MASTER'S THESIS | LUND UNIVERSITY 2016

Patterns in live performance data

Simon Svensson

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2016-37

Patterns in live performance data

(Method to find explanations for the app start times)

Simon Svensson
dat11ss1@student.lu.se

August 23, 2016

Master's thesis work carried out at Sony Mobile.

Supervisors: Prof. Pierre Nugues, Pierre.Nugues@cs.lth.se
Jens Gulin, Jens.Gulin@sonymobile.com

Examiner: Jacek Malec, Jacek.Malec@cs.lth.se

Abstract

This thesis explores methods to find features that affect the start time of mobile apps. To help app developers improve performance over patch cycles, we implement an alarm pipeline in Apache Spark and tested it. This implementation is able to detect notable changes in the start time distribution and alert the developer. Spark is a scalable cluster computing framework, that proved well suited for the given problem.

The program consists of five steps; preprocessing the data, fitting a Gaussian mixture model (GMM) to the data, and finding differences in the distributions. Then a linear regression model is fit to map the available features to the GMM parametrization. The linear regression enables the developer to find relationships between the available features and the parametrization, by analyzing the regression weights.

We tested the alarm pipeline on two data sets and shown that it could identify statistically significant changes in the start-time distribution.

Keywords: Android, Linear regression, Gaussian mixture models, Clustering, Performance data, Sony Mobile

Acknowledgements

I would like to thank my supervisor Prof. Pierre Nugues for his guidance and advice throughout the thesis work.

My gratitude also goes to Jens Gulin for his relentless criticism and for sharing his invaluable knowledge about both Sony and Android, as well as inviting me to the “Cupcake” fika group.

I would like to thank Snild, Björn and the rest of the performance team at Sony Mobile, where members have helped me with everything from computer problems to feature finding.

And Maria, without her proofreading and support I would never have been able to finish this thesis.

Lastly I am gratefully that Sony Mobile allowed me to use their data and allowing me to work at their great office.

Contents

Chapter 1
Introduction

1.1 Background

Sony Mobile produces high-quality mobile phones that run the Android operating system. Sony Mobile designs the hardware, and customizes the Android operating system to better suit Sony's customers needs. Sony Mobile gathers some data with user consent from it's mobile phone users. This data is used in analysis to improve products. This thesis will focus on performance analysis. In particular the thesis focuses on finding what affects the start time of applications on Sony phones. By start-time is meant the time from when a user starts an application until the application is usable by the user.

The current mobile market is massive and growing, with a large amount of people using their phones, generating more data at all times. The users are not only using their phones to call, quite the opposite, other applications are used at least as frequently. As the demands on the phones' performance grow there is a need to monitor and improve the performance of the phones to enhance the user experience. Many users view the start time of applications as an important part of their phone's everyday experience. Therefore it is important to reduce application start times. In complex hardware and software, it is often difficult to know why something is slow. Currently, this performance data is analysed using classical statistical methods rather than machine learning. Machine learning could help bring new insights to the table and allow for a better understanding of the performance of phones.

The goal of the thesis is to use machine learning to find phones that operate particularly slow compared to similar phones and to identify what causes the observed delays; as well as finding a method for doing these analyses semi-automatically and with new data. With a growing amount of performance data, it would be beneficial to use machine learning in addition to classical statistical models.

The application start-times differ between phones and have a high variance. Many machine learning models are known to perform poorly on high-variance data. Therefore it would be useful to estimate the probability distribution of the application start-times and use the estimated distribution as an input to machine learning algorithms instead of the raw data.

In summary, the questions this thesis attempts to answer are the following:

- How to find the important features?
- How to describe distributions of start-time data?
- How to apply machine learning models on performance data?

The distribution estimation and machine learning models found are then combined into a pipeline. The found model can be used to notify or alarm the application developers of significant changes in the start-time distributions. This pipeline will be referred to as the Alarm pipeline.

1.2 Introduction to Android Applications

Android applications, called apps for short, consist of a number of components. All apps consist of one or more activities. An activity is a single screen in the user interface. Almost all apps have a main activity that starts when the user starts the app. Some apps start with a different activity than the main, for example the album app can start in show image mode directly.

All parts of an app are delivered to the user in the form of a single package. The name of the package should be descriptive and unique as this is used to differ between applications.

Some apps available to the user are delivered with the phone. On Sony phones most of these pre-installed apps are produced by Sony or Google. The user can also install additional apps to the phone.

An app that is shown on the screen is considered to run in the foreground, and an application that runs without user action is considered to run in the background. An application that is running in background can be terminated by the Android operating system if need be. For example an app running in the background can be terminated when the phone runs out of RAM.

An app can also start another app. For example the contact list app cannot make calls, so the contact list app has to start the phone app to make calls. To give the user a smooth experience it is important the start-time of the newly initialized app is short.

1.3 The Data Mining and Machine Learning Process

Cross Industry Standard Process for Data Mining, CRISP-DM, is a method of data mining and data analysis that is widely used. Data Mining is broad term, referring to various methods for exploring patterns in data. CRISP-DM is an iterative process with six steps to perform data mining (Chapman et al., 2000). Figure 1.1 show these six steps.

The six steps are shortly described below:

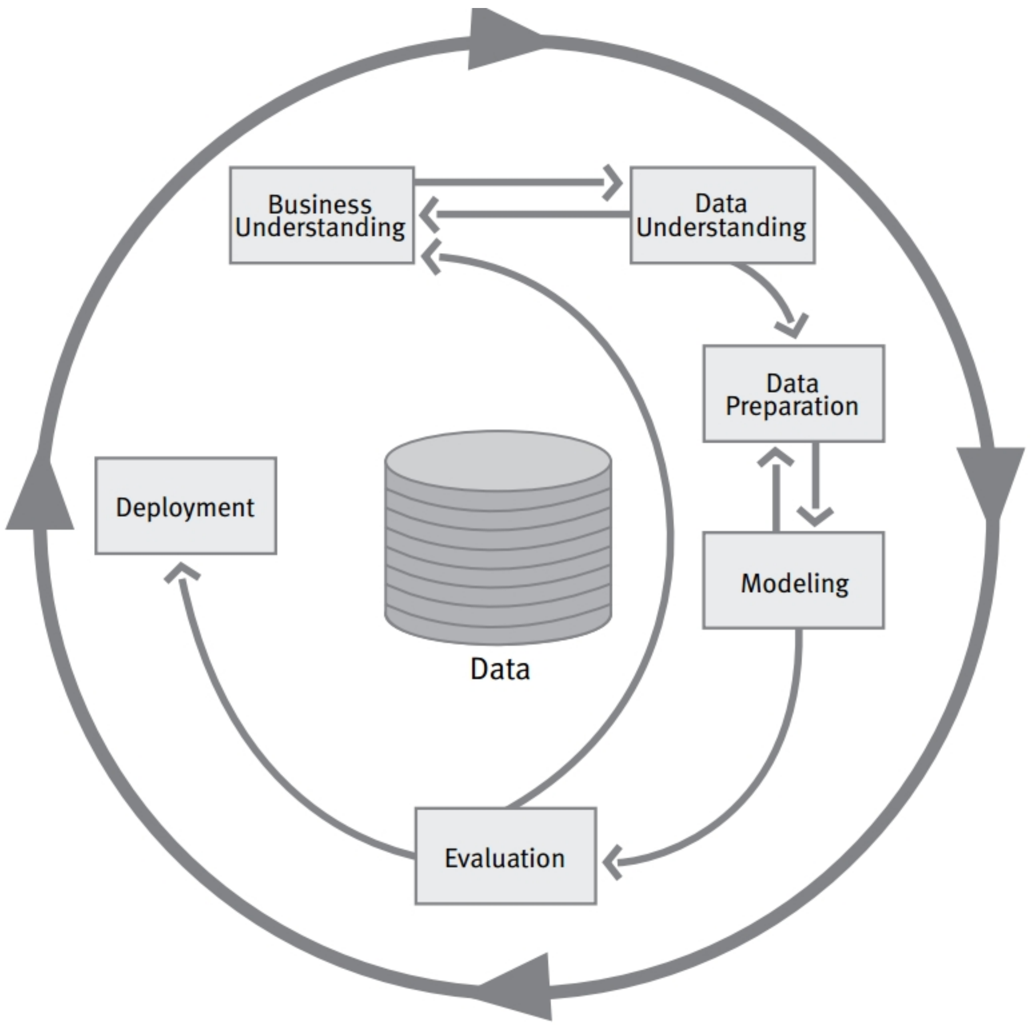


Figure 1.1: An illustration of the CRISP-DM steps. From Chapman et al. (2000)

1. **Business Understanding.** Decide what is the business benefit of the given data mining task, and how is this related to the data at hand.
2. **Data Understanding.** Understand the data structure of the data. Locate possible problems with the data.
3. **Data Preparation.** Collect and prepare the the data set for modelling.
4. **Modeling.** Model the data sets.
5. **Evaluation.** Evaluate the model.
6. **Deployment.** Apply the model to provide new knowledge of the data set for solving the initial problem.

All of these steps are important. The 1 to 5 loop is used actively during the the thesis work to improve the existing models and prepare the data for new models. It is easy to lose track of the goal of the task at hand when dealing with large amounts of data. CRISP-DM is used to perform data analysis in an organized and focused manner.

1.4 Previous Work

There has been work published on predicting the starting times of applications, based on application byte-code analysis (Kwon et al., 2013). They analysed the byte code to find slow or heavy computation as well as possible branches that could make the program run slower. They then formulated a function that predicts the running time of apps. The byte-code level analysis of Kwon et al. (2013) obtained a quite good result with only a few percent error rate in the prediction. This thesis differs from Kwon et al. (2013) since we look at the performance of various apps on different phones without analysing the source code at any level. Further Kwon et al. looked only at one phone in their analysis, this phone was a Galaxy Nexus, where as in this thesis several phones are compared.

Some similar research has been done on Windows PC machines in Bird et al. (2014). The authors look at the reliability of software with relation to other programs on the machine, and try to find what causes the software to function incorrectly. In Bird et al. (2014) the combination of programs is emphasised as well as the performance of the PC. The information used in the analysis consist of which apps the user has installed, hardware specifications of the computer, information about program usage such as number of starts and the number of crashes. In Bird et al. (2014) they have features describing the hardware, it should be noted that the hardware of PC:s can vary greatly. In this thesis we concentrate on Sony phones, therefore exploring hardware with some variation but less than the variation of the hardware studied in Bird et al. (2014).

In Bird et al. (2014) they found that certain applications affect other applications in a significant way. For example if the user plays games it seems to affect all the other programs and reduce their reliability. It is interesting that applications can affect each other and this study could be extended to Android apps as well.

Chapter 2
The Data

This chapter describes the data and features used in the thesis. The simpler preprocessing steps applied to the data are also discussed.

2.1 Overview

Sony gathers, with user consent, some information from phones used by Sony Mobile customers. Sony gathers additional data from test phones. The test phones are tested internally at Sony by automated test rigs, test scripts and by Sony employees.

The test rigs are run by robots that use the phone's screen with a touchscreen pen and record the apps' start times. The robot tests are made to simulate a user that uses the same apps several times. The tests do not imitate exactly a human's usage of a phone but catch some crashes. There are also test scripts that run similar tests. The test scripts are run on software only and thus lack some of the errors that result from touchscreen usage.

Sony employees use test phones as their daily phones. Their usage should resemble a typical user, but there are some problems with this test data. First, the sample group is not entirely representative of the customer population. For example, most usage data is collected in the countries where Sony Mobile has development sites, with additional bias representing employee gender and age amongst developers. Secondly the employees are often technically inclined and if they experience a crash they often try to replicate it, resulting in an unusually high number of crashes.

Sony also has a retail user base from where some users send in data to Sony. To ensure the users' privacy Sony limits the amounts of data gathered from private users to a minimum. Due to this data that is collected from retail phones is more limited compared to the data collected from internal phones. The external data is also anonymised when it is sent to Sony such that that two samples from the same phone can be identified to originate from the same device, but not which phone the samples come from.

This thesis focuses on Sony apps.

Table 2.1: Comparison between Z5 phones

Phone name	**RAM (GB)**	**Display**	**Battery (mAh)**	**Size (mm)**	**Abbr.**
Xperia Z5	3	5.2" FHD	2900	146 x 72 x 7.3	Z5
Xperia Z5 Compact	2	4.6" HD	2700	127 x 65 x 8.9	Z5-C
Xperia Z5 Premium	3	5.5" 4K	3430	154.4 x 75.8 x 7.8	Z5-P

The phones gather a set of log messages from the start of apps. When the phones have collected enough data, it is sent to Sony's servers. After arriving at Sony some processing is done to pick out the interesting parts of the log message. The experiments in this thesis are done on a subset of this data.

2.2 About the Data Set

This thesis concentrates on the data from phones in the Z5 series. In this series there are three phones, Xperia Z5, Xperia Z5 Compact and Xperia Z5 Premium. Table 2.1 shows the different models. Note that CPU is not in the table since they all share the same CPU type, Quad-core 1.5 GHz Cortex-A53 and Quad-core 2.0 GHz Cortex-A57. The phones also have the same version of Android 5.1 natively, but have an optional upgrade to Android 6.0 or later available on most markets (Devenish, 2015).

In Table 2.1 the reader can see that the Z5-P phone is the largest, has the best screen and battery. The normal Z5 has medium values and the Z5-C is a bit smaller. A larger screen contains more pixels than a smaller one and therefore can require more CPU power to draw an image, and may thus affect the phone's performance. Since battery power is rather limited it affects the CPU speed. With more battery available the phone's CPU usage could be increased thus resulting in increased performance. Heat is also a factor that is taken into consideration when deciding on CPU usage. A large phone has a larger surface area to give off heat from and ventilate. Therefore a larger phone can use its CPU more effectively since it is able to ventilate more of the heat.

For the initial experiments, a small partition of the data was used. The algorithms were tested on a personal computer, with Intel® Xeon(R) CPU E5-1650 0 @ 3.20GHz × 12 and 15.6 GB of RAM. The data set has approximately one million entries. Entries with null-valued features are not included in the data set. The data gathered varies between phone models, therefore only phone models of the Z5 series are studied. This series was chosen because it is one of the newer mobile phone series and provides a large number of features in the captured data. The data is from internal testing, both from the employee group and automatic testing with robots and scripts.

2.3 Data Storage

The data is sent to Sony as log files that are opened and interpreted as they arrive to Sony.

The data is stored on a Hive Cluster or a Relational Database. The cluster storage contains more raw data while the relational database is cleaner. The relational database

Table 2.2: Features used

Feature	Unit
Start time	in ms
Up-Time	in ms
Start Type	0 or 1
Android version	String
Package	String
Encrypted	Boolean
Frequency limit	String
Model	String
Group	String

also contains less data and has a few days lag time, from when data enters the cluster storage until the data is stored in the database. This thesis uses the data from the relational database but the program could be extended to support the hive data. The only difference would be in the first preprocessing method of the program.

To have a stable test set and an easy format to work with, a tab separated text file was generated from the database.

2.4 Features

When given user consent, each time an app is launched Sony collects non-personal data describing the event. In this section follows a description of some of the data fields that are recorded in the database. For clarity they are split into two sub-categories; software and hardware features. Table 2.2 summarisers the features.

2.4.1 Software Features

Some of the features collected are created by the software and are related to the software that is running.

Start Time

The app start time that is recorded is called "Fully-drawn time". It is the time from when the application starts until a log message is sent from the app that it is ready for being used. Fully-draw time is what we use as the start-time in this thesis.

There is also a time recorded that is called "Activity Time". It is the time from when the application starts until it is done launching in the android operating system. This time exists natively in Android.

All times are recorded as an integer in milliseconds.

Up-Time

Sony collects data about how long the phone has been powered on. This is called up-time. Up-time could affect the start time, due to possible issues such as memory leaks and similar issues that could arise after a long up-time. Android defines up-time as the time that the phone has been on and not in sleep mode continuously since the last shut down or sleep mode deactivation. This feature could be used to find if there is a dependency between the frequency of restarts of the phone and the performance of the phone.

Up-time is recorded as an integer in millisecond.

Start Type

Applications starts in Android can be classified as warm and cold starts. A cold start is when the app has not been started recently and is not active in the memory of the phone. A warm start is when an app that has been used recently and is still active in the memory is started.

This information is logged when available. Complications with logging this information have been noted. A sample that has a logged "cold start" could in practice have been incorrectly logged and in fact have been either cold or a warm start. However a sample that has a logged state "warm start" is guaranteed to have had a warm start.

This feature is saved as a integer, 0 or 1 where 0 is a cold start and 1 is a warm start.

Android version

Sony collects information about the software version of the android operating system. This is not only the Android version that the phone is running but also which Sony release is used. Sony release is necessary because Sony may over time have released several patches for an Android version, also there may exist separate releases for different operators or for different markets, e.g. to handle network specific requirements.

Package

The package of the app is recorded. The package is used as a general description to find which application was launched. Several activities might share the same package but these are usually a part of the same application. For example the photo camera app and the video camera app share package but have different activities.

This package is saved as a string in the style of “com.example.app”, in the presented results this is written without the “com.example” part and becomes “app”.

Encrypted

Android allows encryption of the phone’s file system to prevent theft of personal data from the phone. During use the data has to be decrypted. The constant encryption and decryption of data could possibly lead to lower performance. This feature is a Boolean called “Encrypted”, that is set true if encrypted.

2.4.2 Hardware Features

Some of the features collected are hardware related, such features could be about the phone model or the usage class, on internal phones.

Frequency limit

Some of the phone models, such as the Z5 series, have a software feature that limits CPU power to make sure that the phone does not overheat. Limited CPU power should have a negative impact on the performance of the phone since it is artificially making the CPU run slower. This is a fairly new method for reducing overheating and is not present in older phone models. In this thesis the frequency limit is recorded as a string of the phone's limit on the CPU during launch of the apps. This feature will be referred to as the "Frequency limit". The older phone models that lack this feature report the "Frequency limit" as the hardware specific maximum CPU frequency.

Sony's Z5 phones have each two CPUs. The CPUs have four cores. One of the CPUs has a higher performance and power usage than the other. For example Sony's Xperia Z5 has two CPUs one with 4 cores at 2 GHz and one with 4 cores at 1.5 GHz. The slower CPU in general runs background tasks, while the faster is often used for foreground tasks and applications. The two different CPUs can be limited in different ways. The frequency limit per core is recorded.

Model

Sony collects information about the phone model. This is recorded in the database as the features "Product" and "Name". It is saved as a string consisting of the phone's name, for example Xperia Z5 is called "XperiaZ5" in the database. The phone model could have a large impact on the performance since different phones can behave very differently.

See Table 2.1 for abbreviations of phones, that are used in this thesis to make tables more readable.

Group

For each sample the user group of the given phone is recorded. The feature that records the user group is referred to as "Group". The feature records if the phone is used in automated tests or by human users. This feature is only recorded on internal phones. This feature contains some uncertainty if the phone is recorded to be a part of the automated test group. Phones used by Sony employees can have the same group as phones used in test rigs. This is due to unintentional errors in recordings of the features. This feature is recorded as a string with the name of the group, and requires additional internal tables to interpret the content of.

2.5 Preprocessing the Data

Many of the features are categorical and need to be preprocessed to be used in numerical algorithms. This is done by encoding the categorical features using one-hot mapping. It is describe below in the Section 3.2.1.

Some cleaning of the data is performed by removing outliers in start-time. To make start-times comparable between samples, there is a need to remove samples with long start-times. For example a start time of 2 weeks is unrealistic for any app. Therefore there has to be some kind of error that occurred in the recording of the app's start-time. Samples with a start-time longer than 10 seconds are removed. According to experts at Sony 10 seconds is a reasonable limit for an app's start-time on Sony devices. A start time that is longer than 10 seconds is probably an error. All zero times are also removed, because they are obviously erroneous given the ms-precision used.

As is good practise the data is scale between 0 and 1. This is done using this formula,

$$x = \frac{x_{org} - x_{min}}{x_{max} - x_{min}}. \tag{2.1}$$

Where x is the normalized feature value, x_org is the unnormalized value of the feature, x_{min} is the minimum value of the given feature that occurs in the dataset and x_max is the maximal of the given feature in the dataset. The values $x, x_{org}, x_{min}, x_{max}$ are all scalar. Normalization is performed feature-wise.

Scaling is done before applying any algorithms. This increases the numerical accuracy of the algorithms.

Chapter 3
Approach

An important part of finding out what makes a phone slow is discovering the features that contribute the most to making the phone slow. Several different methods to find features with the most effect on start-time were tested. In this chapter the tested methods will be described.

The Alarm pipeline will also be introduced. The Alarm pipeline allows the user to find changes in the performance of a given app's start-time and helps identify the possible cause of the change.

3.1 Method

Here the used methods will be introduced.

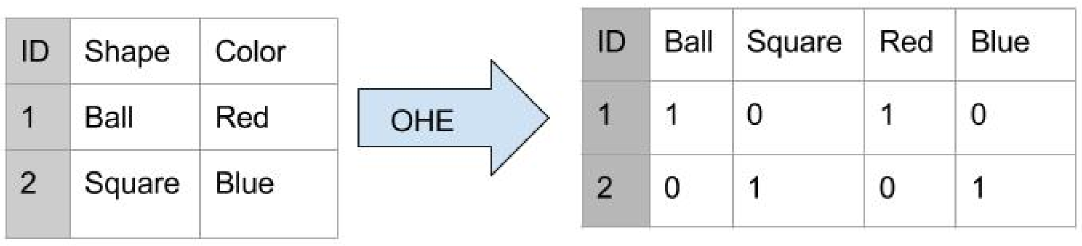
3.1.1 Identifying Relevant Features

The first sub-problem of the thesis is finding which features are the most important in deciding the performance of the phone. To find the important features, several different methods were used.

Some of the collected features in the database have little relevance to performance and were not studied. To identify features that could be of interest and likely to be relevant to the performance of the phones, experts at Sony were consulted before performing any computational work. The features described in Section 2.4 were pointed out by the experts as possibly relevant to app-start times.

3.1.2 Clustering

Clustering was used to identify groups of fast and slow start-times. These cluster centers were then analysed to study which features had an effect on the performance of the phone.



ID	Shape	Color
1	Ball	Red
2	Square	Blue

ID	Ball	Square	Red	Blue
1	1	0	1	0
2	0	1	0	1

Figure 3.1: An illustration of one hot encoding, with the table before on the left and the table after on right

Clustering was done using K-means. DB-scan was also tested but is not further mentioned in the report due to its complexity. DB-scan has a number of hyper-parameters that needed to be tuned, and since tuning is time consuming K-means was preferred.

3.1.3 Linear Regression

Linear regression was among the first methods applied after the first clustering experiments. The linear regression models built evolved into the Alarm. The amount of data and its variance made it difficult to estimate the behaviour of the whole data set given the available features, so a different approach had to be used.

3.2 Theory

This section will introduce the algorithms used as well as the concepts, parameters and notation used later in the thesis.

3.2.1 One-Hot Encoding

One-hot encoding is an encoding method used to transform categorical data into numeric data. This method is also known as dummy features or dummy encoding. The encoding works by extending the feature space with a new feature per categorical data-value. The newly introduced feature is then set to one if the observation contains this feature-value and to zero if it does not. See Figure 3.1 for example. One-hot encoding increases the number of features of the data set dramatically, which make it difficult to visualise the model without the use of some dimensional reduction method. In numerical methods the one-hot-encoded features can result in ill-conditioned feature matrices. When calculating with ill-conditioned matrices one stands at risk of increasing numerical errors. (James et al., 2013)

3.2.2 Kernel Density Estimator

The Kernel density estimator (KDE) is a method for creating a continuous histogram-like representation of the data. A Kernel density estimator can be used as a non-parametric probability distribution estimator. First a kernel function, often a Gaussian, and a bandwidth of the kernel, often variance of the Gaussian, are chosen. The KDE is the sum of kernels centered at each data point. This is seen in the equation below,

$$p(x) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{h} K\left(\frac{x_i - x}{h}\right), \tag{3.1}$$

where m is the number of data points, h is the smoothing parameter bandwidth and x is a data-point. Note that K is the kernel function. KDE describes the distribution of the sampled data as a continuous distribution, therefore likely resembling the continuous data's distribution better than a histogram does. (Bishop, 2006)

An example of this KDE is show in Figure 3.3.

3.2.3 Gaussian Mixture Model

Gaussian mixture model (GMM) is a probability distribution that consists of multiple weighted Gaussians. It allows multiple-peaked data distributions to be described with minimal number of parameters. The GMM of K Gaussians is described by the following equations,

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \sigma_k), \tag{3.2}$$

$$\mathcal{N}(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}. \tag{3.3}$$

Note that $0 \leq \pi_k \leq 1$, where π_k is the weight of the kth Gaussian, and that $\sum_{k=1}^{K} \pi_k = 1$ need to be fulfilled in order for the GMM to be valid. Further μ_k is the mean and σ_k is the standard deviation of the kth Gaussian.

The model is fitted to the data using the Expectation Maximisation (EM) algorithm. EM is an iterative algorithm that finds maximum likelihood parameters. It consisting of two steps; the expectation step and the maximization step. In the expectation step the expected conditional likelihood of the data points given the current Gaussian distribution are calculated. In the maximization step the expected conditional likelihood is maximized to find the best parameter values. The GMM fits best if the data is known to originate from Gaussian distributions (Bishop, 2006).

The GMM can be seen as a clustering model that gives the probability that a data point is part of a Gaussian cluster.

3.2.4 K-Means

K-means is a clustering algorithm where the mean of a selection of points are determined to be the cluster center. First the centers are assigned randomly in the data space. Then for each point the distances to all cluster centers are calculated and the point is assigned to the cluster with the closest cluster center. A point x is assigned to cluster μ_k, such that

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2 \tag{3.4}$$

$$\text{if k=argmin}_j ||x_n - \mu_j||^2 \text{ then } r_{nk} = 1 \tag{3.5}$$

$$\text{else } r_{nk} = 0. \tag{3.6}$$

Where N is the number of data points, K is the number of clusters and μ_k is the center of cluster $k = 1, ..., K$. The goal of K-means is to minimize the value of the loss function J . Each cluster center is then updated to be the mean of the points that belong to the cluster. These steps are repeated as long as the cluster centers change significantly between iterations.

In this method the user has to specify the number of clusters when initializing the algorithm. It is a non-trivial problem to decide how many clusters are sufficient. To decide upon the optimal number of clusters one can perform clustering with a varied number of clusters and plot the loss function J against the number of clusters, or study the within-cluster variance. With these two methods one can find the least number of clusters that still yield a sufficient loss value (Bishop, 2006).

3.2.5 Linear Regression

Linear regression is a model. It is modelled by fitting a linear hyperplane through the data points, to give a prediction of the label given the input-features vector. The weights of the model can give information about the importance of different features (James et al., 2013).

A linear regression model can be expressed in the following manner,

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n, \tag{3.7}$$

where y is the scalar output called the dependent variable, $\beta_i, i = 0, ..n$ are the weights and $x_i, i = 0, .., n$ are the input features. In matrix notation equation 3.7 can be expressed as,

$$y = \boldsymbol{x}\boldsymbol{\beta}. \tag{3.8}$$

Generally $\boldsymbol{x}$ is not a square matrix. The equation 3.8 is multiplied by $\boldsymbol{x}^T$ to solve for $\boldsymbol{\beta}$,

$$\boldsymbol{x}^T \boldsymbol{x}\boldsymbol{\beta} = \boldsymbol{x}^T y, \tag{3.9}$$

$$\boldsymbol{\beta} = (\boldsymbol{x}^T \boldsymbol{x})^{-1} \boldsymbol{x}^T y. \tag{3.10}$$

In practice the weights $\boldsymbol{\beta}$ are solved for iteratively by minimizing the sum of squared error between the true and estimated labels. The analytical solution as shown in equation 3.10 is prone to numerical errors and memory issues for large datasets.

Since the sum of squared error is a convex function, stochastic gradient descent (SGD) is used to find the minimum. SGD is an algorithm where at each iteration a small step is taken in the direction in which the gradient of the loss function is negative. The data is shuffled between iterations. This prevents the GD to get stuck in local minimum and allows for faster convergence. This iterative method is faster than taking the inverse of the feature covariance matrix and provides a more numerically stable solution; especially since matrices with One-hot encoding often have bad numerical properties which make the inverses hard to calculate numerically correctly.(Bottou and Bousquet, 2008)

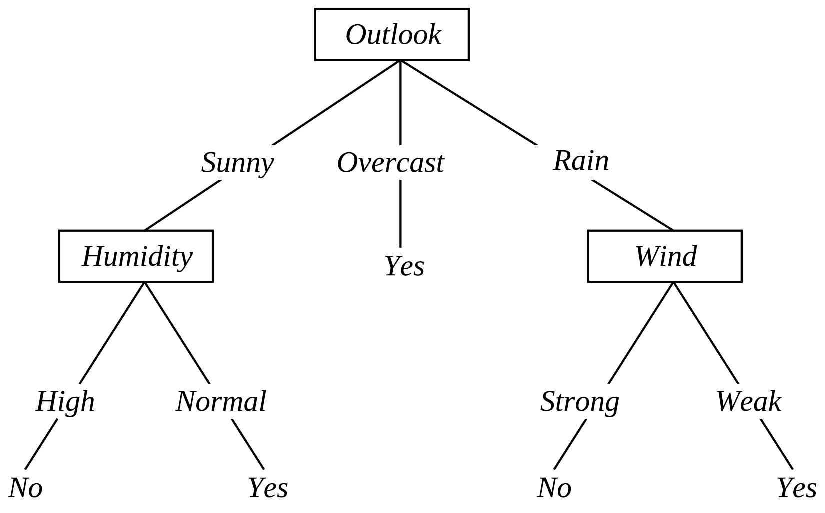


Figure 3.2: An illustration of a decision tree of the famous tennis data set. From Quinlan (1986)

3.2.6 Decision Tree

Decision tree is a model that uses a tree to describe the decisions needed to classify data. Intuitively it is very easy to understand with the help a picture. For example the tree in Figure 3.2 is a decision tree that decides if the weather conditions are suitable to play tennis or not. This example is from Quinlan (1986).

Decision trees are trained from previously labelled data. The tree is built from the root, and every branch discriminates the data with respect to an expected feature-value. The algorithm for training decision trees consist of a loop of calculating the next best feature to split the data set in two parts. This splitting can be done in different ways, two of the most common measures used to decide weather to split a leaf or not are the information gain and Gini impurity. The goal of the training is to find a tree structure such that the leaf nodes classify data correctly (Mingers, 1989).

3.3 Visualisation of the Result

The analysed data has a high number of dimensions and the data cannot be plotted in two dimensions without preprocessing. Therefore different methods for plotting the results have been used. We used two methods to visualise the results from kernel density estimation and linear regression:

- Kernel density estimation results in histogram-like plots, where the height of the curve indicates the probability of data points with this value. A transition from histogram to kernel density estimation is shown in Figure 3.3

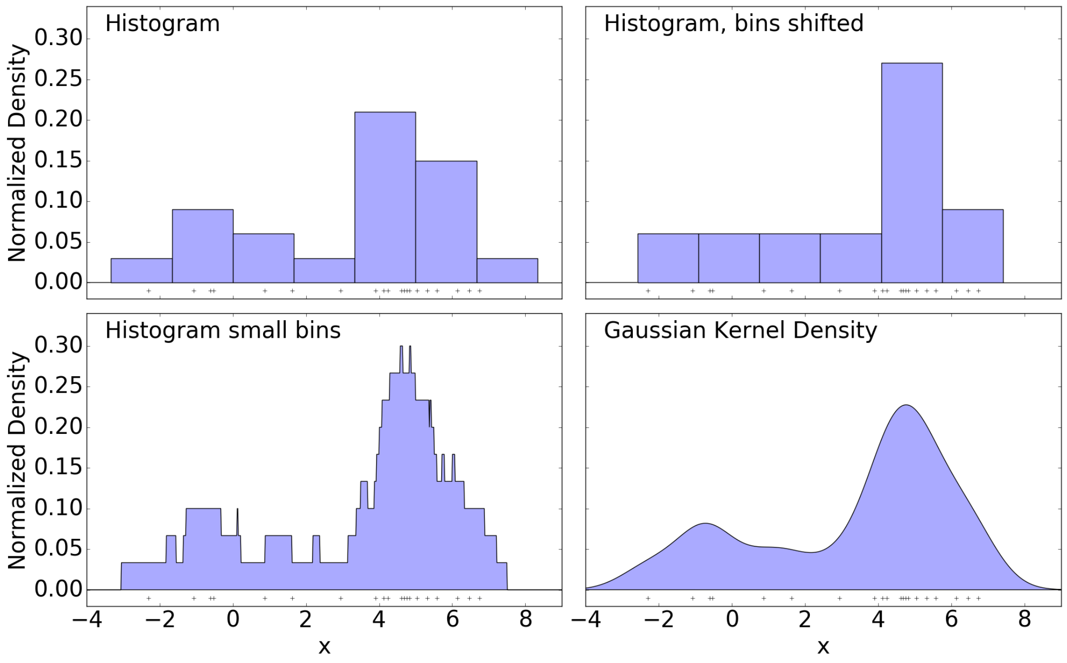


Figure 3.3: A plot of the transition from histogram to kernel density estimation, generated using Scikit-learn

- We visualise linear regression using a residual plot. A residual plot shows the observed value of the data against the predicted value. Optimally the predicted and estimated values should be equal. We add a line to the residual plot to show the optimal model's residual. An example of this is show in Figure 3.4, where the line has the formula $x = y$ and the red dots are the data-points.

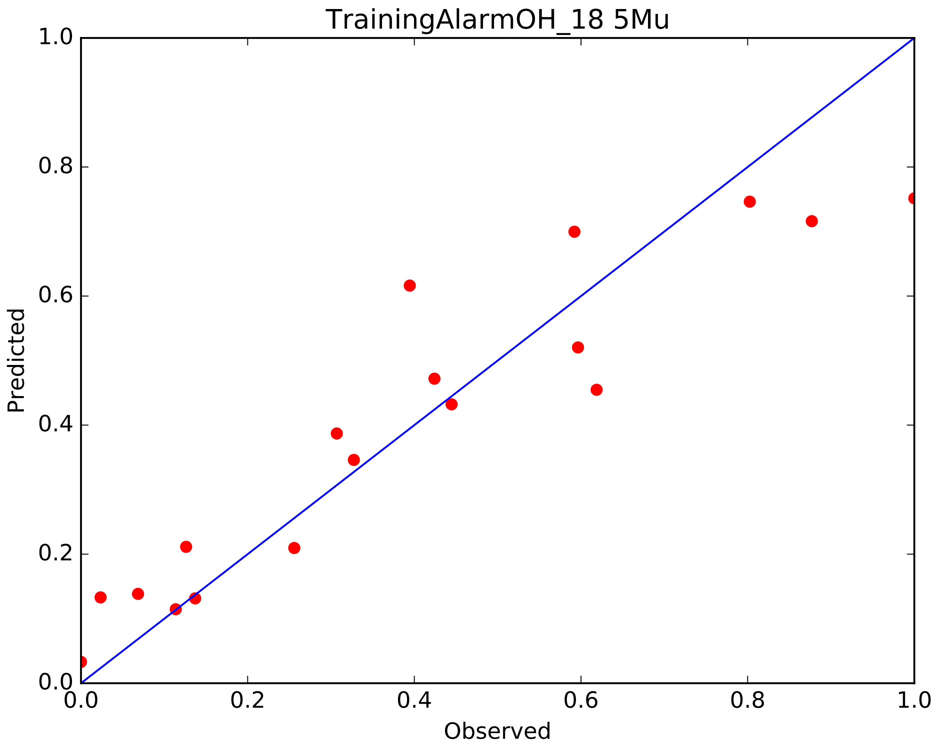


Figure 3.4: A residual plot of the linear regression, this is a figure from the linear regression tests presented in 5.3.2, this figure is of μ_5. The blue line is the line $x = y$ and the red dots are the data-points

A dimension reduction method such as the principal component analysis (PCA) could be used to plot only significant features. However during testing it was noted that plotting data in the most significant principal components was not more informative than KDE and residual plots. Therefore this method was omitted.

3.4 CRISP-DM Revisited

In the introduction a method for data mining called CRISP-DM is described. This section will describe how CRISP-DM was used throughout the thesis.

This is a connection to show what work was done during the different steps show in Figure 3.5 and described in Section 1.3

1. **Business Understanding.** To decide upon relevant questions to study regarding start times, an open-ended discussion was held with Sony employees before the start of the thesis. It was decided with the help of experts at Sony that a partially automated system for identifying reasons for delays in app launches was of interest. App start times reflect user's opinion of the quality of the phone as discussed in Section 1.1.
2. **Data Understanding.** Understanding of how the data is structured. Sony employees were of great help when figuring out how to get the data and also what the data was.

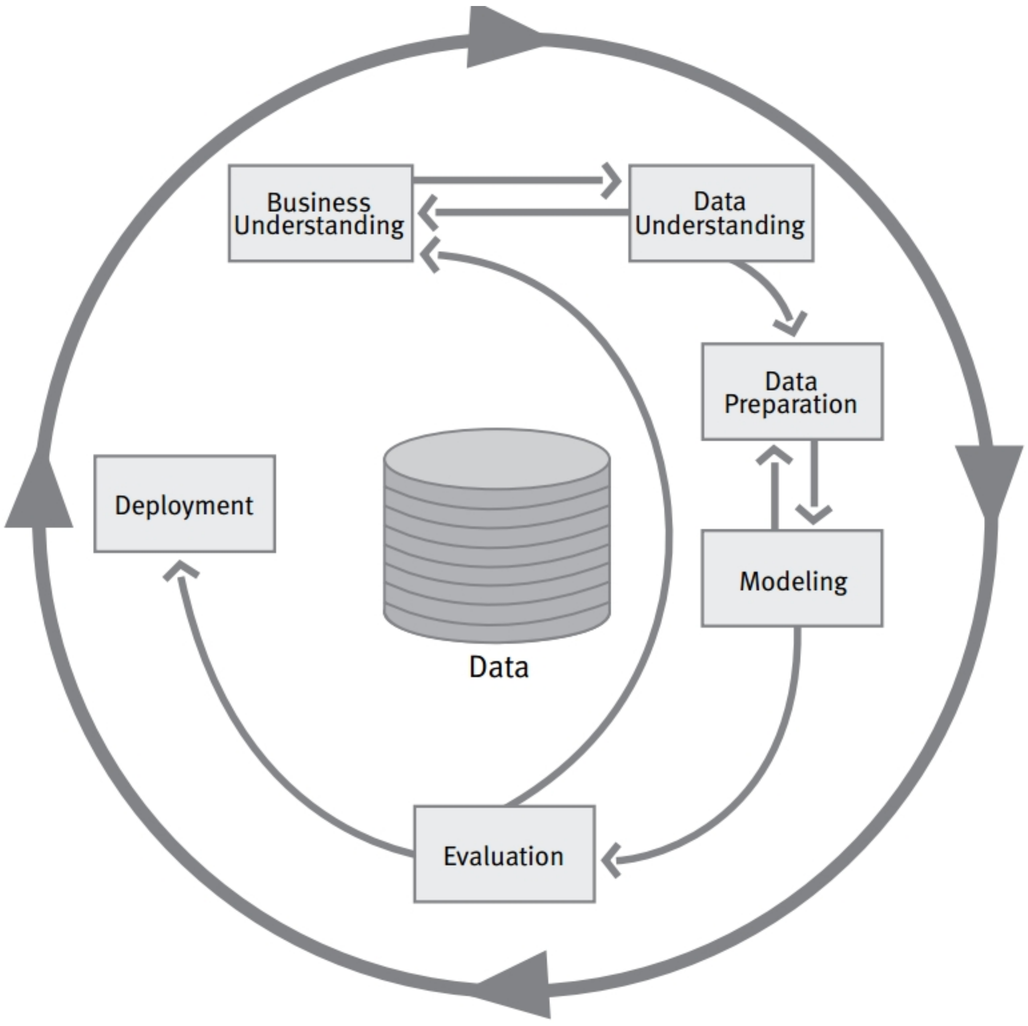


Figure 3.5: An illustration of the CRISP-DM steps. From Chapman et al. (2000)

3. **Data Preparation.** This is the process of creating the data set. Since the dataset was already gathered by Sony, this step was mostly the preprocessing of the data to fit the models described in Section 3.1.
4. **Modeling.** Find a good model for the data via experiments. This is the step of fitting a distribution to the data.
5. **Evaluation.** Evaluating the model with numerical methods like mean square error if it makes sense in the context of the data.
6. **Deployment.** This step consist of using the “Alarm” program to help the debugging.

Throughout the thesis the 1-5 loop was done several times testing different models and gathering data. The loop of 3-4 was done several times for every model with preparing the data in different ways to ensure maximum performance of the model.

CRISP-DM is a very loose model that works because it is simple and easy to use, while still giving some structure to the work.

3.5 Introduction of the Alarm Pipeline

In this section we introduce the Alarm pipeline to find the changes in the app-launch times and possible causes for the change. The Alarm pipeline is intended to be used to compare two different data-sets. After the experiments with linear regression a pipeline was created. It will be more thoroughly describe below but these are the basic steps of the pipeline;

1. Preprocess.
2. Fit a distribution to the data.
3. Find differences in test- and training- distribution.
4. Fit a linear regression model.
5. Manually analyse the regression weights.

The last step is not done automatically in the pipeline, but the weights are readily available in the output.

3.5.1 Preprocessing

Preprocessing is the first step in the Alarm pipeline. It consist of making the data ready for the rest of the steps in the pipeline. During preprocessing outliers are removed from the data. Outliers are data-points which have a start-time longer than 10 seconds. This is chosen as it is suitable limit for phones. Data point with a start time that is slower than 10s are not uninteresting but they require a different analysis. The data is collected into different groups where all features that are evaluated have the same value. The groups that have a very low amount of samples are removed. One hot encoding is performed.

3.5.2 Fitting a Distribution

In this step a GMM is fitted to the data. This is done for every group of features that is available in the data set and that is not removed by the pre-processing step. The number of Gaussians is decided by linear search to find the first local minimum.

3.5.3 Differences

In this step we look at the distributions that were created in the previous steps and find if there has is large difference in the distributions. The distributions are analysed with the help of decision trees.

3.5.4 Linear Regression

Linear regression is performed on the data generated from step two. The mean and standard deviation of the distributions are used as the dependent variable. If the number of Gaussians in the best fitting GMM model is greater than 1, multiple regressions are performed (two regressions per Gaussian). In this manner the weights from multiple regression models can be compared separately in a simple fashion.

3.5.5 Finding the Cause of the Change

Given a data point group the weights in the test and training regression can be compared to find the change in the regression weights. This can currently be done by simply looking at the weights as there are at most six Gaussians. In the future a more suitable method can be added to automate this step.

Chapter 4
Implementation

This chapter will explain the implementation details of the experiments and the Alarm pipeline. To begin with the programming language used will be described followed by a description of the implementation of the different algorithms and methods used.

4.1 Spark

We used Spark 1.6.0 to implement the program. Spark is a scalable open source cluster computing framework. It is a popular extension of Hadoop (Zaharia et al., 2012). It even has some native support for machine learning.

Spark has a library for Linear Regression with stochastic gradient descent which is parallelizable and works easily with PySpark. Spark also has support for performing kernel density estimation, applying K-means and Expectation Maximization for fitting Gaussian Mixture Models.

Further NumPy (Van Der Walt et al., 2011) for python and Scikit-learn (Pedregosa et al., 2011) for python were used in the implementations. NumPy is a python library which implements matrix operation and other practical mathematical operations. Scikit-learn is a machine learning toolbox with support for a large amount of algorithms and machine learning tools.

A tool called Jupyter was used for code development. Jupyter allows the programmer to execute small code blocks instead of running the whole program at once. This allows for a fluid implementation process.

Spark can be run in python with the help of the program PySpark.

Even if Spark is a great tool it misses some of the algorithms that are available in Scikit-learn. Scikit-learn is also better documented than Spark.

As Spark is a cluster computing frame work it uses nodes to do the calculations. When running Spark on a PC with limited memory it is important to manage the nodes of Spark in a good manner. Too few nodes can result in a low amount of parallelization but a too high number of nodes could result in out of memory errors or a too expensive overhead.

The suitable number of nodes and the memory requirement of each node may vary between programs. These variables need to be tuned by hand. This was very relevant when doing linear regression with large amounts of data without the tuning the program would run out of memory.

The computer where the tests were executed has a Intel® Xeon(R) CPU E5-1650 @ 3.20GHz × 12 and 15.6 GB of RAM. To run the programs two set-ups were used, one with two nodes with 5 GB each for memory demanding calculations and another set-up with 12 nodes with 1 GB each when less memory was required. These set-ups were sufficient for this work but could probably be improved.

4.2 K-means

The Spark 1.6 implementation of K-means was used. The implementation assigns the first cluster centers at random points in space. There is an option to specify the starting positions, but this was not used. We choose number of clusters to fit with the elbow method. One can note that the execution time increases significantly as the number of clusters increases. (Zaharia et al., 2012)

4.3 Kernel Density Estimator

We use Kernel Density estimator to estimate the true distribution of the start-time data. Spark has an implementation of Kernel Density estimator that was used. The kernel function in the Spark implementation is a Gaussian centered at the data point. Here the size of *mu* has to be picked, during the thesis this was chosen to be 250 ms, since this made the curve smooth without flattening out the curve unnecessarily much.

4.4 Gaussian Mixture Models

Expectation Maximization for Gaussian mixture models (GMM) is implemented in Spark. The Spark implementation was used in the experiments. To find the right number of Gaussians to use, a local minimum was searched for beginning with one Gaussian, and increasing the number of Gaussians in the mixture model until the error did not decrease. A good match is one that is close to the Kernel Density estimation that had been done on the data set. One Gaussian is used in the beginning because in this manner the simplest locally-optimal model is fitted.

4.5 Linear Regression

Linear regression is implemented in Spark using Stochastic gradient descent. Fitting linear regression with Stochastic gradient descent has better properties than fitting regression with matrices. Calculating the inverse on ill-condition matrices causes numerical errors and large matrices can have problems with fitting in memory. Taking the inverse of the matrix is a very time demanding task. The correlation matrix is very sparse due to the use of one-hot encoding. This further causes the matrix also to be ill-conditioned. Therefore calculating the numerical inverse of the matrix is error prone with large numerical errors.

The Spark implementation of linear regression has several hyper-parameters available. Notable is that the max numbers of iteration can be chosen, the default is 100.

4.6 Decision tree

Decision trees can be learnt with a number of algorithms. One of the most well-known methods is ID3 (Quinlan, 1986). ID3 uses entropy and information gain to make decisions when building the tree. Since these measures are adapted for categorical data, an improved version of the algorithm, C4.5, is preferred. (Quinlan, 2014) The decision tree implementation that was used, from scikit-learn, is based on the CART algorithm (Pedregosa et al., 2011). The CART algorithm is specifically adapted to build decision trees of numerical data. Since a large amount of the features are categorical it is likely that the C4.5 method would give shallower trees since there would be no need for one-hot-mapped features. As the experiments with decision trees were small this hypothesis was not explored.

4.7 Alarm Pipeline

The Alarm program consist of two different programs. One is the preprocessing part of the pipeline and the other program is the alarm part of the program. The two steps are carried out in two separate programs to allow for further analysis on the data after the preprocessing. The split in programs also helps to optimise Spark configurations for each separate part of the program.

Both parts of the program are implemented using PySpark.

Chapter 5

Evaluation

Here follows a description of the experimental setup, a presentation of the results and a discussion of these results. The features that are used in the experiments are also discussed.

5.1 Feature selection

It is known that uninformative features can decrease the statistical significance of a model. By decreasing the feature space linearly the model space is decreased exponentially, due to one-hot-encoding.

Therefore it is beneficial to perform feature selection before proceeding with model fitting.

In this section the feature selection is described in detail, the reasons for selecting or omitting each feature from the model are discussed. The features are described in Section 2.4.

Table 5.1 shows the results of feature selection. Observe that the features that are selected to be a part of the model are only the most interesting for this thesis. The other features may be used in further work.

Start-time is dependent variable of the models except when stated otherwise. Start-time is never omitted from the models since it is what is analysed in the thesis.

Up-Time was not included in the final model. Weak dependencies were noted between Up-time and start-time.Up-time was plotted against start-time. We noted that the data was not relatively uniformly distributed on the plot indicating that there is little or no dependence between up-time and start-time. It can be seen in Figure 5.1 that there is no obvious correlation between up-time and start-time.
This may be because automatically tested phones have in general a short up-time, either due to memory cleaning issues or due to the patches required for updates.

Start Type was used in the model. Not only was this heavily encouraged by the experts, but also found to correlate to start-time significantly. This was tested by chi-squared

Table 5.1: Features used

Feature	**Unit**	Used?
Time	in ms	Yes
Up-Time	in ms	No
Start Type	0 or 1	Yes
Android version	String	No
Package	String	Yes
Encrypted	Boolean	No
Frequency limit	String	Yes
Model	String	Yes
Group	String	No

test and is also supported by Table 5.3, where a large difference can be seen when comparing warm and cold start.

Android version was not included in the model. The feature was omitted because there are only a few android versions per data-set. In the data sets used in this thesis there was exactly one version per data set, with one being in the test and another in the training. This made the feature uninformative. In a different data set this could be an interesting feature to explore further.

Package was used in the model. Not only was this heavily encouraged by the experts we also found examples where it influenced the start time significantly. As this feature indicates the app that was used it is of great interest to Sony and is one of the main interest points of the thesis.

Encrypted was expected to have a significant effect on the start-time. But it did not. Most of the datapoints were not encrypted. The distribution of the start-times of apps on encrypted phones was plotted and noted that it did not significantly vary from the distribution of app start times on unencrypted phones. Since the difference was insignificant and the feature is not used by most Android users it was decided to not include the feature in models.

Frequency limit was used, it had a significant correlation to start-time. It is also of great interested to Sony.

Model was used in the model. Not only was this heavily encouraged by the experts but there where also examples found where it influenced the start-time significantly.

Group was not used. The data sets used in this thesis contained data from only one group. This makes the feature uninteresting on the analysed data sets.

5.2 Experimental Setup

Here follows a explanation of the experiments we ran during the thesis work.

5.2.1 Clustering

Given a data set consisting of 24 mobile phone models and total of 1,000,000 samples from all user groups. Note that this data set was only used for the clustering. The categorical

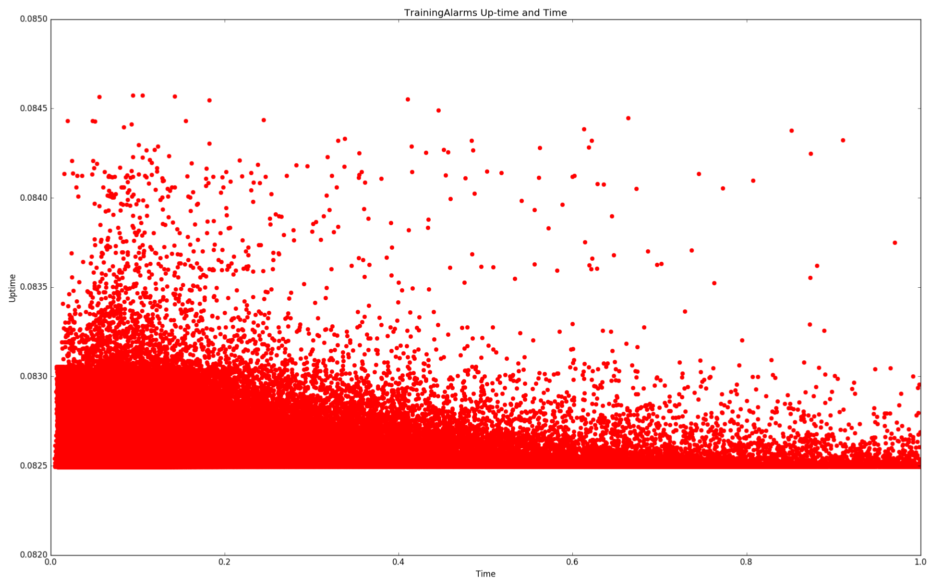


Figure 5.1: A plot of the Up-time and the start-time where the Up-time is on the y axis and the start-time is on the x axis. It can be seen that there is little or no dependence between up-time and start-time

features listed in Section 2.4 were one-hot-mapped. This resulted in approximately 80 features. We used the elbow method to find sufficient K-means clusters (James et al., 2013). Mean distance within-set was used as a measure of error in the clusters. K-means was carried out with 3 to 30 cluster-centers. The sufficient number of clusters was chosen as the number of clusters corresponding to the point in the error graph where the error did not decrease significantly when an additional cluster center was added.

5.2.2 Linear Regression

A large part of the thesis consisted of applying linear regression (LR) in different ways to the data. We will refer to three general models used as LR, LR using mean and LR using GMM. We ran all of the LR experiments with the same data. The training data set consists of 957,407 points and a test set of 111,350 data points. The datasets were chosen randomly, although such that the samples in training set were captured before the samples in the test set. The test and training subsets were chosen amongst samples with no missing feature values. For LR using mean and LR using GMM we grouped the data into different "groups" when taking the mean and fitting the GMM. We define a "group" as all of the data points that have the same value for every feature.

To show correlation between Model and Package, the Cartesian product of these two features was added to the data as an additional feature. The one-hot encoded Cartesian product adds a feature for every unique combination of the phone's model and package.

Simple Linear Regression

In this set-up data points were used without using groups. First we removed outliers, that is data-points with a start-time greater than 10000 ms were removed from the data-set. Then the categorical features were one-hot mapped to be usable in linear regression. Just before fitting the model the data was normalized to fit between 0 and 1 as described in Section 2.5. After the preprocessing we fitted a linear regression model to the data set. The fitted linear regression model was regularized with an l2-norm and fitted with stochastic gradient descent. The fitting was carried out for a maximum of 150 iterations. Otherwise default PySpark hyperparameters were used when fitting the regression model.

Mean or the Top of Kernel Density Estimation

The same preprocessing was done as in Section "Simple Linear Regression". We noted that Simple LR performed poorly. This is likely due to the few features that were available for the relatively complex data. To improve the performance of the regression model the data was grouped and a model estimating the different “groups” means was fitted. For the model the mean of a group was used as the dependent variable of the LR. Another model was built to estimate the maximum density top of each group’s Kernel Density estimation (KDE) instead of the group’s mean start-time. See Figure 5.6 for a visualization of a typical KDE curve and its maximum. The start-time that is at the maximal top of the probability density is most likely to occur and therefore represents the “group”.

With GMM

Gaussian Mixture model describes the probability distribution of the data with only a few parameters. Therefore the parametrization of the Gaussian mixture model captures more information from the distribution than simply the mean or the top of a “group”. The same data preprocessing was done as in Section “Simple Linear Regression”. Each group was fitted with a GMM model, as described in Section 4.4. The fitted GMM’s were used to better capture the behavior of the distribution. The mean (μ) and standard divination (σ) of the GMM distributions were used as the dependent variable. If the number of Gaussians in the best fitting GMM model was greater than one, multiple regressions were performed (one regression per Gaussian).

5.2.3 Alarm

The experiments consisted of testing and developing the Alarm pipeline that consist of 5 steps which are as follows, from Section 3.5:

1. Preprocessing.
2. Fit a distribution to the data.
3. Find differences in test- and training- distribution.
4. Fit a linear regression model.
5. Manually analyse the regression weights.

We used two data sets; the previously described data set consisting of approximately 1M training samples and approximately 100k test samples and a smaller data set, consisting of 5465 traning samples and 3330 test samples. Since the Alarm pipeline is looking for differences between the start-times of the test and training data, it was important to use some data that has known differences in start-times between the test and training data. This is the purpose of the smaller data set. The small-scale data set originates from two internal patches. One of the patches we known to be slower than the second patch. Note that these patches were internal and were never released to the public. We chose the slower patch to be the training data and the faster patch to be the test data. The training data is chronologically gathered before the test set.

The differences in start-times between the test and training data in the large-scale data-set are unknown. Experiments on these two data-sets can show that the Alarm pipeline can handle a larger unknown data-set as well as identify known difference in the data. Two separate experiments were carried out, one per data-set. In each the pipeline was applied to the test and training set.

5.2.4 Decision Trees

We used decision tree to find which features discriminate between the Gaussians in the GMM fitted to the data. In this application the Gaussians from the GMM were used as classes. The decision tree estimates the Gaussian a data point is expected to belong to.

We trained one decision tree per Alarm pipeline “group”.

5.3 Results

This section we will present the results of the experiments that were carried out. As the amount of data is large only examples of results are presented, as all result graphs would quickly become overwhelming for the reader.

5.3.1 Clustering

We applied the clustering algorithm k-means to the data. The clusters we received were uninformative. The clustering resulted in clusters along only one feature space. This can be seen in Figure 5.2, in the figure the different colors are the different clusters. It can be seen that only the start-time appears to determine the clusters. The feature that the clustering occurred on was dependent on the one-hot-encoding and standardisation. Clusters spanning patterns across several features were not achieved. Clustering did not yield the intended results. However creating the program for clustering gave insight on Spark and the data that was available.

5.3.2 Linear Regression

The results of the experiments with linear regression models are presented in residual plots, to visualize the fit of the multi-dimensional regression model on the data. The horizontal

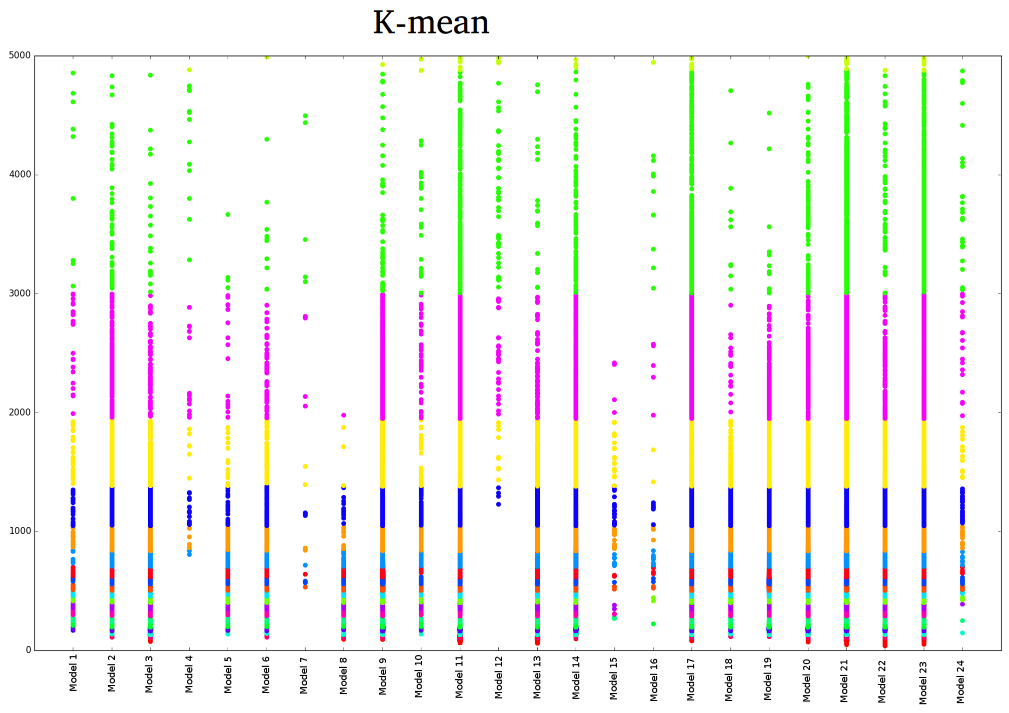


Figure 5.2: A plot of the K-means clustering with time on the y axis and model of phone on the x axis, note that the model is censored, the different clusters are represented using different colors

axis shows the observed value and the vertical axis the predicted value. In the middle of the graph the line $x = y$ is plotted. Points on this line were estimated with no error. The shortest distance to the line from a point is the residual error of the point. Ideally the line should intercept all the data points, but this is not likely to occur.

Simple Linear Regression

The simple linear regression had a problem with too few features for the number of data points, this resulted in a bad fit of the model. As is show in Figure 5.3 the model fits the data well for small values, but quickly looses the goodness of the fit. It can be seen that the model cannot fully express the variance in the data. This baseline test yielded a mean squared error of 0.049.

Mean or the Top of KDE

After introducing the group concept described in Section 2.4.2 we used the mean of the groups as the dependent variable. This resulted in a much better fit as can be seen in Figure 5.5. The same method was used for the maximums of KDE. These results are very similar, as can be seen in Figure 5.4.

The mean squared error of the group-mean linear regression model was 0.077, while the linear regression for the KDE-tops was 0.082. The tops were chosen as in Figure 5.6, as the start-time corresponding to the highest probability when probability density is

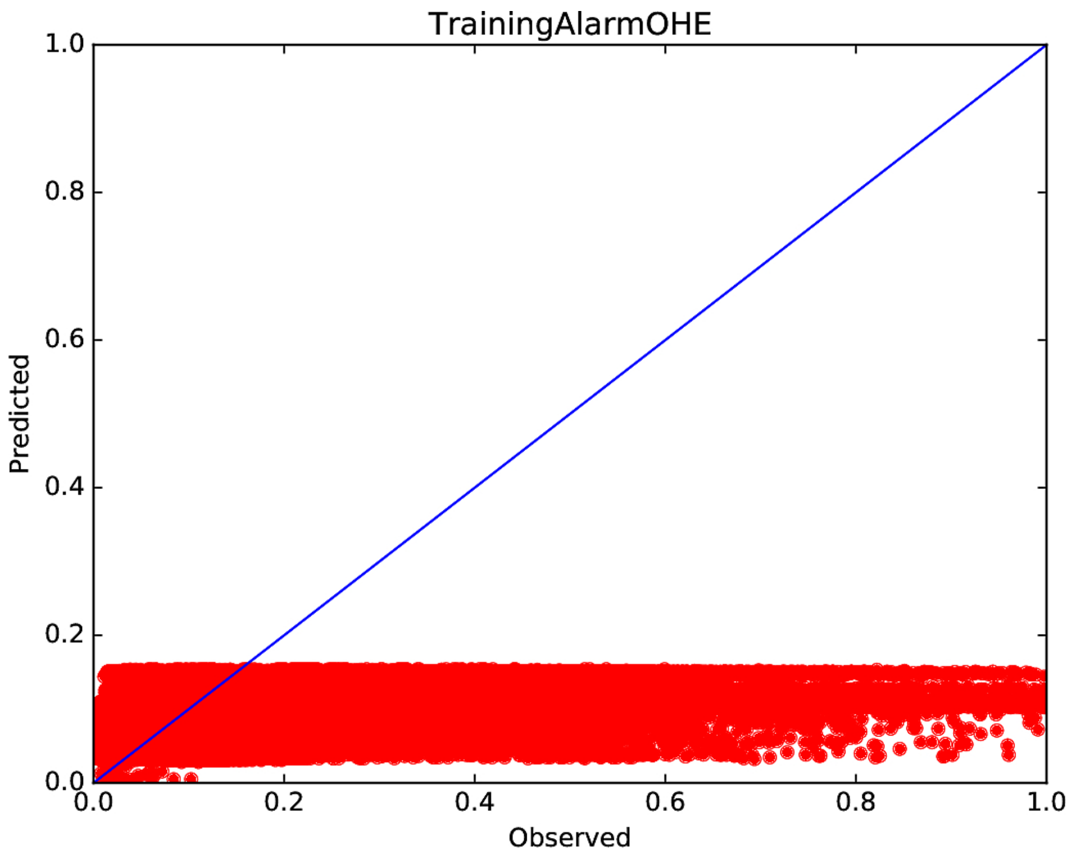


Figure 5.3: A residual plot of the linear regression with the predicted value on the y axis and the observed on the x axis, the blue line represents the goal line. Data points are the red dots and the data is scaled between 0 and 1. Note that it's a bad fit, with several data points map to the same value.

estimated with kernels. This graph is an example since for only one data set more than a thousand graphs were generated. But the general shape of the probability density curve is the most common shape of the start-time distribution. The shape of the curve varies slightly.

With GMM

Linear regression using the GMM fitted to the data fitted to the data with varying success. Overall the results were sufficient. As is shown in Figure 5.7 the model fits approximately as well as the top model, except for a few outliers that are relatively extreme. For each Gaussian in the fitted GMM we built two regression models; one for the standard deviation σ and one for the mean μ of the Gaussian.

Comparing the mean squared errors from these linear regression models yields the Table 5.2. These values seems to be lower than the MSE of the previously described linear regression model's MSE. It is important to note that the number of samples per regression model is dramatically decreased as the number of Gaussians is increased in the GMM. With fewer samples for a regression model a lower MSE could be reached. It is easy to see that the MSE is lower for the model in Figure 5.7 than for the model in Figure 5.8 because

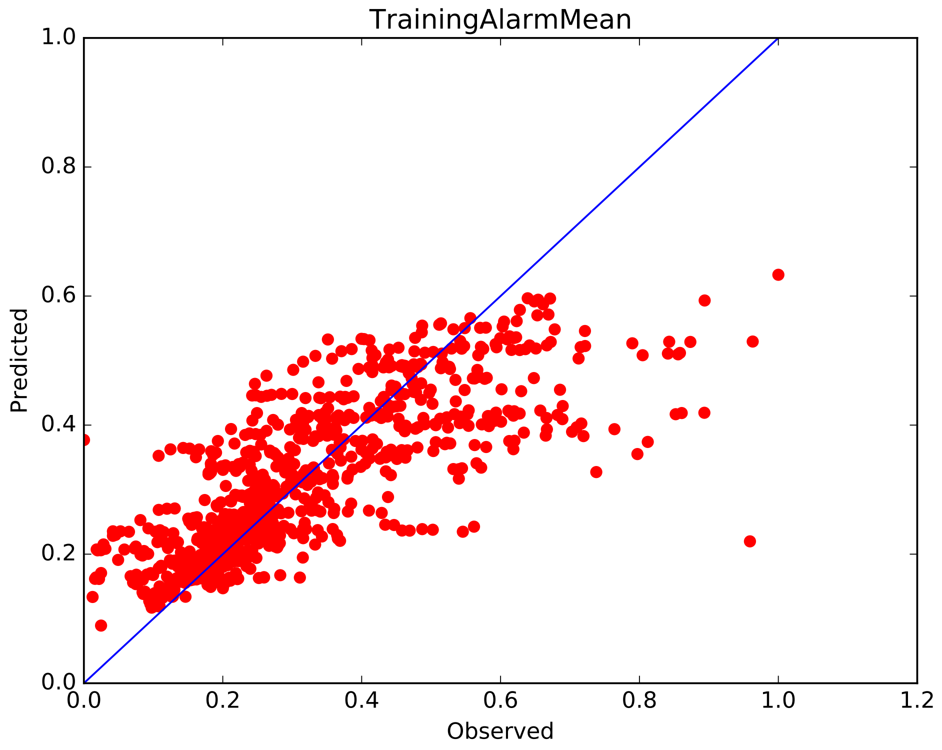


Figure 5.4: A residual plot of the linear regression with the predicted value on the y axis and the observed on the x axis, the blue line represents the goal line. Data points are the dots and the data is scaled between 0 and 1. This plot is of the experiments with the mean of a group as the dependent variable. Note that this model fit the data quite well, with a few outliers between 0.8 and 1.0 on the x axis.

.

it is easier to fit a regression line through the few samples in Figure 5.8.

5.3.3 Alarm

To test the Alarm pipeline a data-set was selected such that it would give conclusive results. That is a data-set were it was known that the start-times in the test set are significantly faster than in the training set. The full set of weights for the 16 fitted LR models are listed in the appendix.

After preprocessing large differences were found between the data sets. One of the greatest differences between the data sets is explored in the Section 5.3.4 with the help of a decision tree.

Figure 5.9 contains the LR model of the training set's μ_0. This model fits the data quite well. There are a few points that are clearly outliers and do not fit the model. These points are also the points with the slowest start-times making it seem that the LR is fitting worse at slower times. In the Figure 5.10 the LR model of the test data can be seen. The fitting

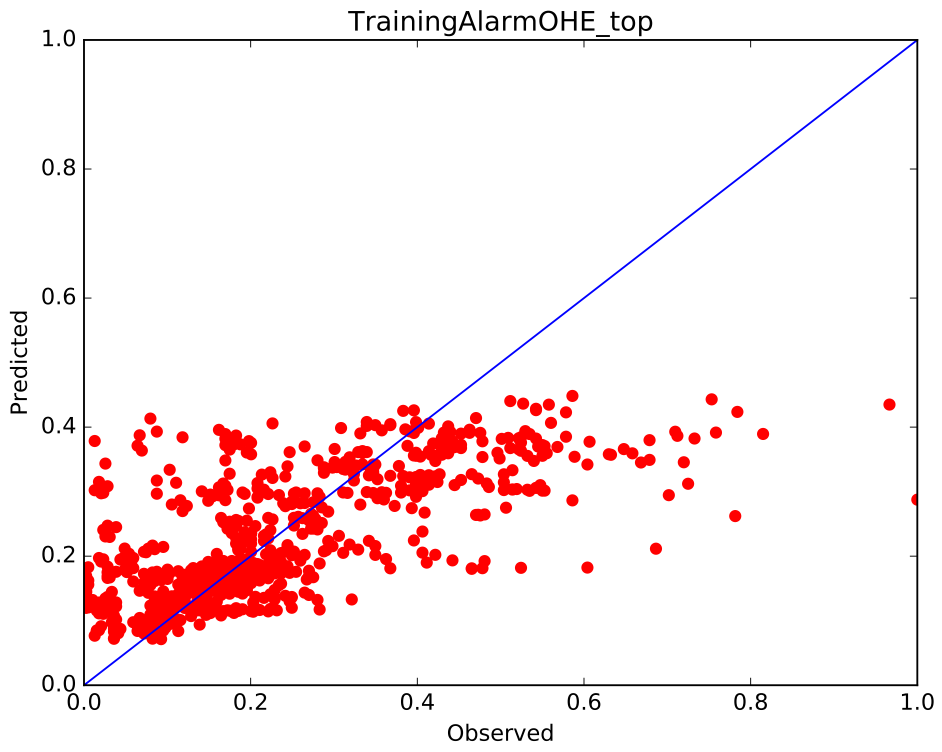


Figure 5.5: A residual plot of the linear regression with the predicted value on the y axis and the observed on the x axis, the blue line represents the goal line. Data points are the dots and the data is scaled between 0 and 1. This plot is of the experiments with the top of KDE as the dependent variable. Note that this model fit the data quite well, with a few outliers between 0.6 and 1.0 on the x axis.

has improved. There is only one outlier and the rest of the data points agree well with the model.

When looking at the weights in the test and training data LR models in Table 5.3 one can see that there has been a general decrease in the weights in the test compared to the training set. The app that had the longest start-times in the training set was the Album app, called “Album” in the table. The weight of the Album app variable has approximately halved from the training to test data, which shows that the start-times for that app have been improved in the test-patch.

Understanding the tables

Since the tables for the result of the alarm can be difficult to understand some clarification is needed. The weights are translated into ms to be easier to understand. An explanation for the features follows:

- The first feature is the intersect in the linear regression model.
- The phone’s model, as an abbreviation, for example Xperia Z5 becomes Z5.

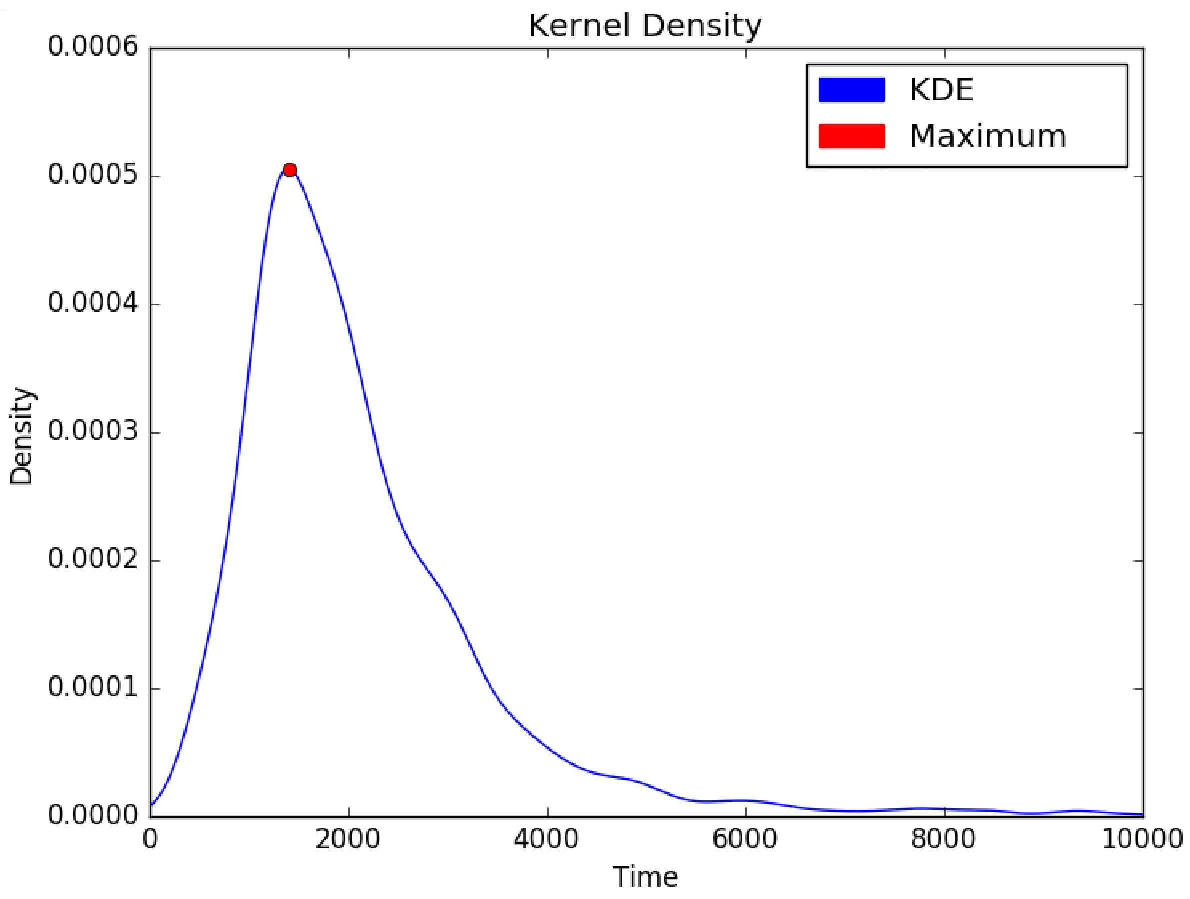


Figure 5.6: A plot of the kernel density estimation of data, with a dot on the maximum of the KDE. You can see that there is one significant maximum at 200. One could also see a maximum at 6000 and at 5000, but as they are not global maximums they are not a part of the linear regression model.

- The package of the apps being launched.
- The Cartesian product of the phone model and the the app being launched. If there is only one phone in the alarm this weight will be the same as the package weight. This feature is shown as the phone models abbreviation a space and the apps package, as example the cartesian feature of comexampleapp and Z5 is “Z5 comexampleapp”.
- The start type that the app has been launched with. The start type is noted as “other start” if the program has reported a strange number for start type, there are a few samples with this feature value. The feature value “other start” indicates high uncertainty of the true start type.
- The CPU frequency limit of the program. If the program has failed to report its frequency limit properly it reports back “UNKNOWN”. There are very few data points with “UNKNOWN” frequency limit. The numbers are in the style of “1.5 GHz x 4 0 0 0 2 GHz” with the current frequency limit in Hz and n x 4 means that the first four cores have the same frequency limit, n.
- The mean of the dataset is added at the bottom, this is not a part of the model but is

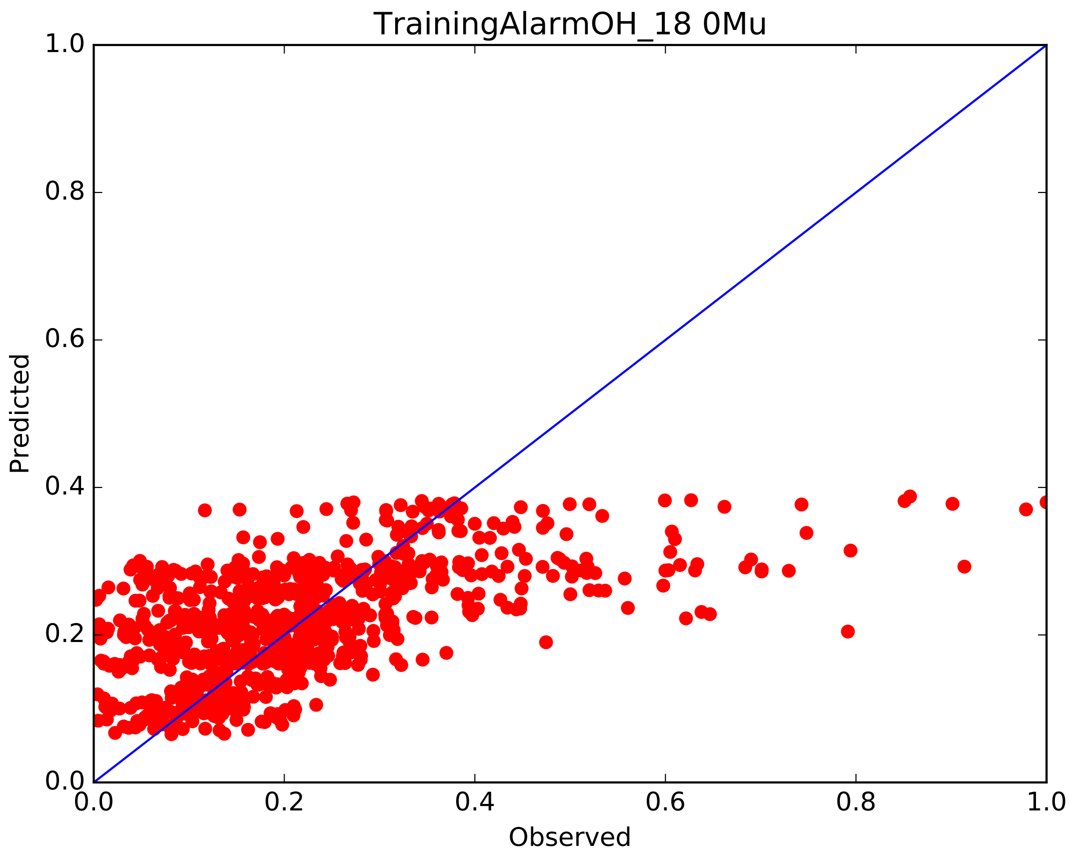


Figure 5.7: A residual plot of the linear regression with the predicted value on the y axis and the observed on the x axis, the blue line represents the goal line. Data points are the red dots and the data is scaled between 0 and 1, of the μ_0. Note that this model fit the data quite well, with a few outliers between 0.6 and 1.0 on the x axis.

added for reference.

5.3.4 Decision Trees

Decision trees were used to discriminate between different Gaussians in the fitted GMM. It generally resulted in informative decision trees that could be used to help experts find the differences between the different clusters.

An example of a fitted decision tree can be seen in Figure 5.11.

5.4 Discussion

Discussion of the results has been split into two different subsections, one for the Linear Regression experiments and one for the Alarm Pipeline.

Table 5.2: A Table of the mean squared errors for different dependent variables

Dependent variable	**MSE**
μ_0	0.083
μ_1	0.211
μ_2	0.187
μ_3	0.184
μ_4	0.110
μ_5	0.086
μ_6	0.006
σ_0	0.055
σ_1	0.133
σ_2	0.166
σ_3	0.162
σ_4	0.095
σ_5	0.054
σ_6	0.009

5.5 Linear Regression

Linear regression generally performed quite well, and the “grouping” concept that was introduced gave good results. It is interesting that the mean-regression model had a better fit than the top-regression model. It appears to be easier to fit a LR model of the mean, than the KDE top model to the data. This is likely to be the case because the means are likely to be similar to one-another.

The Cartesian product features that have been used could be extended to include more features, for example the start type and apps could be an interesting combination. Also when there only exists one product or app the Cartesian features are uninteresting and should be removed.

When looking at the graphs of 5.6 and 5.5 a few outliers can been seen that lie in the right bottom corner of the graph. These points belong to “groups” that have an uncommon start-time probability distribution. For example they can have a sinusoid shape, instead of the normal shape in Figure 5.12. This makes them interesting for further analysis by users of the system.

Other distributions than the Gaussians could also be used to fit the data, for example the Poisson distribution could work given the general shape of the kernel density estimation.

5.6 Alarm Pipeline

The greatest issue with the Alarm pipeline is the threshold for the Alarm to notice an error. That is, when the program should notice that there has occurred a change in the distribution of the data. This part has been explored to some extent but even when using methods to find the differences in the GMM distributions the problem of finding an appropriate threshold still stands. A suggestion is to allow the user of the program to insert such

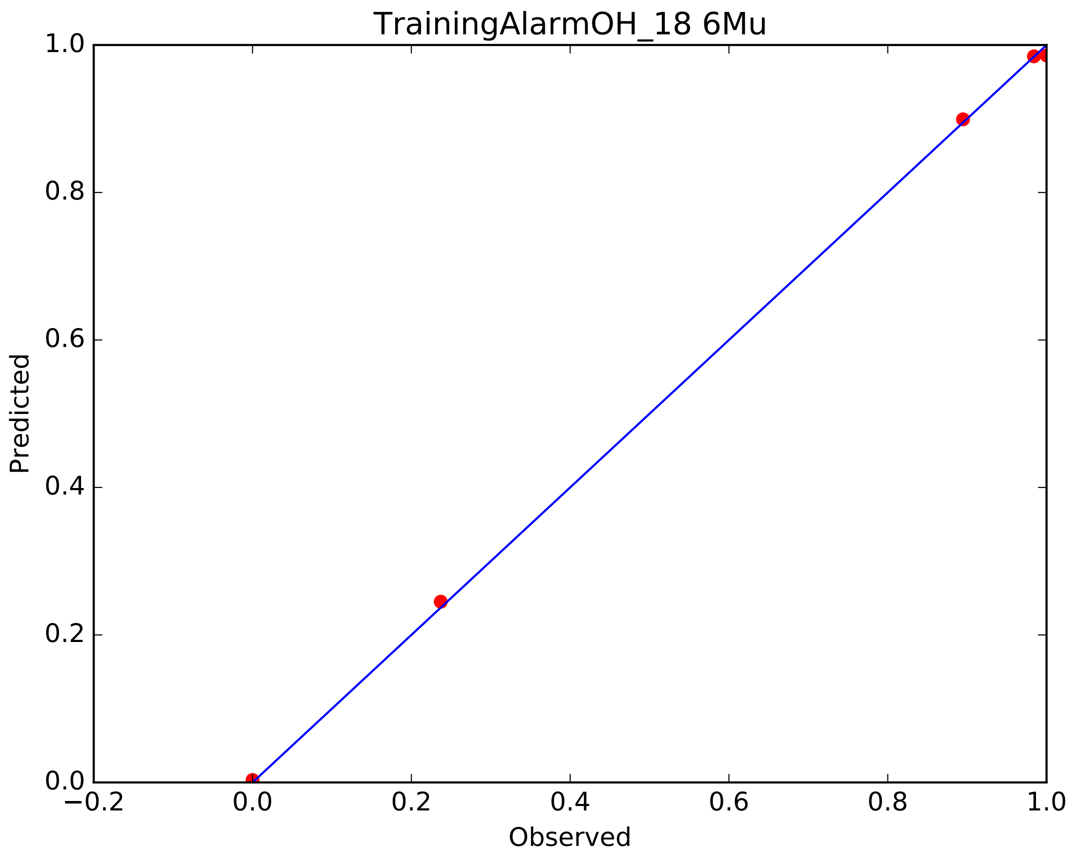


Figure 5.8: A residual plot of the linear regression with the predicted value on the y axis and the observed on the x axis, the blue line represents the goal line. Data points are the red dots and the data is scaled between 0 and 1, of the μ_6. The regression model fits the data points with low MSE this is due to the fact that there are so few data points in the model.

a threshold manually as this limit could vary. Another possibility is an implementation that uses classical statistic measures to detect large changes. These measures could work better but could also remove some of the transparency of the program from the user. The measures could also result in a scaling problems.

The features in the “groups” could be changed. The user of the pipeline may have interest in examining different features during varied applications of the Alarm Pipeline. Possibly reports could come in that the phones appears to be slow when encrypted, then this could be added to the feature list to investigate further.

Comparing the regression weights, is a non-trivial task and it gets more complicated the more features that are considered. To aid the analyst in this a percentage-change in regression weights could be presented.

The Alarm seems to be working as intended. In Table 5.3 it can be seen that most weights have the expected relative values. More experiments with live data could be interesting.

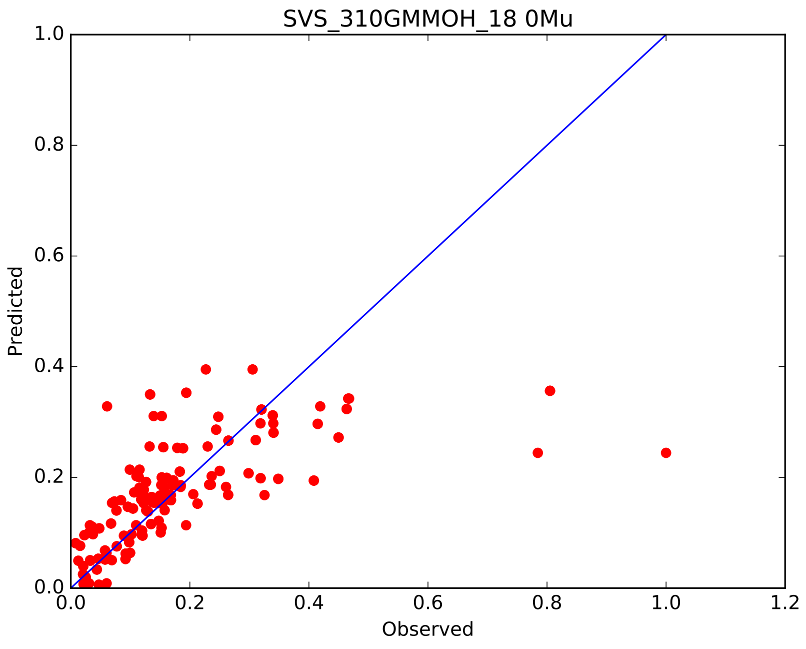


Figure 5.9: A residual plot of the linear regression with the predicted value on the y axis and the observed on the x axis, the blue line represents the goal line. Data points are the dots and the data is scaled between 0 and 1. This linear regression uses μ_0 as the dependent variable and is created for the traning data. Note that this model fit the data quite well, with a few outliers between 0.8 and 1.0 on the x axis

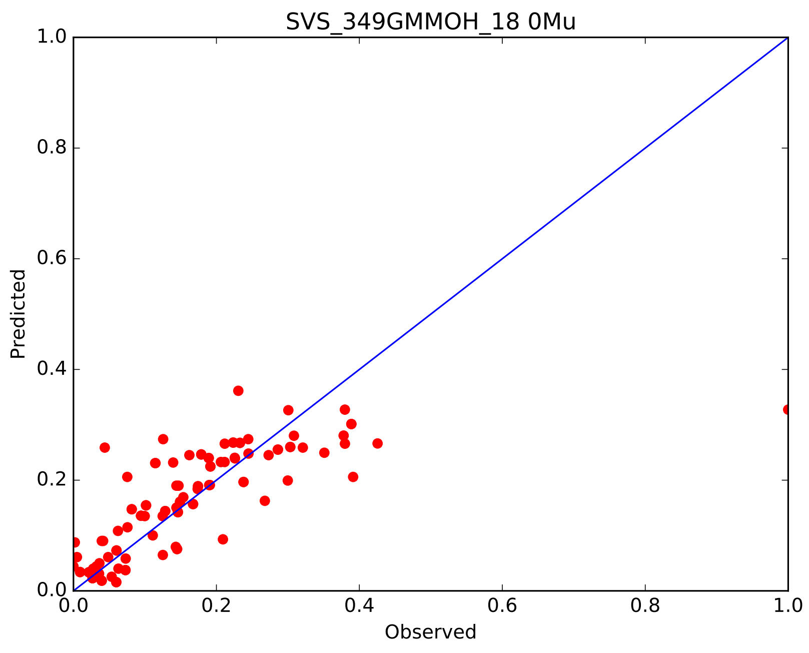


Figure 5.10: A residual plot of the linear regression with the predicted value on the y axis and the observed on the x axis, the blue line represents the goal line. Data points are the dots and the data is scaled between 0 and 1. This linear regression uses σ_0 as the dependent variable and is created for the test data. Note that this model fit the data quite well, with one outliers at $(1.0, 0.3)$

Table 5.3: A Table of the changes in weights (unit milliseconds) in the linear regression, using the SVS data-set, with the expected value of the Gaussian, μ_0, as the dependent variable. An explanation for the features can be found in Section 5.3.3. Some of the larger changes, both positive and negative, are marked with a gray box. Note that generally there are only a few weights that have had a larger change and often the changes absolute value is lower than 25 ms. Also one of the known problems with the data set was that album had a low performance which can be seen in the large change of Album's weight.

Feature	**Traning μ_0**	**Test μ_0**	**Change**
Intersect	132.48	109.18	−23.3
Z5	132.48	109.18	−23.3
Calendar	27.09	6.16	−20.93
Email	45.11	72.33	27.22
Settings	−44.4	−24.8	19.6
Album	184.99	97.45	−87.54
Camera	52.82	71.56	18.74
Organizer	−14.3	12.14	26.44
Z5 Calendar	27.09	6.16	−20.93
Z5 Email	45.11	72.33	27.22
Z5 Settings	−44.4	−24.8	19.6
Z5 Album	184.99	97.45	−87.54
Z5 Camera	52.82	71.56	18.74
Z5 Organizer	−14.3	12.14	26.44
Cold start	102.95	127.94	24.99
Warm start	−1.43	17.69	19.12
Other start	78.49	13.81	−64.68
1.5 GHz x 4 0 x 4	54.02	54.34	0.32
1.5 GHz x 4 0 0 0 2 GHz	−18.47	−8.51	9.96
1.5 GHz x 4 0 0 2 GHz 0	14.11	23.58	9.47
1.5 GHz x 4 0 0 2 GHz 2 GHz	66.23	4.86	−61.37
1.5 GHz x 4 0 2 GHz 0 0	5.33	13.98	8.65
1.5 GHz x 4 0 2 GHz 0 2 GHz	40.25	37.97	−2.28
1.5 GHz x 4 0 2 GHz 2 GHz 0	30.15	23.18	−6.97
1.5 GHz x 4 2 GHz 0 0 0	3.55	1.94	−1.61
1.5 GHz x 4 2 GHz 0 0 2 GHz	−20.83	14.54	35.37
1.5 GHz x 4 2 GHz 2 GHz 0 0	59.69	29.59	−30.1
1.5 GHz x 4 2 GHz x 4	61.87	27.35	−34.52
UNKNOWN	137.91	142.99	5.08
Mean	329.92	264.56	−65.36

Table 5.4: A Table of the changes in weights in the linear regression, using the SVS data-set, with the standard deviation of the Gaussian, σ_0, as the dependent variable. An explanation for the features can be found in Section 5.3.3. The standard deviation is harder to interpreter than the mean, as a small estimated standard deviation can be due to a small amount of samples or due to a small variance in the data. We can see that the largest reduction in standard deviation has been in the "Album" feature. Also note that "Organizer" has had a large increase in standard deviation, possibly suggesting a reduced performance.

Feature	**Traning σ_0**	**Test σ_0**	**Change**
Intersect	560.63	615.94	55.31
Z5	560.63	615.94	55.31
Calendar	81.64	220.25	138.61
Email	168.56	307.87	139.31
Settings	−177.36	17.72	195.08
Album	845.47	778.3	−67.17
Camera	465.05	483.75	18.7
Organizer	−3.07	237.55	240.62
Z5 Calendar	81.64	220.25	138.61
Z5 Email	168.56	307.87	139.31
Z5 Settings	−177.36	17.72	195.08
Z5 Album	845.47	778.3	−67.17
Z5 Camera	465.05	483.75	18.7
Z5 Organizer	−3.07	237.55	240.62
Cold start	329.46	493.74	164.28
Warm start	205.52	251.18	45.66
Other start	353.51	442.82	89.31
1.5 GHz x 4 0 0 0 0	290.51	366.57	76.06
1.5 GHz x 4 0 0 0 2 GHz	144.25	124.8	−19.45
1.5 GHz x 4 0 0 2 GHz 0	128.95	117.9	−11.05
1.5 GHz x 4 0 0 2 GHz 2 GHz	255.25	461.65	206.4
1.5 GHz x 4 0 2 GHz 0 0	208.82	221.88	13.06
1.5 GHz x 4 0 2 GHz 0 2 GHz	132.31	303.6	171.29
1.5 GHz x 4 0 2 GHz 2 GHz 0	90.36	377.86	287.5
1.5 GHz x 4 2 GHz 0 0 0	167.28	221.2	53.92
1.5 GHz x 4 2 GHz 0 0 2 GHz	234.05	222.86	−11.19
1.5 GHz x 4 2 GHz 2 GHz 0 0	200.36	265.18	64.82
1.5 GHz x 4 2 GHz x 4	165.09	275.51	110.42
UNKNOWN	454.94	803.98	349.04
Mean	1394.34	1144.79	−249.55

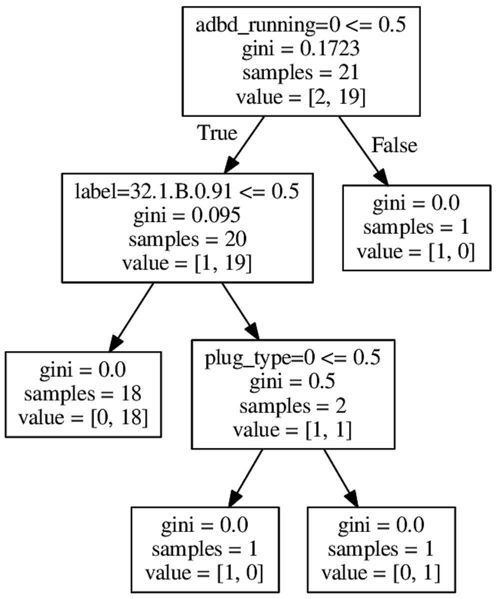


Figure 5.11: A Decision tree that separates two different Gaussians, one that is a faster Gaussian and on that is a slower Gaussian. The first line in a node is the splitting condition, note that all features are one-hot-encoded. On the second line is the Gini impurity value of the splitting condition. On the third line is the number of data samples in that node. On the last line is the number of samples that belong to each Gaussian, in the first node we have 2 samples in the fast Gaussian and 19 in the slower. The first features that is used to split is adbd running. The feature indicates if a debugging tool is running on the phone. Label is a unique identifier for the software version and lastly plug type shows the type of USB connection on the phone where zero is no connection. Note that plug type and adbd are features that are collected only on internal phones.

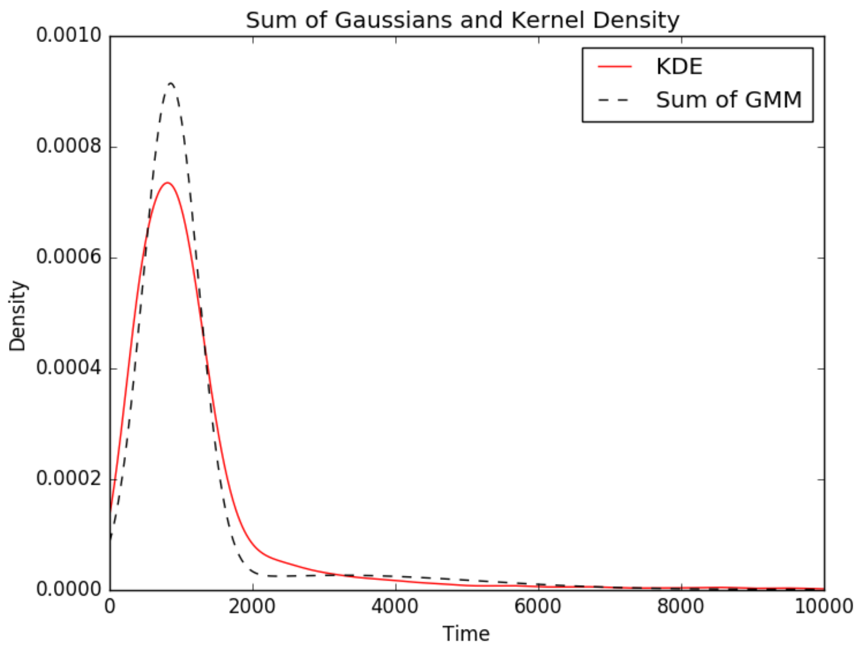


Figure 5.12: Plot of a KDE and GMM where the distribution has a standard shape. Note that the shape is Gaussian like and that the shape of the sum of Gaussian and the KDE are similar.

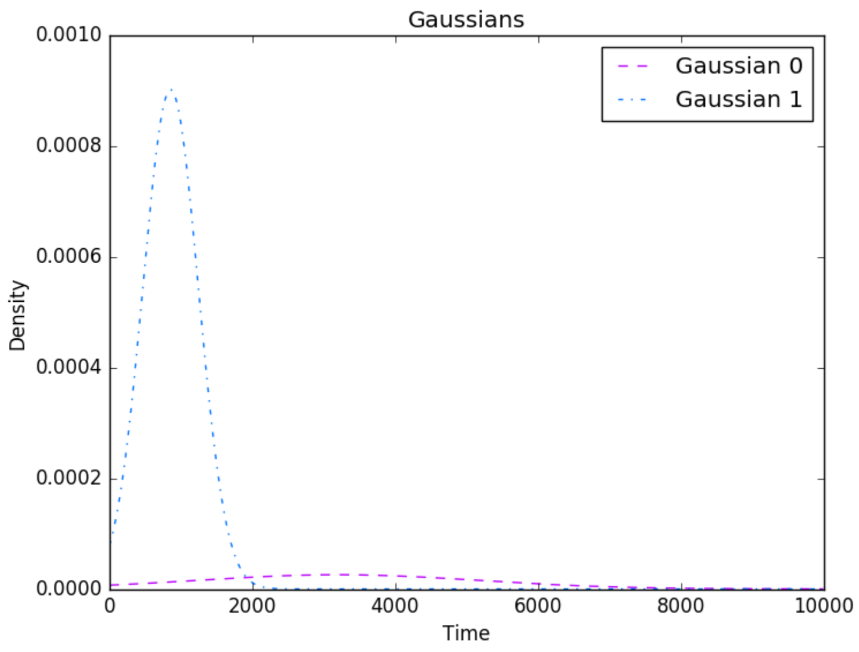


Figure 5.13: Plot of the Gaussians that make up the sum in Figure 5.12

Chapter 6
Conclusions

The goals of the thesis were the following:

1. Find features that affect app start-times,
2. Describe the distribution of the start-times,
3. Apply machine learning models on performance data.

The features found that significantly affect app start-times were app package name, product, start-type and frequency limit. The start-time distribution was fitted using GMM and represented using KDE.

Spark has proven to be a useful tool for implementing and applying machine learning to large amounts of data. Although the documentation of Spark has room for improvement. When combining PySpark and Jupyter, Spark became easy to use. After learning how to best use the Resilient Distributed Dataset (RDD) the programs became efficient also for large data sets. Jupyter enabled fast and iterative development.

The result of the thesis is the alarm pipeline that, based on machine learning models, with Android performance data from Sony Mobile can help Sony's developers improve app quality.

Bibliography

Bird, C., Ranganath, V.-P., Zimmermann, T., Nagappan, N., and Zeller, A. (2014). Extrinsic influence factors in software reliability: A study of 200,000 windows machines. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 205–214. ACM.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer, New York.

Bottou, L. and Bousquet, O. (2008). The tradeoffs of large scale learning. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems*, volume 20, pages 161–168. NIPS Foundation (http://books.nips.cc).

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., and Wirth, R. (2000). Crisp-dm 1.0 step-by-step data mining guide.

Devenish, A. (2015). Sony xperia and android 6.0, marshmallow. `http://blogs.sonymobile.com/2015/10/06/sony-xperia-and-android-6-0-marshmallow/`. (Accessed on 08/15/2016).

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.

Kwon, Y., Lee, S., Yi, H., Kwon, D., Yang, S., Chun, B.-G., Huang, L., Maniatis, P., Naik, M., and Paek, Y. (2013). Mantis: Automatic performance prediction for smartphone applications. In *Proceedings of the 2013 USENIX conference on Annual Technical Conference*, pages 297–308. USENIX Association.

Mingers, J. (1989). An empirical comparison of selection measures for decision-tree induction. *Machine learning*, 3(4):319–342.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1):81–106.

Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.

Van Der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M. J., Shenker, S., and Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28, San Jose, CA. USENIX.

Appendices

Table 1: A table of μ for the known data-sets

Feature	Training μ_0	Test μ_0	Training μ_1	Test μ_1	Training μ_2	Test μ_2
Constant	132.48	109.18	147.04	105.9	124.79	31.11
Z5	132.48	109.18	147.04	105.9	124.79	31.11
Calendar	27.09	6.16	−134.65	−4.79	−54.15	−94.93
Email	45.11	72.33	378.05	180.14	11.02	207.26
Settings	−44.4	−24.8	−80.54	68.49	−27.87	−90.36
Album	184.99	97.45	−12.17	−24.41	65.01	−16.26
Camera	52.82	71.56	64.77	31.05	134.03	74.17
Organizer	−14.3	12.14	−68.41	−144.57	−3.24	−48.75
Z5 Calendar	27.09	6.16	−134.65	−4.79	−54.15	−94.93
Z5 Email	45.11	72.33	378.05	180.14	11.02	207.26
Z5 Settings	−44.4	−24.8	−80.54	68.49	−27.87	−90.36
Z5 Album	184.99	97.45	−12.17	−24.41	65.01	−16.26
Z5 Camera	52.82	71.56	64.77	31.05	134.03	74.17
Z5 Organizer	−14.3	12.14	−68.41	−144.57	−3.24	−48.75
Cold start	102.95	127.94	313.77	63.33	313.55	212.93
Warm start	−1.43	17.69	−54.57	−14.01	−1.92	−128.71
Other start	78.49	13.81	−112.15	56.58	−186.83	−53.1
1.5 GHz x 4 0 0 0 0	54.02	54.34	49.48	−256.95	69.16	−242.79
1.5 GHz x 4 0 0 0 2 GHz	−18.47	−8.51	−36.33	5.92	−177.0	131.12
1.5 GHz x 4 0 0 2 GHz 0	14.11	23.58	181.94	384.3	13.12	170.31
1.5 GHz x 4 0 0 2 GHz 2 GHz	66.23	4.86	−91.5	−19.84	1.92	0.01
1.5 GHz x 4 0 2 GHz 0 0	5.33	13.98	−64.22	154.02	−73.53	267.52
1.5 GHz x 4 0 2 GHz 0 2 GHz	40.25	37.97	−21.61	−50.4	317.77	−224.28
1.5 GHz x 4 0 2 GHz 2 GHz 0	30.15	23.18	−71.25	−68.14	209.56	0.01
1.5 GHz x 4 2 GHz 0 0 0	3.55	1.94	−13.27	237.62	−36.88	−30.49
1.5 GHz x 4 2 GHz 0 0 2 GHz	−20.83	14.54	127.08	−71.67	−54.82	366.01
1.5 GHz x 4 2 GHz 2 GHz 0 0	59.69	29.59	−84.05	−116.03	38.97	0.01
1.5 GHz x 4 2 GHz x 4	61.87	27.35	110.5	−17.05	0.0	−224.28
UNKNOWN	137.91	142.99	104.93	1.03	39.0	−181.97
Mean	329.92	264.56	526.22	412.92	489.21	333.34

Table 2: A table of μ for the known data-sets

Feature	Training μ_3	Test μ_3	Training μ_4	Test μ_4
Constant	153.92	46.68	330.4	230.63
Z5	153.92	46.68	330.4	230.63
Calendar	0.03	0.03	286.28	0.04
Email	77.4	22.25	261.62	507.13
Settings	0.03	10.75	286.28	0.04
Album	144.43	70.4	323.64	−276.46
Camera	−67.86	−56.63	317.7	0.04
Organizer	0.03	0.03	286.28	0.04
Z5 Calendar	0.03	0.03	286.28	0.04
Z5 Email	77.4	22.25	261.62	507.13
Z5 Settings	0.03	10.75	286.28	0.04
Z5 Album	144.43	70.4	323.64	−276.46
Z5 Camera	−67.86	−56.63	317.7	0.04
Z5 Organizer	0.03	0.03	286.28	0.04
Cold start	−31.49	94.38	298.98	230.63
Warm start	0.03	0.03	286.28	0.04
Other start	185.44	−47.67	317.7	0.04
1.5 GHz x 4 0 0 0 0	−314.77	0.03	286.28	0.04
1.5 GHz x 4 0 0 0 2 GHz	−67.2	0.38	286.28	0.04
1.5 GHz x 4 0 0 2 GHz 0	42.76	0.03	352.04	0.04
1.5 GHz x 4 0 0 2 GHz 2 GHz	198.66	0.03	286.28	0.04
1.5 GHz x 4 0 2 GHz 0 0	29.29	102.93	286.28	−138.23
1.5 GHz x 4 0 2 GHz 0 2 GHz	0.03	0.03	286.28	0.04
1.5 GHz x 4 0 2 GHz 2 GHz 0	0.03	0.03	286.28	0.04
1.5 GHz x 4 2 GHz 0 0 0	86.08	12.68	286.28	507.13
1.5 GHz x 4 2 GHz 0 0 2 GHz	185.01	0.03	317.7	0.04
1.5 GHz x 4 2 GHz 2 GHz 0 0	16.28	0.03	286.28	0.04
1.5 GHz x 4 2 GHz x 4	0.03	0.03	286.28	0.04
UNKNOWN	−21.98	−69.22	233.22	−138.19
Mean	465.09	249.83	430.48	737.59

Table 3: A table of σ for the known data-sets

Feature	Test σ_0	Training σ_0	Test σ_1	Training σ_1	Test σ_2	Training σ_2
Constant	560.63	615.94	997.81	1076.91	1787.71	1001.38
Z5	560.63	615.94	997.81	1076.91	1787.71	1001.38
Calendar	81.64	220.25	−11.62	−26.07	−430.42	−112.06
Email	168.56	307.87	1220.64	809.07	2064.1	1986.47
Settings	−177.36	17.72	−119.64	−48.5	605.94	−565.75
Album	845.47	778.3	538.23	460.63	1722.29	60.34
Camera	465.05	483.75	490.45	451.69	1424.45	629.44
Organizer	−3.07	237.55	−227.28	974.69	−122.63	−122.63
Z5 Calendar	81.64	220.25	−11.62	−26.07	−430.42	−112.06
Z5 Email	168.56	307.87	1220.64	809.07	2064.1	1986.47
Z5 Settings	−177.36	17.72	−119.64	−48.5	605.94	−565.75
Z5 Album	845.47	778.3	538.23	460.63	1722.29	60.34
Z5 Camera	465.05	483.75	490.45	451.69	1424.45	629.44
Z5 Organizer	−3.07	237.55	−227.28	974.69	−122.63	−122.63
Cold start	329.46	493.74	558.66	1174.99	1448.91	1291.1
Warm start	205.52	251.18	−261.84	480.56	597.48	525.38
Other start	353.51	442.82	1058.18	39.2	1131.73	−465.33
1.5 GHz x 4 0 0 0 0	290.51	366.57	108.31	1106.53	2488.04	−230.7
1.5 GHz x 4 0 0 0 2 GHz	144.25	124.8	649.28	347.02	960.98	185.65
1.5 GHz x 4 0 0 2 GHz 0	128.95	117.9	602.3	177.34	−167.8	510.87
1.5 GHz x 4 0 0 2 GHz 2 GHz	255.25	461.65	−313.02	−139.0	−188.18	174.89
1.5 GHz x 4 0 2 GHz 0 0	208.82	221.88	376.7	−162.83	1730.86	579.33
1.5 GHz x 4 0 2 GHz 0 2 GHz	132.31	303.6	416.41	190.0	850.38	312.08
1.5 GHz x 4 0 2 GHz 2 GHz 0	90.36	377.86	−25.93	−311.22	−202.63	174.89
1.5 GHz x 4 2 GHz 0 0 0	167.28	221.2	172.67	985.11	1144.69	−216.56
1.5 GHz x 4 2 GHz 0 0 2 GHz	234.05	222.86	13.47	−117.07	458.86	46.76
1.5 GHz x 4 2 GHz 2 GHz 0 0	200.36	265.18	−306.86	725.63	694.92	174.89
1.5 GHz x 4 2 GHz x 4	165.09	275.51	67.31	225.76	695.2	440.31
UNKNOWN	454.94	803.98	1092.06	1162.12	333.47	772.73
Mean	1394.34	1144.79	2687.36	2428.54	5201.03	3444.51

Table 4: A table of σ for the known data-sets

Feature Test	σ_3	Training σ_3	Test σ_4	Training σ_4
Constant	2566.91	1118.58	2923.87	7071.0
Z5	2566.91	1118.58	2923.87	7071.0
Calendar	1780.74	419.77	2432.76	6810.93
Email	2734.45	−481.21	2520.55	6550.86
Settings	1780.74	−577.81	2432.76	6810.93
Album	1349.73	1302.88	3080.5	7331.07
Camera	2044.21	2134.02	2188.34	6810.93
Organizer	1780.74	419.77	2432.76	6810.93
Z5 Calendar	1780.74	419.77	2432.76	6810.93
Z5 Email	2734.45	−481.21	2520.55	6550.86
Z5 Settings	1780.74	−577.81	2432.76	6810.93
Z5 Album	1349.73	1302.88	3080.5	7331.07
Z5 Camera	2044.21	2134.02	2188.34	6810.93
Z5 Organizer	1780.74	419.77	2432.76	6810.93
Cold start	1726.36	1495.68	3168.29	7071.0
Warm start	1780.74	419.77	2432.76	6810.93
Other start	2621.29	42.67	2188.34	6810.93
1.5 GHz x 4 0 0 0 0	3142.24	419.77	2432.76	6810.93
1.5 GHz x 4 0 0 0 2 GHz	2482.0	−759.17	2432.76	6810.93
1.5 GHz x 4 0 0 2 GHz 0	1669.36	419.77	3996.94	6810.93
1.5 GHz x 4 0 0 2 GHz 2 GHz	2662.78	419.77	2432.76	6810.93
1.5 GHz x 4 0 2 GHz 0 0	−667.72	1305.58	2432.76	6296.42
1.5 GHz x 4 0 2 GHz 0 2 GHz	1780.74	419.77	2432.76	6810.93
1.5 GHz x 4 0 2 GHz 2 GHz 0	1780.74	419.77	2432.76	6810.93
1.5 GHz x 4 2 GHz 0 0 0	456.52	708.09	2432.76	6550.86
1.5 GHz x 4 2 GHz 0 0 2 GHz	594.8	419.77	2188.34	6810.93
1.5 GHz x 4 2 GHz 2 GHz 0 0	2390.99	419.77	2432.76	6810.93
1.5 GHz x 4 2 GHz x 4	1780.74	419.77	2432.76	6810.93
UNKNOWN	4081.88	1123.37	1604.1	7845.58
Mean	4446.43	3518.26	4392.4	8197.98

EXAMENSARBETE Patterns in live performance data
STUDENT Simon Svensson
HANDLEDARE Prof. Pierre Nugues (LTH) and Jens Gulin (Sony Moblie)
EXAMINATOR Jacek Malec (LTH)

Metod för att hitta förklaringar till långsamma app start-tider

POPULÄRVETENSKAPLIG SAMMANFATTNING **Simon Svensson**

Androidmarknaden är en gigantisk marknad som växer för varje dag. Den stora mängden data som genereras måste behandlas på ett bra sätt för att hjälpa utvecklarna att skapa bättre och snabbare program samt telefoner. Här presenteras ett arbetssätt som kan hitta problem med Android-appar och därför hjälpa utvecklare att lösa problem.

Sony Moblie samlar viss prestandadata från användare av telefoner som har tillåtit detta. Denna data innehåller bland annat tiden det tar för en app att starta, tiden från att användare klickar på appen till att den är redo att användas. Denna data har analyserats av prestandaingenjörer med hjälp av klassiska statistiska metoder och andra verktyg. I detta examensarbete har denna data analyseras med hjälp av maskininlärning, en typ av artificiell intelligens som använder stora mängder data för att lära programmet. Detta är för att hitta vad det är som gör starttiden för appar på telefonen långsamma. En metod för att hitta vad som gör telefoner långsamma har implementerats och testas för att använda i framtiden. Detta program kallas ”Alarm”.

Det första problemet var att representera tidsdata på ett bra sätt. Först testades det att använda tiden som den är till skapandet av modellen. Den modellen fungerade inte bra, det blev väldigt mycket fel i modellen och gav nästintill ingen information. Modellen byggd på denna tidrepresentationen kunde inte återskapa datan och lärde sig väldigt lite från datan.

Efter detta testades en metod med att använda ett medelvärde av tider för att representera en hel grupp av data, för att skapa en bredare approximation av datan.

Detta fungerade mycket bättre och gav ett tydligare resultat. Det enda problemet med att ta medelvärdet är att en stor del av informationen som finns i datan försvinner när endast medelvärdet används.

Därför testade att beskrivas datan med normalfördelningar. Detta gav ett liknade resultat som att använda medelvärdet men, det blev mycket lättare för utvecklare som använder programmet att förstå vad som hade hänt. Detta vidare utvecklades sen till ”Alarm” programmet. Det består av fem steg.

1. Förarbete och städning av datan.
2. Hitta en distribution som passar data.
3. Hitta skillnader från tidigare distributioner.
4. Använda linjär regression, en metod för att beskriva datan som en linje, på datan.
5. Analysera linjära regressions resultatet för att hitta vad som har ändrats.

När systemet används görs detta genom att använda två olika dataset och jämföra skillnaderna mellan deras resultat. Via experiment har vi kunnat visa att ”Alarm” kan hitta att till exempel en app har blivit mycket långsammare.