etc, which are of interest in our project. As mentioned in subsection 3.1.3 , only the quantitative responses (numerical ratings) will be considered and we will also restrict our analysis to four consecutive surveys, released in 4 weeks intervals. The number of responses for each survey is displayed in table 4.2.

<i>Survey</i>	1	2	3	4
<i>Nbr of users</i>	40	128	102	78

Table 4.2: The number of responses for each of the four surveys.

The distributions of the ratings of each of the nine attributes related to the software are visualised in figure 4.4 with so called box plots. The black line in the middle of the boxes corresponds to the *median* of the set whereas the *whiskers*, the black lines at the very end of the dashed lines, correspond to the maximum and minimum value of the distributions.

In each survey, the users were also asked to rate the overall perceived quality of the device and the resulting distribution is displayed in figure 4.5, where the left figure consists of box plots describing the distribution of the grades for each survey and the right shows an example of a typical distribution of the grades.

4.3 Supervised Learning Models

We have considered a range of supervised learning methods but we narrow down the final analysis to a couple of methods which we apply in two independent cases with different purposes. For the first approach the survey responses constitute the class labels in attempting to create a model for perceived quality prediction while the other aims to create a model for predicting the software version.

When evaluating a classification model based on supervised learning, it is common to present the results in a confusion matrix. The entry at (i, j) of a confusion matrix corresponds to the number of observations which belongs to class i but is classified as j . A correctly classified observation belonging to class i will therefore contribute by a count of 1 to entry (i, i) of the confusion matrix.

4.3.1 Survey Model

In the supervised learning setup as described in the section 3.3.1 we have two different approaches, first using the original feature set and secondly applying

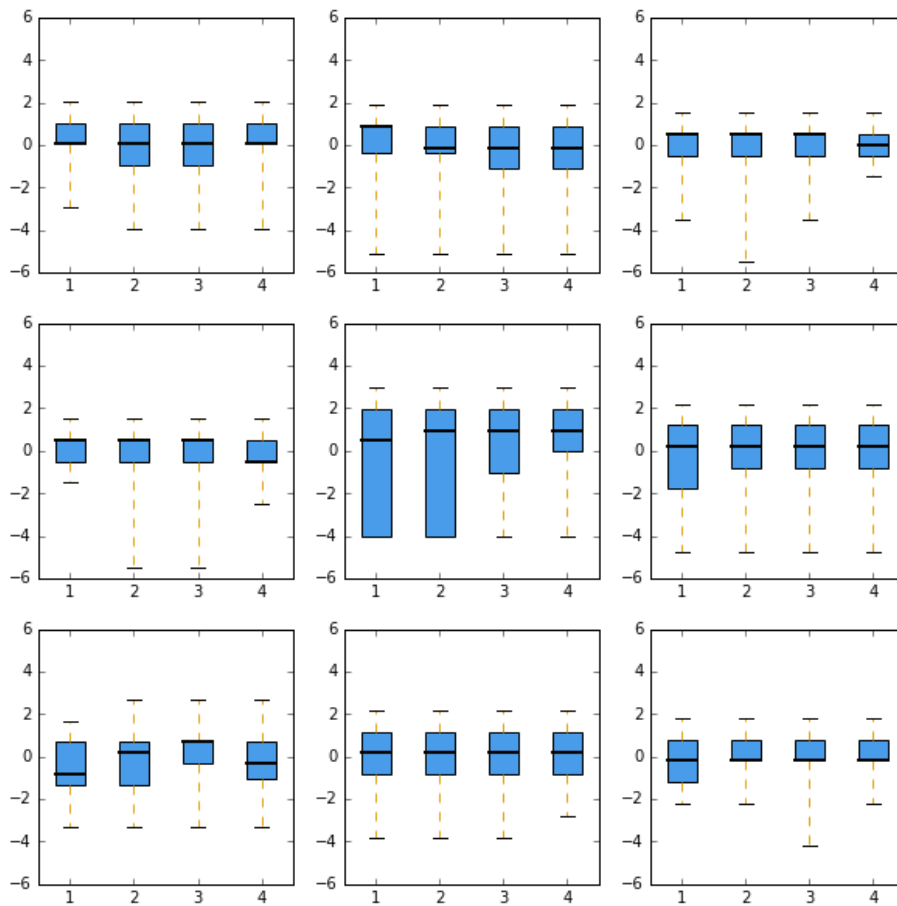


Figure 4.4: Box plots of the grades corresponding to the perceived quality of relevant attributes (e.g battery, camera etc) asked about in the surveys. Each plot corresponds to an attribute and in each plot the result of four surveys are displayed in chronological order. The *whiskers* of the boxes correspond to the maximum and minimum value of the corresponding distribution.

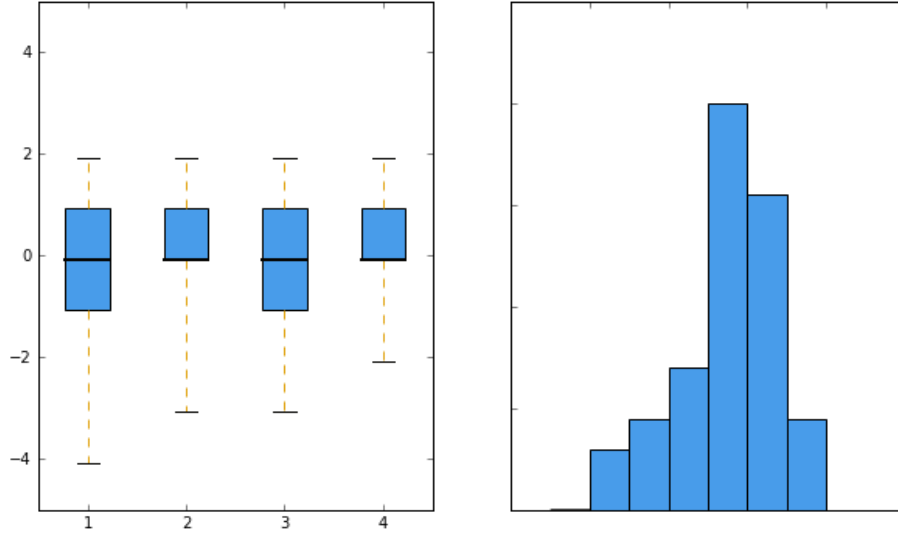


Figure 4.5: The left figure contains a box plot of the grade describing the overall perceived quality for the same surveys as in figure 4.4 and to the right is a histogram that displays a typical distribution of the grades.

data and feature augmentation. The survey responses from survey 2, presented above (figure 4.5), constitute our labels used in the learning of the method.

In the first case, the feature set is built on statistics from a daily time frame and before creating a model, outliers and corrupt data is filtered out before being normalised using min-max normalisation. We subsequently implement a random forest model and evaluate it using our customised variant of N -fold CV where all observations corresponding to an arbitrary device will be left out in one fold for evaluation. The predicted label for the device is then a majority vote among the predicted labels of each observation. Our model is based on a dataset with 756 observations from 52 unique devices and the results are displayed in table 4.3a.

Given that the labels typically follow a distribution as displayed in figure 4.5, we try to compensate for the given label preponderance by adjusting the penalty parameter of each class to be inversely proportional to its class frequency. The resulting model is displayed in table 4.3b. We attempt the same analysis but instead applying an SVM, yielding similar results that are presented in the appendix.

We continue with the case of a larger feature space and augmented data, now having 203508 observations corresponding to the same 52 devices used previously. We proceed exactly the same way, i.e. by training and evaluating a random forest classifier using our modified N -fold CV and performing a ma-

	A	B	C	D	E	F	G		A	B	C	D	E	F	G
A	0	0	0	0	0	0	0	A	0	0	0	0	0	0	0
B	0	0	0	0	0	1	0	B	0	0	0	0	0	1	0
C	0	0	0	0	1	7	0	C	0	0	0	0	1	7	0
D	0	0	0	0	0	8	1	D	0	0	0	0	0	9	0
E	0	0	0	0	5	12	1	E	0	0	0	1	6	10	1
F	0	0	0	0	2	9	4	F	0	0	0	0	2	9	4
G	0	0	0	0	0	1	0	G	0	0	0	0	0	1	0

(a) Without regarding any class priors.

(b) With the class priors taken into account.

Table 4.3: The two confusion matrices obtained from evaluating the model performance of two random forest classifiers, both with $n = 50$ trees and $m = 3$ splitting features through our customised N -fold CV.

majority vote over the predicted labels from the observations corresponding to a device. The confusion matrices in the two cases (direct application of the random forest and taking the class priors into account, respectively) are presented in tables 4.4a and 4.4b.

	A	B	C	D	E	F	G		A	B	C	D	E	F	G
A	0	0	0	0	0	0	0	A	0	0	0	0	0	0	0
B	0	0	0	0	0	1	0	B	0	0	0	0	0	1	0
C	0	0	0	0	1	7	0	C	0	0	0	0	1	7	0
D	0	0	0	0	1	7	1	D	0	0	0	0	1	8	0
E	0	0	0	0	5	13	0	E	0	0	0	0	4	14	0
F	0	0	0	0	2	9	4	F	0	0	0	0	2	10	3
G	0	0	0	0	0	1	0	G	0	0	0	0	0	1	0

(a) Without regarding any class priors.

(b) With the class priors taken into account.

Table 4.4: The two confusion matrices obtained from evaluating the model performance through our customised N -fold CV of two random forest classifier on the enlarged feature space with $n = 50$ trees and $m = 3$ splitting features.

4.3.2 Software Version Model

To investigate how well our features can describe the software of the devices, we replace the labels received from the survey data by labels based on the software versions. The same features as in the previous model will be used which represented a day of usage and with the same preprocessing and filtering applied as before. Since the labels no longer are dependent on the surveys we will extend the dataset with more data that are collected after the surveys, but still from devices within the survey programme. An unit in the new model

will be represented by a day of usage and the label, as mentioned above, will be based on the software version. An additional filtering is also done to the dataset where the software versions not belonging to the main branches and the software versions that are represented by less than 5 days are filtered out.

<i>software version</i>	<i>Nbr of units</i>	<i>%</i>
1.1	77	2.36
1.2	81	2.48
1.3	109	3.34
1.4	42	1.28
1.5	58	1.77
1.6	73	2.23
1.7	107	3.28
1.8	150	4.60
<hr/>		
2.1	137	4.20
2.2	79	2.42
2.3	74	2.26
2.4	73	2.23
2.5	119	3.65
<hr/>		
3.1	203	6.22
3.2	49	1.50
3.3	296	9.07
3.4	144	4.41
3.5	237	7.26
<hr/>		
4.1	18	0.55
4.2	66	2.02
<hr/>		
5.1	29	0.88
5.2	206	6.31
<hr/>		
6.1	28	0.85
6.2	59	1.80
6.3	83	2.54
6.4	152	4.66
6.5	59	1.80
6.6	146	4.47
6.7	65	1.99
6.8	121	3.71
6.9	120	3.68
<hr/>		
<i>Total</i>	3260	

(a) The distribution of units according to the software version.

<i>software Branch</i>	<i>Nbr of units</i>	<i>%</i>
1.x	697	21.4
2.x	482	14.8
3.x	929	28.5
4.x	84	2.58
5.x	235	7.21
6.x	833	25.6

(b) The distribution of units for each of the major branches.

<i>software groups</i>	<i>Nbr of units</i>	<i>%</i>
A (1.x,2.x)	1179	36.2
B (3.x,4.x)	1013	31.1
C (5.x,6.x)	1068	32.8

(c) The distribution of units when two major branches are seen as the same class.

Table 4.5: Three different ways of dividing the dataset into groups, i.e the units are assigned different labels. The software versions are named so that the first number represents the branches of the software and the second number describes the chronological order of the software versions within a certain branch.

With this setup, the dataset contains 3260 units distributed over 31 software versions according to table 4.5a. Since there is a large number of different software versions apparent in the dataset and hence a low number of examples of each of the software versions, it is likely that it would be hard to fit any model with high precision to the dataset using the software versions as class labels. Instead, alternative representations are created according to the tables on the right hand side of table 4.5. In table 4.5b, the distribution of the units for each of the major software branches are shown and in table 4.5c, the units are divided into three equally large groups by joining two branches with each other.

As a first try, we use the representation stated in table 4.5b to train a random forest model, evaluate it with a 10-fold cross-validation and then collect the results in a confusion matrix, see table 4.6. The result obtained with this representation looks promising, with an overall precision of 0.602, but the precision is low for the two classes with a small number of examples in the dataset which may indicate that another representation is to prefer.

	1	2	3	4	5	6
1	0.659	0.083	0.197	0.000	0.001	0.060
2	0.216	0.351	0.359	0.000	0.002	0.073
3	0.154	0.082	0.587	0.008	0.018	0.152
4	0.060	0.024	0.333	0.119	0.131	0.333
5	0.047	0.034	0.247	0.013	0.260	0.400
6	0.028	0.018	0.091	0.005	0.010	0.849
Total	1.07	0.68	1.10	0.286	0.421	1.26

Table 4.6: The resulting confusion matrix obtained from a 10-fold cross-validation applied to a random forest with $n = 200$ trees and $m = 7$ splitting variables when the label of the units is based on the major software branch. The last row is the ratio between the number of predicted units of a class and actual numbers of units of that class.

We repeat the same procedure for the other representation, stated in 4.5c, and the resulting confusion matrix is displayed in table 4.7. The overall precision is increased, as expected due to fewer classes, 0.705, but more importantly the precision of each individual class is higher (or at least equal) than in the previous case. In addition to the precision, the importance of each feature for the classification was calculated, see figure 4.6. From the figure, it is evident that $Var\#8$ and $Var\#1$ have a big impact on the classification done by the random forest and we chose to plot the values of these two variables for all units, see figure 4.7, where the units are ordered along the x-axis according to their class belongings.

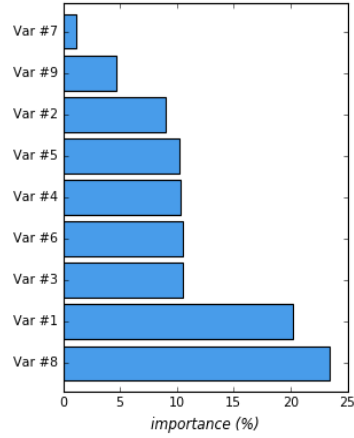


Figure 4.6: The variable importance for the random forest classifier used in the case of three classes.

	A	B	C
A	0.770	0.162	0.068
B	0.311	0.479	0.210
C	0.051	0.107	0.842
<i>Total</i>	1.08	0.780	1.12

Table 4.7: The resulting confusion matrix obtained from a 10-fold cross-validation with a random forest with $n = 200$ trees and $m = 7$ splitting variables in the case of three classes.

In an attempt to see how well these two variables can describe our representation, we split the original feature set into two distinct sets, one which will contain *Var#8* and *Var#1* and another set that contains the rest of the features, and are then fitting two separate random forests classifiers to the subsets.

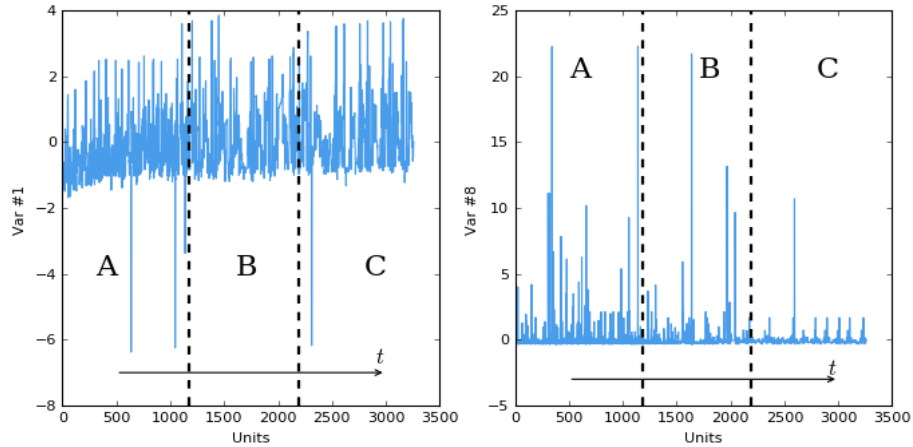


Figure 4.7: The two most important variables for the random forest grown in the case of the classification problem with three classes. Along the x-axis of the two plots, the units have been ordered according to their class belongings and the arrow indicates the internal time dependency between the classes (but there exists overlaps).

	A	B	C		A	B	C
A	0.746	0.203	0.052	A	0.574	0.221	0.204
B	0.265	0.583	0.152	B	0.378	0.302	0.320
C	0.016	0.108	0.876	C	0.308	0.23	0.462
<i>Total</i>	0.987	0.933	1.08	<i>Total</i>	1.18	0.803	0.991

(a) Random forest trained with only $Var\#1$ and $Var\#8$.(b) Random forest trained without $Var\#1$ and $Var\#8$.

Table 4.8: Confusion matrices showing the result of a 10-fold cross-validation applied on two different random forests. To generate the result in the left table, only the two most important variables, see figure 4.6, were considered to describe each unit whilst the result in the right table was generated of a random forest where the seven other variables were used to describe the units.

The results are presented via the two confusion matrices in table 4.8, with an overall precision of 0.749 for the model with the two most important features and an overall precision of 0.445 for the other model where these two variables are omitted.

Throughout this section, numerous random forests have been used to obtain the results and in all of the cases, where it has been possible, the same parameter values were used. There exists around 5-7 parameters in the random forest setup [7], but in our settings only two of these were considered to influence the result in such degree that they needed to be tuned, namely the number of trees n in the forest and the number of splitting variables considered in each split m . The tuning was done in a simple manner, where one of the parameters was kept constant whereas the other parameter was altered. For each of the parameter pairs, 50 randomly initialised random forests were fit to the data and evaluated with the OOB precision and the final precision was obtained by averaging over the forests. The resulting plots are displayed in figure 4.8 and from these the two parameter values $n = 200$ and $m = 7$ were chosen to be the most suitable ones for our application.

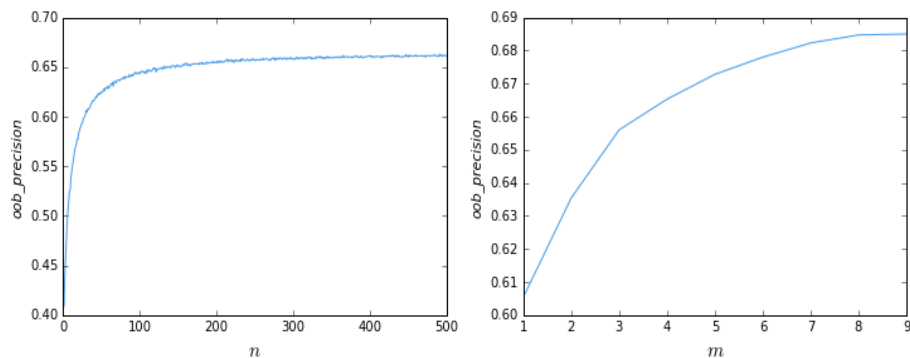


Figure 4.8: Plots displaying the OOB precision for the random forest models for different values on the parameters. The left plot show the precision for different numbers of trees in the forest n , when the number of splitting variable is kept constant, i.e $m = \sqrt{p}$, where $p = 9$ is the dimension of the feature space. In the right plot the number of trees are kept constant $n = 200$ while the number of splitting variables are increased. The precision for each value of parameters was achieved by taking the average result generated of 50 differently initiated random forests.

Chapter 5

Discussion

We will attempt in this section to walk through the project in a natural order, argue about our methodology and discuss how our results were affected accordingly. Through the discussion we implicitly provide answers to our formulation of questions and in addition we give suggestion to SoMC on related future work.

The first crucial part of this project was about extracting features and finding a relevant representation which would capture the anticipatory software version dependence. Our approach, partially inspired by previous SoMC work but certainly influenced by personal intuition, consisted of computing certain statistics for the different performance areas which altogether constituted the feature set. Even though different time frames were considered, a critical delimitation lied within the approach of letting features be based on conditional and perhaps too intricate statistics. The multiple conditions concerning features that were computed conditionally, were tweaked until a level of completeness was obtained such that a satisfactory amount of devices would pass the filtering process and be compatible for further analysis. Although being difficult to evaluate the impact of this methodological looseness, the deficiency in feature quality which become apparent in the unsupervised learning setup indicates that more work is needed in finding consistent features. Trying to capture device performance in our features, supposedly changing with software development, failed miserably according to evaluation of our attempted clustering from which we conclude that our features are inadequate. The performance area related to device temperature that was supposed to be integrated in a potential model turned out difficult to include because of vast incompleteness and inconsistency. Despite this we believe that relevant data was used given the predetermined performance areas but that our features suffer from not being general enough. Our recommendation is that if SoMC wishes to continue working on this track, focus should be put in first finding a feature representation that preserves generality.

Secondly, models using survey data as labels was studied in a supervised setting. The unsupervised learning method gave an indication of lacking separability between devices of different software versions, so any model based on supervised learning and the same assumptions about the software is expected to suffer at least slightly from the potential feature deficiency. We considered the survey with the largest population to investigate the potential of predicting the perceived quality which, if results were successful, would have enabled to build a temporal model taking multiple surveys into account which would predict future perceived quality. This idea of a prognostic model for perceived quality prediction was clearly rejected by the poor results generated in this approach and our attempt of increasing model accuracy through data augmentation did not improve the results either. We can neither confirm nor deny if an accurate model for perceived quality prediction could be built on the given data since in addition to the uncertain feature quality, we are not sure about the label quality. First of all, the population of survey attendants are not really statistically significant to draw conclusions from. However, one interesting observation that can be made from the survey responses about overall perceived quality, displayed in figure 4.4, is that the median is constant over time. So if our assumption about conformity between software development and increasing quality would be correct, that would say or indicate that users follow the Kano model since the perceived quality, being constant, is diminishing relative to the software quality which is increasing by assumption. Along with general subjectivity comprised in survey data, the effect described by Kano would also need to be taken into consideration if ever attempting on creating a supervised model based on survey data.

Turning to the last model tested, where the software versions were used to create labels, the idea was to use this model as a verification of how the software development of new platforms progresses. Assuming that our model would work perfectly and could distinguish between early and late software versions of the old platform, then the predicted software version when applied on new data would correspond to a software version of the old platform. This information could then be used as a reference for how far the platform development has come compared to the old one. To have use of such a model, a major assumption needs to be done, which is that the quality is increasing for each software version. Of course is not the case, but there will hopefully be an increase in quality if large time intervals are considered. With this in mind, we created the representations described in table 4.5c and table 4.5b, hoping to capture such behaviour. The results obtained seemed promising at first, but the results acquired from the model with only 2 out of 9 variables indicated something else. These two variables corresponded to attributes of the device that are expected to change in a certain way. In particular, the two features describe the number of installed packages on the device and the start-up time of a certain application which are expected to increase and decrease, respectively, throughout the development of the software. These results reconfirm the deficiencies in our features.

A major part of this thesis has consisted of familiarising with the different

SoMC elements such as probes, the data warehousing and the means used for data extraction. The acquainting process was very time consuming and developing sufficient understanding for how features were going to be successfully extracted required at least 2 months of work. This meant first learning about what probes are and how they log data in order to understand the database content. Afterwards, obtaining sufficient data warehousing insight was required for a successful data mining process which implied learning HiveQL for database querying. Despite having queries running around the clock, the task of fetching the data for subsequent feature extraction was heavily time consuming as well as the corresponding scripting. Luckily we were provided with explanatory scripts which accelerated the actual scripting process and helped us understand how to utilise cluster functionalities such as MapReduce. Moreover, the survey data which was received relatively late during the project also required work to integrate and the time was insufficient for considering other quality measures.

In spite of being a greatly educating process, to instead have been provided with a prepared dataset of satisfactory quality would have allowed for an extensive data analysis and implementation of more models. The task of collecting relevant data, finding appropriate representations and perhaps integrating it with different kinds of information (e.g. quality measures) from other databases could potentially by itself average the work conform to a master thesis. With the same argument, studying given data and investigating different models based on machine learning could be done thoroughly enough to meet the scope of a thesis. Hereby, we suggest for future thesis opportunities that a project of this range be divided into two parts in order to allow for exhaustive investigation of the respective domain. Likewise, having identified the lack of a sufficiently general quality measure, we think attention should keep being turned towards finding a satisfactory norm. Meanwhile we emphasise the importance of impeaching results of models based on supervised learning in which label quality may have crucial impact, such as the inevitable subjectivity characterising survey data.

For this thesis to be finished in time and still to have completed the pipeline of tasks as described in the background, major delimitations had to be done which is the primary reason not to draw precipitant conclusions about the machine learning applicability on the diagnostics of SoMC devices. Superior feature representations, being a prerequisite for successful model investigation, would enable for more qualitative analyses before which we believe that definite conclusions cannot be drawn regarding this matter.

Chapter 6

Conclusion

This thesis did not end in a model for quality prediction where the lack of a good feature representation was identified as the major issue. Furthermore, this rendered the relevant analysis of the ML applicability futile since our extracted features turned out not to contain any anticipated structures. Despite poor results generated by the proposed models, their adequacy should not be rejected before having evaluated the models using a superior feature representation. Consequently, for future work we propose that focus be turned to first finding several qualitative representations before reevaluating the potential of ML in the field of mobile device diagnostics. The difficulties in working with survey data has moreover been emphasised and applying it as label or quality measure should be done with caution.

Appendix A

Appendix

A.1 Correlation

The correlation matrix (2 digit precision) for the larger set of devices considered in 4.1

$$\Sigma = \begin{pmatrix} 1.0 & 0.19 & 0.2 & 0.21 & -0.11 & 0.05 & 0.01 & 0.05 & 0.03 & 0.04 & 0.01 \\ 0.19 & 1.0 & 0.51 & 0.46 & -0.27 & 0.35 & 0.14 & 0.04 & 0.02 & 0.27 & 0.08 \\ 0.2 & 0.51 & 1.0 & 0.82 & -0.35 & 0.33 & 0.14 & 0.09 & 0.02 & 0.27 & 0.06 \\ 0.21 & 0.46 & 0.82 & 1.0 & -0.32 & 0.31 & 0.13 & 0.05 & 0.02 & 0.16 & 0.03 \\ -0.11 & -0.27 & -0.35 & -0.32 & 1.0 & -0.15 & -0.1 & -0.12 & 0.03 & -0.09 & -0.02 \\ 0.05 & 0.35 & 0.33 & 0.31 & -0.15 & 1.0 & 0.22 & 0.02 & -0.02 & 0.21 & 0.05 \\ 0.01 & 0.14 & 0.14 & 0.13 & -0.1 & 0.22 & 1.0 & 0.01 & 0.01 & 0.03 & -0.01 \\ 0.05 & 0.04 & 0.09 & 0.05 & -0.12 & 0.02 & 0.01 & 1.0 & 0.01 & 0. & -0.01 \\ 0.03 & 0.02 & 0.02 & 0.02 & 0.03 & -0.02 & 0.01 & 0.01 & 1.0 & -0.01 & -0. \\ 0.04 & 0.27 & 0.27 & 0.16 & -0.09 & 0.21 & 0.03 & 0. & -0.01 & 1.0 & 0.07 \\ 0.01 & 0.08 & 0.06 & 0.03 & -0.02 & 0.05 & -0.01 & -0.01 & -0. & 0.07 & 1.0 \end{pmatrix} \quad (\text{A.1})$$

Scatter plot between all pairs of features

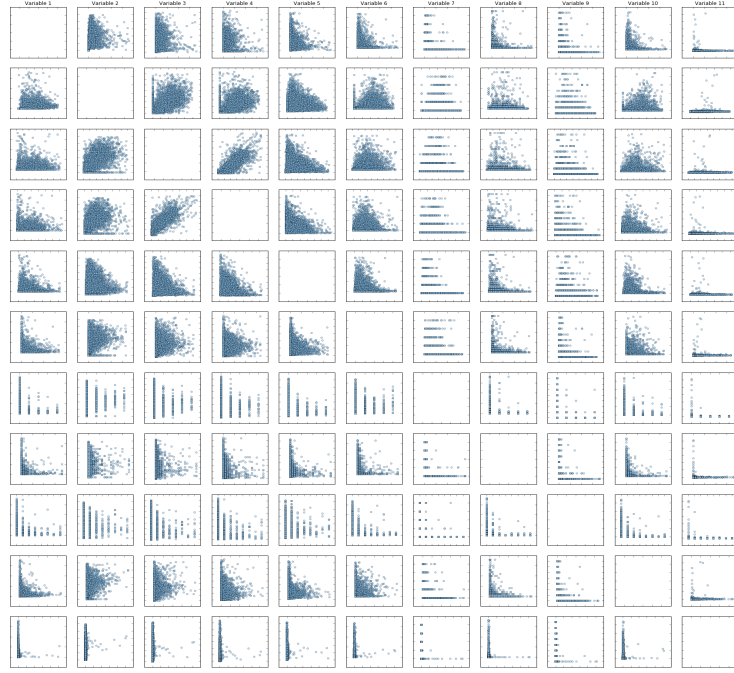


Figure A.1: Scatter plot for all combination of features from the data set.

The correlation matrix for the set of devices considered in 4.3.2

$$\Sigma = \begin{pmatrix} 1.0 & 0.37 & 0.0 & 0.18 & 0.0 & 0.02 & 0.05 & 0.03 & 0.16 \\ 0.37 & 1.0 & 0.12 & 0.28 & 0.2 & 0.22 & 0.05 & 0.01 & 0.06 \\ -0.0 & 0.12 & 1.0 & 0.17 & 0.2 & 0.11 & 0.02 & 0.01 & 0.0 \\ 0.18 & 0.28 & 0.17 & 1.0 & 0.7 & 0.14 & 0.01 & 0.02 & 0.06 \\ 0.0 & 0.2 & 0.2 & 0.7 & 1.0 & 0.12 & 0.05 & 0.02 & 0.01 \\ 0.02 & 0.22 & 0.11 & 0.14 & 0.12 & 1.0 & 0.03 & 0.01 & 0.03 \\ -0.05 & 0.05 & 0.02 & 0.01 & 0.05 & 0.03 & 1.0 & 0.01 & 0.0 \\ 0.03 & 0.01 & 0.01 & 0.02 & 0.02 & 0.01 & 0.01 & 1.0 & 0.0 \\ 0.16 & 0.06 & 0.0 & 0.06 & 0.01 & 0.03 & 0.0 & 0.0 & 1.0 \end{pmatrix} \quad (\text{A.2})$$

A.2 Additional Results

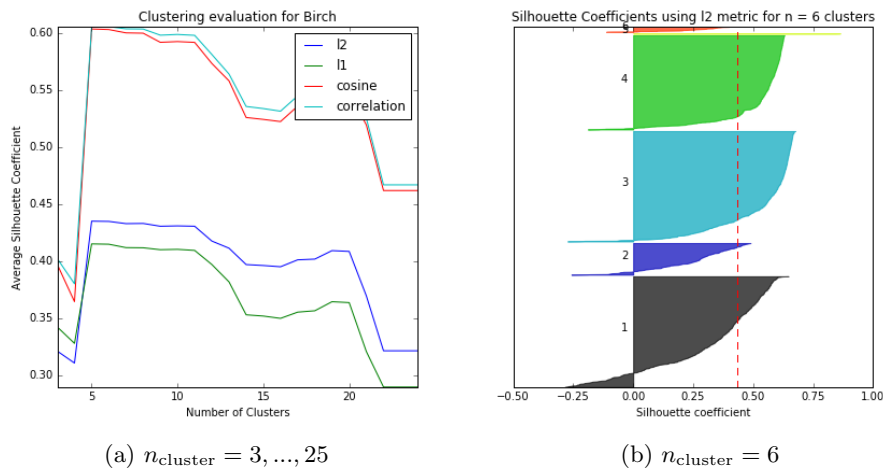


Figure A.2: Evaluation of the BIRCH clustering algorithm by calculation of the silhouette coefficients. In the left figure is the average of the silhouette coefficients plotted as a function of the number of clusters and the different colours represent different measures applied in the calculation of the silhouette coefficients. The right figure displays the silhouette coefficients obtained with $n_{\text{cluster}} = 6$, which give the highest average of the silhouette coefficients.

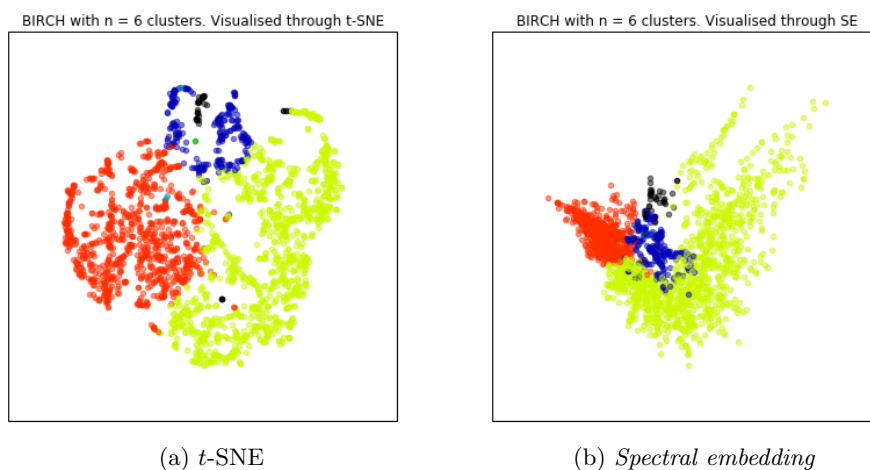


Figure A.3: The clusters computed with the BIRCH algorithm, see figure A.2b, illustrated in two dimensions with the help of dimensionality reduction techniques, *t*-SNE and spectral embedding.

	A	B	C	D	E	F	G		A	B	C	D	E	F	G
A	0	0	0	0	0	0	0	A	0	0	0	0	0	0	0
B	0	0	0	0	0	1	0	B	0	0	0	0	0	1	0
C	0	0	0	0	0	8	0	C	0	0	0	0	0	8	0
D	0	0	0	0	2	7	0	D	0	0	0	0	2	7	0
E	0	0	0	0	2	15	1	E	0	0	0	0	2	15	1
F	0	0	0	0	1	12	2	F	0	0	0	0	1	11	3
G	0	0	0	0	0	1	0	G	0	0	0	0	0	1	0

(a) Without regarding any class priors
($C_i = 5, \forall i$).

(b) With the class priors taken into account
(class specific C_i).

Table A.1: The two confusion matrices obtained from evaluating the model performance of two SVMs with parameter $\gamma = 1.3$ (obtained from grid search with SSE as objective function), using our customised N -fold CV.

Bibliography

- [1] Mikhail Belkin and Partha Niyogi. “Laplacian eigenmaps for dimensionality reduction and data representation”. In: *Neural computation* 15.6 (2003), pp. 1373–1396.
- [2] Christopher M Bishop. “Pattern Recognition”. In: *Machine Learning* (2006).
- [3] Marketing Accountability Standards Board. *Customer Satisfaction*. URL: <http://www.commonlanguage.wikispaces.net/Customer+satisfaction> (visited on 05/22/2016).
- [4] L Breiman. ““Bagging predictors” Technical Report”. In: *UC Berkeley* (1994).
- [5] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.
- [6] Leo Breiman et al. *Classification and regression trees*. CRC press, 1984.
- [7] Lars Buitinck et al. “API design for machine learning software: experiences from the scikit-learn project”. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.
- [8] Aimee L Drolet and Donald G Morrison. “Do we really need multiple-item measures in service research?” In: *Journal of service research* 3.3 (2001), pp. 196–204.
- [9] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. Springer series in statistics Springer, Berlin, 2001.
- [10] Mark Hall, Ian Witten, and Eibe Frank. “Data mining: Practical machine learning tools and techniques”. In: *Kaufmann, Burlington* (2011).
- [11] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Elsevier, 2011.
- [12] Apache Hive. *LanguageManual Joins*. URL: <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Joins>.
- [13] ibm.com. *What is MapReduce?* URL: <http://www-01.ibm.com/software/data/infosphere/hadoop/mapreduce/> (visited on 03/18/2016).

- [14] ibm.com. *What is the Hadoop Distributed File System (HDFS)?* URL: <http://www-01.ibm.com/software/data/infosphere/hadoop/hdfs/> (visited on 03/18/2016).
- [15] Noriaki Kano et al. “Miryokuteki Hinshitsu to Atarimae Hinshitsu”. In: *Attractive Quality and Must-Be Quality*). *Quality, JSQC* 14.2 (1984).
- [16] Kurt Matzler and Hans H Hinterhuber. “How to make product development projects more successful by integrating Kano’s model of customer satisfaction into quality function deployment”. In: *Technovation* 18.1 (1998), pp. 25–38.
- [17] Thomas M Mitchell. “Machine learning”. In: *Machine Learning* (1997).
- [18] Dorian Pyle. *Data preparation for data mining*. Vol. 1. Morgan Kaufmann, 1999.
- [19] Andrew Rosenberg and Julia Hirschberg. “V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure.” In: *EMNLP-CoNLL*. Vol. 7. 2007, pp. 410–420.
- [20] Peter J Rousseeuw. “Silhouettes: a graphical aid to the interpretation and validation of cluster analysis”. In: *Journal of computational and applied mathematics* 20 (1987), pp. 53–65.
- [21] Jon Shlens. *A Tutorial On Principal Component Analysis. Derivation, Discussion and Singular Value Decomposition*. 2003. URL: https://www.cs.princeton.edu/picasso/mats/PCA-Tutorial-Intuition_jp.pdf (visited on 05/23/2016).
- [22] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Data mining cluster analysis: Basic concepts and algorithms*. 2013.
- [23] Clas Thurban. Personal communication. Mar. 1, 2016.
- [24] Laurens Van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE”. In: *Journal of Machine Learning Research* 9.2579-2605 (2008), p. 85.
- [25] Joe H Ward Jr. “Hierarchical grouping to optimize an objective function”. In: *Journal of the American statistical association* 58.301 (1963), pp. 236–244.
- [26] wikipedia.org. *20 Questions*. URL: https://en.wikipedia.org/wiki/Twenty_Questions (visited on 05/16/2016).
- [27] wikipedia.org. *Apache Hadoop*. URL: https://en.wikipedia.org/wiki/Apache_Hadoop (visited on 03/18/2016).
- [28] wikipedia.org. *Apache Hive*. URL: https://en.wikipedia.org/wiki/Apache_Hive (visited on 03/18/2016).
- [29] wikipedia.org. *Bootstrapping (statistics)*. URL: [https://en.wikipedia.org/wiki/Bootstrapping_\(statistics\)](https://en.wikipedia.org/wiki/Bootstrapping_(statistics)) (visited on 05/19/2016).
- [30] wikipedia.org. *Cross-entropy*. URL: https://en.wikipedia.org/wiki/Cross_entropy (visited on 05/12/2016).

- [31] wikipedia.org. *International Mobile Station Equipment Identity*. URL: https://en.wikipedia.org/wiki/International%5C_Mobile%5C_Station%5C_Equipment%5C_Identity (visited on 05/12/2016).
- [32] wikipedia.org. *Unix time*. URL: https://en.wikipedia.org/wiki/Unix_time (visited on 03/18/2016).
- [33] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. “BIRCH: an efficient data clustering method for very large databases”. In: *ACM Sigmod Record*. Vol. 25. 2. ACM. 1996, pp. 103–114.