Detection of human alertness using supervised learning

Anders Hørsted

Kongens Lyngby 2011
IMM

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Summary

This is the summary/abstract

Resumé

På dansk

Preface

This thesis was prepared at Informatics Mathematical Modelling, the Technical University of Denmark in partial fulfillment of the requirements for acquiring the Ph.D. degree in engineering.

The thesis deals with different aspects of mathematical modeling of systems using data and partial knowledge about the structure of the systems. The main focus is on extensions of non-parametric methods, but also stochastic differential equations and neural networks are considered.

The thesis consists of a summary report and a collection of ten research papers written during the period 1996–1999, and elsewhere published.

Lyngby, December 1999

Henrik Aalborg Nielsen

Acknowledgements

I thank my...

Contents

Chapter 1

Introduction

Igennem de sidste 30 år er computerens ydeevne vokset markant. Den forbedrede ydeevne har givet mulighed for at udføre statistisk dataanalyse, i et omfang der ikke tidligere har været muligt. En af de ting der er blevet mulighed for, er at programmere såkaldte "classifiers". Et godt eksempel på en classifier, er de programmer bankerne bruger til at opdage evt. snyd med kreditkort. Baseret på alle tidligere korttransaktioner, og viden om hvilke der var "falske" transaktioner, kan bankerne programmere en classifier, der med høj præcision kan forudsige om en ny transaktion er snyd eller ej.
Denne opgave tager udgangspunkt i en konkurrence på hjemmesiden kaggle.com. Konkurrencen er udbudt af Ford Motors og handler om at udvikle en classifier, der kan forudsige om en bilist er ved at blive ukoncentreret, mens han/hun kører bil. Til at udvikle denne classifier har Ford foretaget en række målinger på bilister, mens de kørte bil, og til hver måling er det blevet noteret om bilisten var opmærksom eller ej. Med udgangspunkt i disse målinger udarbejdes – ved brug af de mest gængse metoder – en række classifiers, og deres evne til at klassificere ny data sammenlignes.

1.1 The competition

1.1.1 Problemformulering

I denne opgave vil jeg...

- ... bruge de mest gængse klassifikationsmodeller (nearest neighbour, logistic regression, neural networks og SVM) til at lave en classifier der (forhåbentlig) kan forudsige om en bilist er ved at falde i søvn.
- ... undersøge hvor stor indflydelse den indledende databehandling (feature selection og outlier removal) har på det endelige resultat.
- ... undersøge om classifieren kan forbedres ved at implementere en Hidden Markov Model, der tager hensyn til det temporale aspekt af data.
- ... implementere en ensemble classifier, der kombinerer resultatet af flere classifiers i én classifier.

TODO: Remeber to define the different datasets

Chapter 2

The Ford Challenge

In this chapter, the Ford competition that is the basis for this report, is introduced. Before introducing the Ford Challenge, a short overview of other online machine learning competitions is given. As part of introducing the Ford competition, the kaggle.com website that hosted the competition is presented, and the data set used in the competition is described in detail. But as metioned the chapter starts with a short overview of other online machine learning competitions.

2.1 Other online machine learning competitions

To get a little perspective on the Ford Challenge, this section gives a short review of some past and present online machine learning competitions.

2.1.1 The Netflix Prize*

One of the most talked about competitions, may very well be the Netflix Prize. The Netflix competition was launched on October 2, 2006 and the aim of the competition was to predict how users would grade new movies, based on a large dataset of previous grades. Why did the Netflix Prize gather a lot of attention? First of all the grand prize was $1M, which is a lot of money. And secondly the dataset was huge, consisting of 100,480,507 ratings given by 480,189 users, leaving room for a lot of interesting new techniques to be tested.

When the Netflix Prize was awarded on September 18, 2009, 5169 different teams had participated in the competition. The winning team consisted of three different teams, that at one point decided to team-up and compete as a join-team.

Netflix originally wished to follow up the Netflix Prize, with another competition but decided to dismiss the idea, due to a lawsuit regarding privacy concerns related to the first Netflix Prize.

2.1.2 KDD Cup

Although the Netflix Prize gathered a lot of attention, it wasn't the first online machine learning competition. An example of an earlier competition, is the KDD Cup. The KDD Cup is a competition that is held every year, and it started back in 1997. The subject changes every year, and can be anything from mining purchase data from an online store, to computer aided detection of breast cancer (the 2000 and 2008 competitions respectively).

This year the KDD Cup is held in cooporation with Yahoo! Labs and the task is to predict user ratings of musical items (both tracks, albums, artists and genres). One noteworthy detail about the 2011 KDD Cup is the huge data set, containing over 300 million ratings of more than 600,000

*The section about The Netflix Prize is based on Wikipedia (2011) and Netflix (2011)

distinct items. [†]

2.1.3 And many others

The Netflix Prize and the KDD Cup are just two examples of online machine learning competitions. Many others exists, such as

- *The Hearst Challenge 2011* - Every year The Hearst Corporation hosts a machine learning competition. This year the task is to data mine the history of 1.8 million emails sent to subscribers of Hearst's publications, and then predict who will open emails in the future.
  Read more at `http://www.hearstchallenge.com`

- *The Reclab Prize* - RichRelevance is a company that specializes in online product recommendation. They offer a $1M prize for the team that first improves their product recommendation algorithm by 10%.
  Read more at `http://www.overstockreclabprize.com`

- *The Heritage Health Prize* - In this competion a dataset of anonymized patient data should be used to create a classifier that predicts and prevents unnecessary hospitalizations. With a first prize of $3M, and the ethical issues surrounding the use of machine learning on patient data, this competition has received much attention.
  Read more at `http://www.heritagehealthprize.com/`

Since almost all the machine learning competition websites, needs the same functionality, websites that specializes in hosting competitions has appeared. One example of such a website is the kaggle.com website, which hosts The Ford Challenge.

[†]Read more about the KDD Cup history at ACM (2011). For more info about the 2011 competition see Labs (2011)

2.2 Kaggle.com and The Ford Challenge

The first competition hosted by kaggle.com was started in April 2010, and since then a total of 18 competitions have been held. Every competition at kaggle.com has some background information, links to the data sets, a submission system and a forum, as seen in figure 2.1.

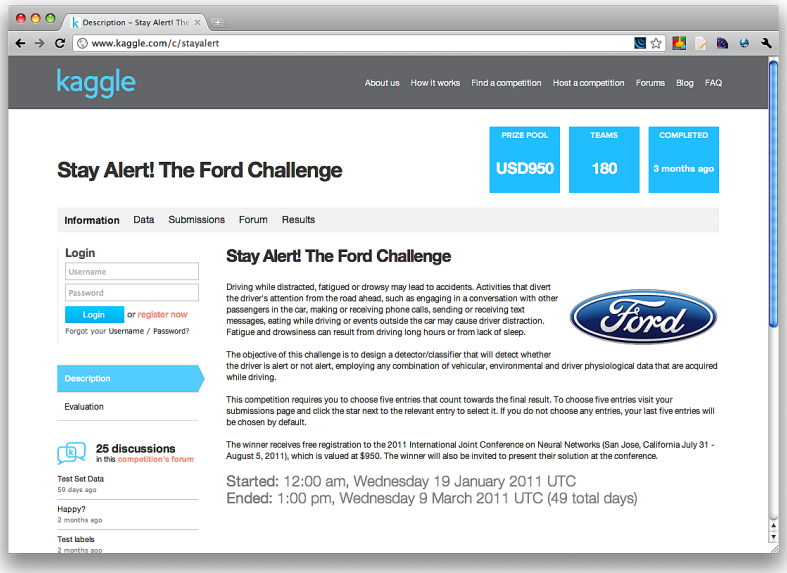


Figure 2.1: *The Ford Challenge frontpage at the kaggle.com website*

The Ford Challenge began on January 19, 2011. The task was to create a classifier that is able to detect when a driver is about to get distracted while driving. The dataset that Ford made available for the competition consisted of measurements of 30 different features, measured on drivers, along with a binary feature (IsAlert) that was 1 if the driver was alert and 0 otherwise. The 30 features was a mix of environmental, driver physiological and vehicular features[‡]. Based on this dataset a classifier should be made, that predicts the IsAlert feature of a distinct test dataset held by Ford.

[‡]The dataset is explained in much more detail in the section 2.2.1

One detail that made the competition slightly different than many other competitions, was that Ford would not disclose any information about what the different features represented[§]. The official reason was that

> "We like to encourage the participants to pursue classification without preconceived notions based on prior knowledge of the subject, focusing on variables which lead them (based on their own experiments) to better classification." (Abou-Nasr, 2011c)

Doubts about the true motive behind the lack of details about the features, was expressed by, what later turned out to be, the winner of the competition (Inference, 2011b)

The performance of the classifiers was measured by calculating the AUC score (see section 4.5.1) of the classifiers, on the test set. A limit of two submissions per contestant was set as a way to counteract the possibility of someone reverse-engineering the IsAlert-feature of the test dataset. This could of cause be circumvented by one person registering more than once. And this was exactly what happened. On March 9, 2011 when the competition ended, two users had achieved exactly the same AUC (six significant digits) and other contestants immediately questioned the probability of two unrelated users getting exactly the same AUC (Pardos, 2011). One of the two leaders (Rosanne/Shen) quickly admitted that he and his friend had indeed used two accounts to get 4 submission per day (Rosanne, 2011), and after some discussion in the forum, the leaders were disqualified. The end result was that the user Inference in third position was declared the official winner of the competition.

Two weeks after the competition ended, Inference described the technique used to win the competition. This will be described in detail in chapter 5.

2.2.1 The dataset

The dataset used to create classifiers for the Ford Challenge was released on the competition website, the day the competition started. The dataset

[§]see forum replies from the Ford spokesperson (Abou-Nasr, 2011b,a)

TrialID	ObsNum	IsAlert	P1	…	P8	E1	…	E11	V1	…	V11
0	0	0	12.2	…	1.2	4.3	…	33	12	…	7.34
⋮											⋮
0	1200	1	11.1	…	10.7	1.3	…	21	8	…	8.82
⋮											⋮
510	1198	0	11.1	…	10.7	1.3	…	21	8	…	8.82

Table 2.1: *Structure of the data set*

consisted of a number of trials and each trial was approximately 2 minutes of sequential data recorded every 100ms during a driving session on the road or in a driving simulator (Kaggle.com, 2011). As the interval between two datarow was 100ms, each trial consists of approximately $2\,\text{minutes} \cdot 60\,\frac{\text{secs}}{\text{minute}} \cdot 10\,\frac{\text{rows}}{\text{sec}} = 1200$ rows.

Each row has a total of 33 data columns structured as shown in table 2.1. Some details are worth noticing:

- The `TrialID` starts at 0 and the trials from 469 to 479 (both inclusive) are missing. The last trial has `TrialID=510`. This gives a total of exactly 500 trials.
- The `ObsNum` also starts at 0 and there are not exactly 1200 observation for every trial.
- The total number of rows is 604,229
- The row number is not part of the data set, so a row is uniquely identified by the pair (`TrialID`, `ObsNum`).

As mentioned before no additional information about what the different features represent or what datatype (discrete, continous) they are, was disclosed by Ford. The only way to get these informations is by doing a thorough data exploration of the dataset and that is what the next chapter is about.

Chapter 3

Data exploration

In this chapter the dataset for the competition is explored. First some summary statistics are calculated and after that the datatype of the features are tried to be determined. Then outlier detection is attempted and some thoughts about what defines an outlier are presented. It is also attempted to find features that discriminates well between the alert and the not-alert datapoints. And finally a Principal Component Analysis of the data is calculated.

3.1 Overview of the datasets

When reading this report some confusion about the naming of different datasets may arise. In this section a short overview about the datasets is given.

When The Ford Challenge was presented, two datasets was released.

- The *Ford trainingset* - This is the dataset introduced in the previous section, and consists of 604,229 rows, 500 trials and 33 data columns. As the name suggests, this dataset was used to train models, when participating in the Ford Challenge
- The *Ford testset* - This dataset consists of 120,841 rows, 100 trials and 33 data columns. But the column for the `IsAlert` feature had no data. The task of the Ford Challenge was to predict the `IsAlert` feature of this dataset.

The evaluation of a participants prediction on the Ford testset was done by uploading the Ford testset, with the predictions in the `IsAlert` column, to the kaggle.com website. Unfortunately the true `IsAlert` feature of the Ford testset wasn't released after the competition ended, and therefore the Ford testset can't be used for calculations in this report. The Ford testset will play a role in some of the discussions though.

Since a testset was needed*, the Ford trainingset wassplit in two parts, namely the *report trainingset* and the *report testset*. The split was done by taking every fifth trial in the Ford trainingset and assign it to the report testset. All other trials was put in the report trainingset. The report trainingset and report testset are often just referred to as *the trainingset* and *the testset*.

3.2 Calculating common statistics

To start the data exploration, four common statistics, namely the mean, min, max and standard deviation, of every feature across the whole dataset was calculated. The standard deviation was calculated as (with n equal the total number of rows in the dataset, x_i equal to the i'th value of any of the features, and $\bar{x} = \sum_{i=1}^{n} x_i$)

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

*see section 4.5 for an explanation of why a testset is needed

Feature	Mean	Min	Max	Std.Dev
E9	0.87	0.00	1.00	0.33
IsAlert	0.58	0.00	1.00	0.49
P6	843.73	128.00	228812.00	2795.32
P8	0.00	0.00	0.00	0.00
V5	0.18	0.00	1.00	0.38
V7	0.00	0.00	0.00	0.00
V9	0.00	0.00	0.00	0.00

Table 3.1: *Highlights from the results of the summary statistics. See appendix B.2 for all results.*

The source code for the calculations can be found in appendix B.1 and all results in appendix B.2. Most results did not tell that much, but a few results stood out. These are shown in table 3.1. The table shows that the features `P8`, `V7` and `V9`, are zero throughout the whole dataset, and can be ignored. The features `E9` and `V5`, could be binary, but that is only speculation at the moment. Finally the feature `P6` has a mean of 843.73 and a standard deviation of 2795.32, but its maximum is 228812.00, which is many, many standard deviations away from the mean. This could be a sign of some serious outliers, but it could as well be a single trial with a mean far from the other trials. Further investigation is needed to conclude anything here. Finally it is seen that the mean of the `IsAlert` feature is only a little above 0.5, and therefore only a little over 50% of the time are the drivers alert.

The results of this first step have been to exclude a few features, and get some rough ideas about the shape of some other features. It is now time to really get to know the different features.

3.3 Determining the datatype of features

Since Ford would not disclose any information about the different features, it is important to get a good picture of which features are discrete/-categorical and which features are continous. A natural first step to learn the datatype of the features, is to calculate the number of unique values each feature takes.

3.3.1 Unique values

It requires a little thought to pinpoint exactly what needs to be calculated. On one hand it is natural to calculate the number of unique values a feature takes across the whole dataset. On the other hand it could happen, that a discrete feature might have only (eg.) 3 unique value within any trial, but the values differ between various trials. This way we would have a feature with 1500 unique values across the whole dataset, but with max 3 unique values within any given trial. Therefore the number of unique values within a trial is calculated for every feature. Based on this result, the minimum and maximum number of unique values within a single trial, is calculated for every feature. The source code can be found in appendix B.3 and results are shown in table 3.2.

A couple of things should be noticed about the results. Almost all features have som trials where they only take on one unique value. For categorical features, this could be perfectly normal, but for a continous feature it seems to be pretty unlikely to have only one value in a trial spanning two minutes. Is a feature, that in some trials seems continous, but then only have one value in other trials, just turned off in the latter trials? And how should the feature be handled in trials where it is "off"?. This discussion is continued in section 3.4 about outlier detection.

From table 3.2 it is also seen, that only features `P1`, `P2` and `V11` is consistently having lots of unique value, across all trials. It seems fair to call these features continous. Feature `V1` have some trials where it takes on 969 different values, but it also have trials with only one unique value. Across the whole dataset it takes on 12374 unique values. This could be explained by feature `V1` being continous in a small number of trials (about 15-30), and turned "off" in all other trials. Further exploration will show whether it is true or not.

Another interesting detail is that there are two pairs of features ((`P3`,`P4`), (`P6`,`P7`)), that share exactly the same number of unique features. Both min and max and total. This indicates a possible relationship between the features, and later (section 3.5.1) it is shown that this is in fact true.

A final remark regarding the number of unique values, is that feature `E9`

Feature	Min in trial	Max in trial	Whole dataset
E1	1	131	12265
E10	1	86	121
E11	1	79	116
E2	1	129	28502
E3	1	3	3
E4	1	97	249
E5	1	134	254
E6	1	116	247
E7	1	15	25
E8	1	10	10
E9	1	2	2
IsAlert	1	2	2
P1	967	1208	105715
P2	1155	1207	128666
P3	24	106	406
P4	24	106	406
P5	34	114	1070
P6	3	99	406
P7	3	99	406
V1	1	969	12374
V10	1	5	5
V11	875	1206	166455
V2	1	76	90
V3	1	29	33
V4	1	204	324
V5	1	2	2
V6	1	775	2727
V8	1	239	323

Table 3.2: *The minimum and maximum number of unique values within the trials, for every feature in the dataset. Also the total number of unique values for each feature, across the whole dataset, are shown.*

and V5 indeed are binary. Also if a limit of max 40 unique values within a single trial is set as an indicator for categorical features, E3, E7, E8, V3 and V10 are seen to be categorical.

3.3.2 Plotting some features

To get a more nuanced picture of the various features, it is now time to visualize the data. The first thing of interest is to plot every feature for a couple of trials, to see the structure of the features, and how much they vary between different trials. For a start some random plotting was done in the interactive ipython-shell[†].

Some selected plots from the interactive plotting session are shown i figure 3.1. The main results from the introductory plotting of features are that:

- A feature varies alot in structure between different trials
- Many features are constant in some trials (was already hinted at in section 3.3.1)
- In a few trials some features deviates a lot from their "normal" structure, (more about that in section 3.4 about outliers).
- The feature P2 looks like white noise**TODO: Is this right?**
- The features E3, E7, E8, V3 and V10 does indeed seem to be categorical features.

Apart from the observations above, a subjective conclusion from the plots of the features is that the data quality could be better. Since the data quality isn't perfect it is important to consider how outliers should be dealt with. This is what the next section is about.

[†]For details about the software setup used, see chapter A

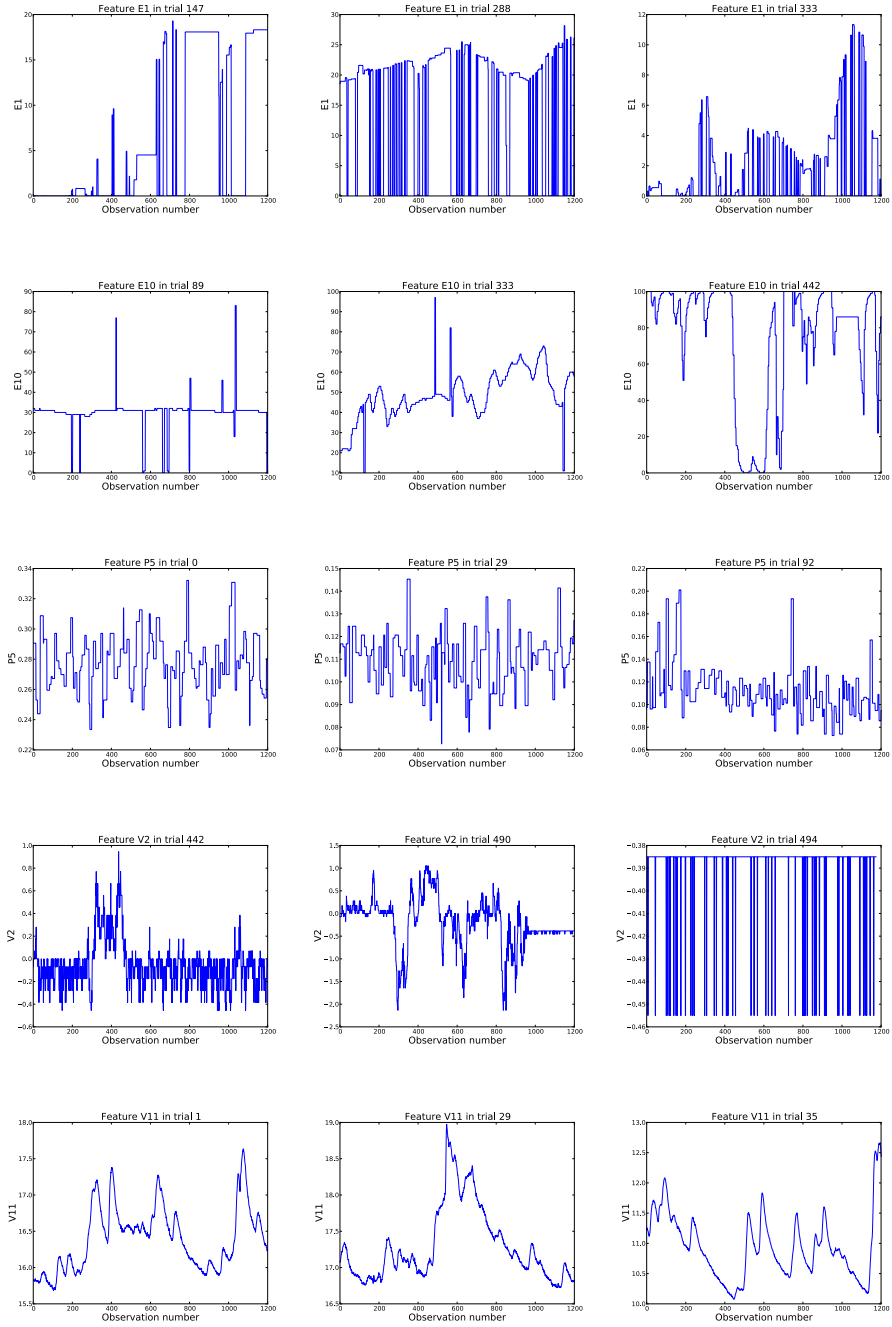


Figure 3.1: *Plots of the features E1, E10, P5, V2, V11, for a couple of trials. The trials was selected to show the diversity a single feature exhibits between trials.*

3.4 Outlier detection

The textbook approach for detecting outliers is to create boxplots of the different features, and then argue that data points outside the 90-percentile should be removed. Although this is a naive approach some useful information could be obtained from boxplots of the features. But it doesn't necessarily make sense to make boxplots of the features for the whole dataset. Some features have the same structure in different trials, but their mean varies. Eg. feature `V11` in trials 29 and 35 have similar structure but the range in trial 29 is $[16.75, 19]$ and $[10, 12.6]$ in trial 35 (see figure 3.1). If a feature has a different mean in only a few trials, those datapoints could be seen as outliers, if we look at the feature across the whole dataset. Instead boxplots of the features for the 50 first trials are created. This also gives a visual idea about how feature means varies from trial to trial.

3.4.1 Making boxplots of features

The boxplots for 4 selected features are shown i figure 3.2. All boxplots are in appendix B.6, and the source code can be seen in appendix B.5.

From the boxplots a couple of things are of interest.

- For some features the trial mean varies a lot between different trials **TODO: Remember to check with boxpots in appendix**
- Within a single trial some features still have a lot of data points outside the 10- and 90-percentiles. See eg. `P1` in figure 3.2

As many features have datapoints outside the 10- and 90-percentile even within a single trial, the next step could be to categorize these datapoints as outliers. But inspection of some examples of specific features in specific trials, shows that it would be wrong to remove these "outliers". Eg. feature `V11` in trial 29 is plotted in figure 3.1 and seems to be perfectly normal. Careful inspection of the boxplot for feature `V11` in trial 29 (figure 3.2) shows that the feature has many outliers in this specific trial.

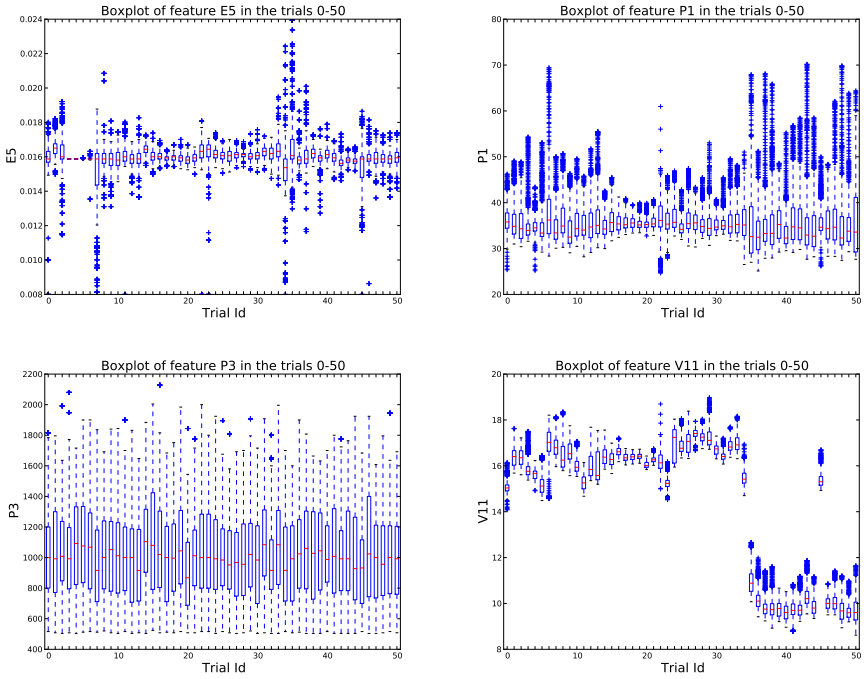
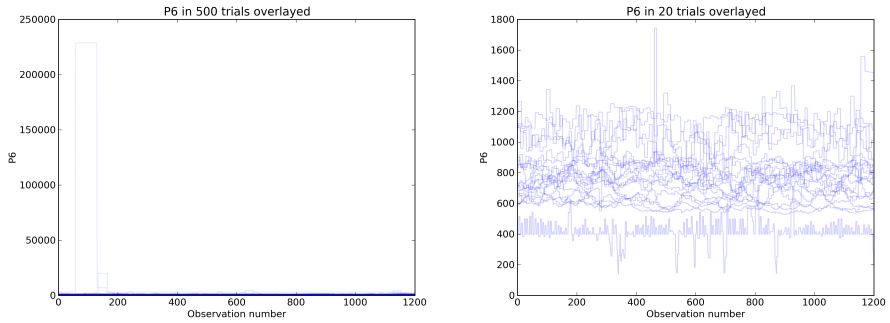


Figure 3.2: *Boxplots of feature E5, P1, P3 and V11 in the first 50 trials*



(a) **(b)**

Figure 3.3: *Layered feature plots of feature P6. All 500 trials are shown on the left. A couple of possible outliers are seen. On the right 20 randomly chosen trials are plotted*

The boxplots do not correctly identify the outliers in the dataset, and another method is therefore needed.

3.4.2 Making layered feature plots

Instead of trying to categorize single datapoints as outliers, it may make more sense to define a specific feature in a specific trial as an outlier. The fact that some features deviates a lot from their normal structure, in a few trials was already mentioned in section 3.3.2 about exploratory plotting of features. Since the exploratory plotting was a rather unstructered process, a more thorough picture, of the different features across all trials, is needed. One way to get a better picture is by plotting all trials of a single feature, in a single plot. The graph of each trial is given a low opacity and then all 400 trials in the trainingset are plotted on top of each other. That way the common structure of a feature across trials are visualized, and possible outlier trials can be detected. Two examples of this plot type can be seen in figure 3.3‡

‡Source code for the layered feature plots can be found in appendix B.7 and the results are in appendix B.8

From the layered feature plots it is confirmed that many features have some trials that seems to deviate from the normal pattern of the feature. Using feature P6 as an example, it is seen in figure 3.3a that P6 in almost all trials lies within the range [0,2000]. But one trial have a max value larger than 200000. This behaviour was hinted in section 3.2 and now there is a visual confirmation.

The question is, how should this “outlier" trial of feature P6 be handled? One approach is to exclude the trial from the dataset. But as nothing about the origin of the dataset is known, it can’t be justified with a commen sense argument, and if this trial is excluded then how is the line between outlier and normal drawn? There is a couple of other trials where the P6 feature has values as high as 20000, but should this be seen as outliers? And are there other ways a trial could be an outlier than the scale? In figure 3.3b most trials have the same structure, but the trial with the lowest mean has a noticeable different shape than the other trials. It somehow has a baseline at the value 400 and then oscillates at a higher frequency than the other trials. Is this an outlier? If it is regarded as an outlier many other trials could possibly be classified as outliers, and we’re are still only looking at one feature. If the same amount of cleaning is done for the other features, not much is left for the “normal" dataset.

The conclusion is that the outlier detection easily becomes, either too allencompassing or is done in an inconsistent manner. An alternative way to tackle the trials with datapoints with large values, could be to do some kind of variable transformation (eg. taking the log) of the feature.

3.4.3 Features that are constant in a trial

Another type of potential outlier trials, are all the trials where one or more features are constant throughout the trial. This could be a sign of a failure in the measurement equipment, and then maybe the trial should be regarded as an outlier? A great example is the IsAlert feature, that is the feature that should be predicted by the classifier. In 142 out of the 400 trials the IsAlert feature is constant and in 127 of the trials with a constant value, the driver isn’t alert. At first it may seem a bit odd, and indeed some of the contestants of The Ford Challenge revealed that they had

excluded all trials where the IsAlert feature was 0 throughout the trial (see David (2011)). But it is speculative to do, since the precise meaning of the IsAlert feature isn't known, let alone the fact that, measuring human alertness with a binary variable is speculative by itself.

As it seems too speculative to exclude features with constant values, this idea won't be pursued any further. Instead the focus changes from "finding features to get rid of" till "finding features that could be benificial for classification" as the next section is all about finding features that can discriminate between alert/not-alert drivers.

3.5 Finding possible discriminating features

In this section all features are examined in an attempt to find one or more features that are in some way correlated with the IsAlert feature that should be predicted.

3.5.1 Scatterplots

A great way to visualize whether a pair of features can be used to discriminate between two classes, is to create scatterplots of the possible pair of features. Since feature V7, V9 and P8 are removed and feature E3, E7, E8, E9, V3, V5 and V10 was found to be categorical, only 20 features are left for the scatterplots. Since the order of the two features in a scatterplot doesn't matter there is a total of

$$\binom{20}{2} = 190 \text{ combinations}$$

All combinations of the 20 features was plotted[§], but no interesting results was found; at least not regarding feature pairs that could classify the IsAlert feature. One interesting result was found though. As can be seen in figure 3.4 the feature pairs (P3, P4) and (P6, P7) both appear to have an arithmetic relationship. A quick calculation shows that P3 multiplied

[§]Source code can be found in appendix B.9 and the results in appendix B.10

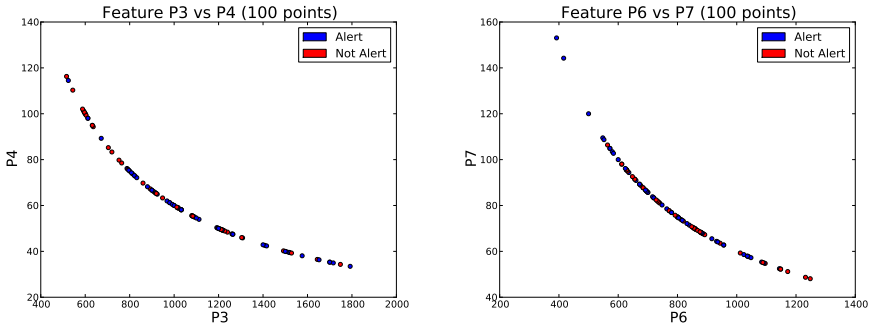


Figure 3.4: *Scatterplots of the feature pairs (P3, P4) and (P6, P7). An arithmetic relationship between the features is clearly present*

by P4 is 60000 ± 0.2 for all rows in the dataset, and likewise for P6 and P7. We can therefore ignore one of the features from each pair.

3.6 Principal Component Analysis

In this section the theory behind the Principal Component Analysis is introduced, and then a Principal Component Analysis is calculated on the whole dataset.

3.6.1 Theory[¶]

The idea in Principal Component Analysis (PCA) is to find a mapping from a d-dimensional input space, to a new k-dimensional space (with $k < d$), while keeping as much of the variation in the input data, as possible. To begin with let $k = 1$. Then the 1-dimensional subspace is identified by one d-dimensional basis vector $\boldsymbol{w}_1$. Without loss of generality it is assumed that $||\boldsymbol{w}_1|| = 1$.

[¶]Theory section is based on Alpaydin (2010, p.113)

For any vector $\boldsymbol{x}$ in the input space, the projection onto $\boldsymbol{w}_1$ is given by

$$z_1 = \boldsymbol{w}_1^T \boldsymbol{x}$$

and we now wish to maximize the variance of z_1. By the definition of variance we get (with $\boldsymbol{\mu} = \mathrm{E}[\boldsymbol{x}]$)

$$\begin{aligned}
\mathrm{Var}[z_1] &= \mathrm{Var}[\boldsymbol{w}_1^T \boldsymbol{x}] \\
&= \mathrm{E}[(\boldsymbol{w}_1^T \boldsymbol{x} - \boldsymbol{w}_1^T \boldsymbol{\mu})^2] \\
&= \mathrm{E}[(\boldsymbol{w}_1^T \boldsymbol{x} - \boldsymbol{w}_1^T \boldsymbol{\mu})(\boldsymbol{w}_1^T \boldsymbol{x} - \boldsymbol{w}_1^T \boldsymbol{\mu})] \\
&= \mathrm{E}[\boldsymbol{w}_1^T (\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{w}_1] \\
&= \boldsymbol{w}_1^T \mathrm{E}[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T] \boldsymbol{w}_1 \\
&= \boldsymbol{w}_1^T \Sigma \boldsymbol{w}_1
\end{aligned}$$

where $\Sigma = \mathrm{Cov}[\boldsymbol{x}]$. So maximizing the variance of z_1 is the same as maximizing $\boldsymbol{w}_1^T \Sigma \boldsymbol{w}_1$, subject to the constraint that $\boldsymbol{w}_1^T \boldsymbol{w}_1 = 1$. By introducing a Lagrange multiplier α, we get

$$\max_{\boldsymbol{w}_1} \boldsymbol{w}_1^T \Sigma \boldsymbol{w}_1 - \alpha(\boldsymbol{w}_1^T \boldsymbol{w}_1 - 1)$$

This is a normal, unconstrained maximization problem. Taking the derivative with respect to $\boldsymbol{w}_1$, and setting this equal to 0, gives (using that Σ is symmetric)

$$\begin{aligned}
2\Sigma \boldsymbol{w}_1 - 2\alpha \boldsymbol{w}_1 = 0 \qquad &\Leftrightarrow \\
\Sigma \boldsymbol{w}_1 = \alpha \boldsymbol{w}_1 &
\end{aligned}$$

which is true exactly when $\boldsymbol{w}_1$ is an eigenvector of Σ and α is the corresponding eigenvalue. When α is an eigenvalue for Σ, what needs to be maximized is

$$\mathrm{Var}[z_1] = \boldsymbol{w}_1^T \Sigma \boldsymbol{w}_1 = \alpha \boldsymbol{w}_1^T \boldsymbol{w}_1 = \alpha$$

and therefore the eigenvector with the largest eigenvalue is selected as $\boldsymbol{w}_1$ and is called the first principal component. The second principal component should also maximize variance, be of unit length and be orthogonal to $\boldsymbol{w}_1$. By introducing another Lagrange multiplier to handle the orthogonality constraint and maximizing the expression for maximum variance, it can be shown that the second principal component will be the eigenvector with the second highest eigenvalue. This procedure can be repeated and

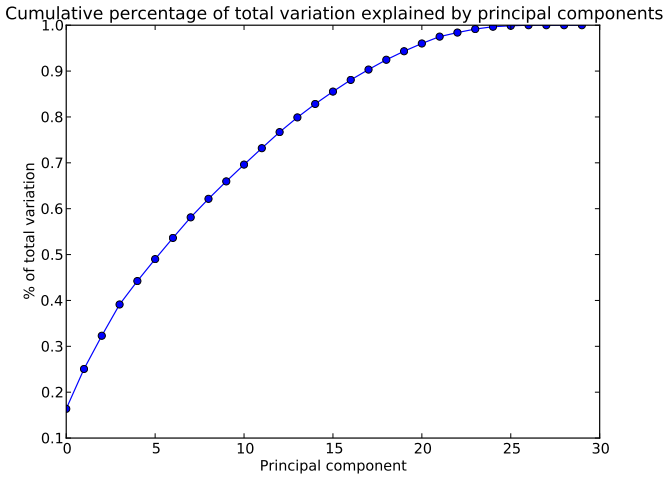


Figure 3.5: *sldkfj*

the end result is that the principal components are given as the eigenvectors of Σ, sorted by the value of the corresponding eigenvalues. It can further be shown that the proportion of the total variation explained by the first k principal components is given by

$$\frac{\lambda_1 + \lambda_2 + \cdots + \lambda_k}{\lambda_1 + \lambda_2 + \cdots + \lambda_k + \cdots + \lambda_d} \tag{3.1}$$

where λ_i is the eigenvalue of the i'th eigenvector sorted by the eigenvalue.

3.6.2 Result

A Pricipal Component Analysis was done on the dataset. First the data was normalized to avoid that the variance of some features dominated the total variance, and then a PCA was calculated for the whole dataset.**TODO: Source Code**. To get an overview of the pricipal components, a plot of the proportion of total variance as a function of the number of principal components, was plotted. The result can be seen in figure 3.5.

From the graph it is seen that 90% of the variance can be explained by the first 16-17 principal components. This means that 90% of the variance

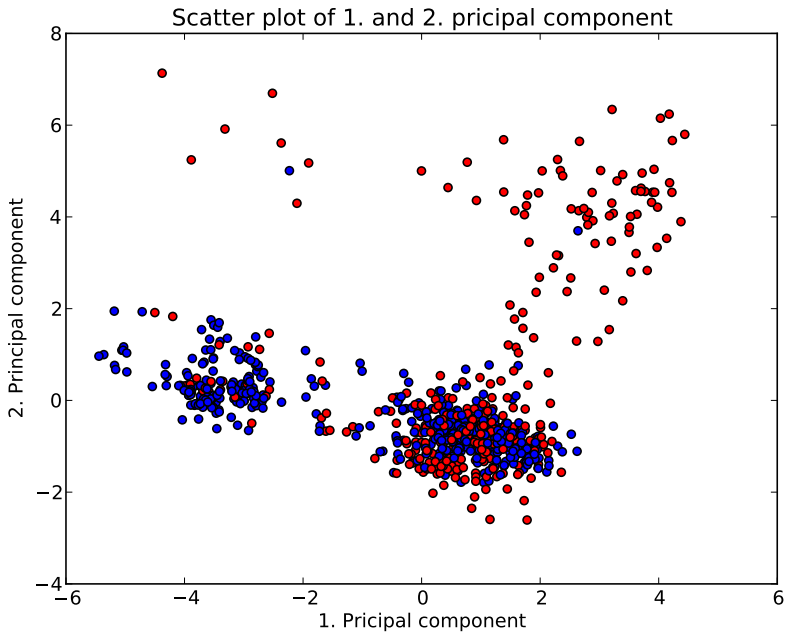


Figure 3.6: *sldkfj*

can be explained by a 16-dimensional dataset, instead of the original 30-dimensional dataset.

A scatter plot of the first two principal components and another scatter plot of the three first principal components was created, each having 800 randomly selected datapoints from the dataset. Each point in the scatter plots was colored to show the class (alert/not-alert) of the datapoint; with red indicating not-alert. The plots are shown in figure 3.6 and figure 3.7. It looks as if a 1. principal component above 0 and a 2. principal component above 2, implies that the datapoint is in the class not-alert. This wasn't investigated further though.

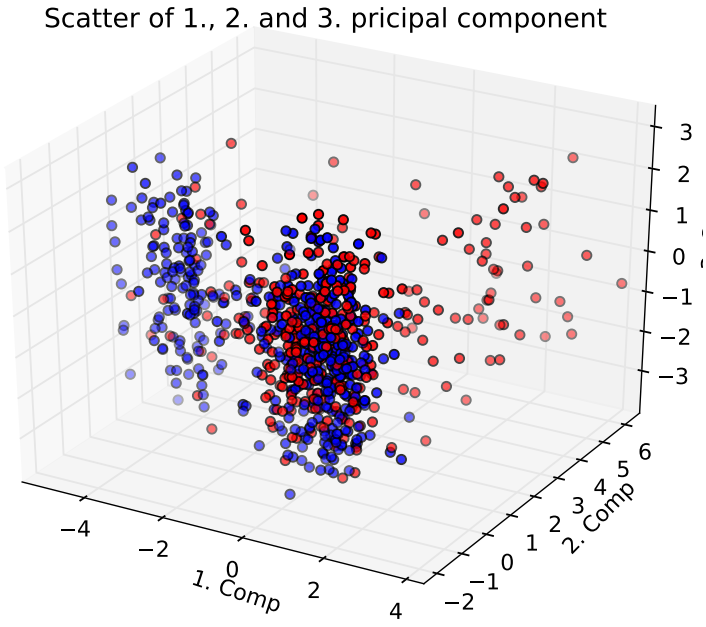


Figure 3.7: *sldkfj*

Chapter 4

Theory of classification

In this chapter some of the theory of classification is introduced. The discussion is constrained to binary classification, of which the Ford Challenge classifier, is an example. First a general problem definition is given and notation is introduced. Then two concrete examples of classification models are introduced (Logistic Regression and Neural Network), and similarities and differences between the methods are described. Finally a techniques to measure the performance of different classifiers are discussed, including AUC that was used as the grading method in The Ford Challenge.

4.1 The binary classification problem*

In a binary classification problem a binary outcome t must be predicted from a d-dimensional input vector $\boldsymbol{x} = [x_1, x_2, \ldots, x_d]^T$. The input vector

*This section is loosely based on Wasserman (2004, sec 22.1-22.2)

represents an event, and the outcome variable t represents the assignment of the event to one of two classes.

EXAMPLE 4.1 *In The Ford Challenge the input is an instant in a driving situation, and this input should be assigned to either the class alert, or the class not-alert. The input is represented by an 30-dimensional vector and the assignment to a class is represented by* $t = 0$ *meaning not-alert and* $t = 1$ *alert.*

To make the prediction possible, a trainingset $\mathcal{T}$ is given consisting of n pairs of input vectors and corresponding, known outcomes.

$$\mathcal{T} = \{(\boldsymbol{x}_i, t_i)\}, \quad i = 1 \ldots n$$

Based on this trainingset, a classification rule f is learned, that can predict the outcome, of yet unknown input vectors. Therefore a classification rule is a function $f : \mathbb{R}^d \to \{0, 1\}$, that is used to make the prediction $t = f(\boldsymbol{x})$ on a new input.

A common way to obtain a classification rule is by the Bayes classification rule. First define $p_k(\boldsymbol{x})$ as

$$p_k(\boldsymbol{x}) = P(t = k|\boldsymbol{x}), \quad k \in \{0, 1\}$$

and then the Bayes classification rule is defined by

DEFINITION 4.1 *The Bayes classification rule* f^* *is given by*

$$f^*(\boldsymbol{x}) = \begin{cases} 1 & \textit{if} \quad p_1(\boldsymbol{x}) > p_0(\boldsymbol{x}) \\ 0 & \textit{else} \end{cases}$$

Since $p_0(\boldsymbol{x}) = 1 - p_1(\boldsymbol{x})$, the Bayes classification rule can also be written

$$f^*(\boldsymbol{x}) = \begin{cases} 1 & \text{if} \quad p_1(\boldsymbol{x}) > \frac{1}{2} \\ 0 & \text{else} \end{cases}$$

By the Bayes classification rule we have got a theoretical classification rule. In practice the posterior class probabilities $p_0(\boldsymbol{x})$ and $p_1(\boldsymbol{x})$ are unknown, so the trainingset must be used to find some approximation for

these probabilities.[†] The problem of approximating a probability distribution from a given data set is a classic statistical problem. The next section gives one way to solve this problem.

4.2 Parametric models and maximum likelihood

Two distinct ways to approximate the probability distribution $P(t|\boldsymbol{x})$ is using either a parametric or a non-paramtric approach. The focus in this report is on the parametric approach. The distribution $P(t|\boldsymbol{x})$, is therefore approximated by a distribution $\widehat{P}(t|\boldsymbol{x})$ restricted to a class of distributions

$$\mathcal{F} = \left\{ f(t|\boldsymbol{x}, \boldsymbol{w}) \right\}$$

where the distributions $f \in \mathcal{F}$ are uniquely identified by their parameter $\boldsymbol{w} = [w_1, w_2, \ldots, w_k]^T$. By restricting the approximating distribution $\widehat{P}(t|\boldsymbol{x})$ to the set $\mathcal{F}$, the problem of approximating $P(t|\boldsymbol{x})$ reduces to the problem of estimating the parameter $\boldsymbol{w}$.

4.2.1 Maximum likelihood

There exists some different techniques to estimate the parameter $\boldsymbol{w}$ in a parametric model.[‡] In this section we look at the most common method for estimation of the parameter in a parametric model, namely the maximum likelihood method.

The approximation $\widehat{P}(t|\boldsymbol{x})$ of the posterior class probability is restricted to the class $\mathcal{F}$. Using a trainingset, the likelihood function is now defined as the probability of obtaining the data in the trainingset, given as a function of the parameter $\boldsymbol{w}$. More formally[§]:

DEFINITION 4.2 *The likelihood function is defined by*

$$\mathcal{L}(\boldsymbol{w}) = \prod_{i=1}^{n} f(t_i|\boldsymbol{x}_i, \boldsymbol{w}), \quad (t_i, \boldsymbol{x}_i) \in \mathcal{T}$$

[†]Since $p_0(\boldsymbol{x}) = 1 - p_1(\boldsymbol{x})$ only one of the probabilities actually needs to be estimated

[‡]See eg. Wasserman (2004, Sec.9)

[§]Taken directly from Wasserman (2004, p.122)

and the log-likelihood function is defined by $\ell(\boldsymbol{w}) = \log \mathcal{L}(\boldsymbol{w})$.

It is worth noticing that the definition of the likelihood function assumes that the t_i's are independent random variables. An assumption that in the case of The Ford Challenge may be a bit questionable. This is due to the sequential nature of the Ford Challenge dataset, which makes it very likely that datapoints that are close to each other in time, are not independent.

Given the definition of the likelihood function, the maximum likelihood estimator is now defined by

DEFINITION 4.3 *The maximum likelihood estimator* $\widehat{\boldsymbol{w}}$ *is the value of* $\boldsymbol{w}$ *that maximizes* $\mathcal{L}(\boldsymbol{w})$.

Notice that as the logarithm is a monotonic increasing function, the maximum of $\mathcal{L}(\boldsymbol{w})$ is equal to the maximum of $\ell(\boldsymbol{w})$. This turns out to be handy, since the log-likelihood is often easier to work with than the likelihood function.

With these definitions it is now time to introduce two concrete examples of classification methods.

4.3 Logistic Regression

In the logistic regression the parametric form of the posterior class probability $P(t = 0|\boldsymbol{x})$ is assumed to be

$$P(t = 0|\boldsymbol{x}) = \frac{1}{1 + e^{-(\boldsymbol{w}^T \boldsymbol{x} + w_0)}} \tag{4.1}$$

where $\boldsymbol{w} = [w_1, w_2, \ldots, w_d]^T$. The function

$$\sigma(a) = \frac{1}{1 + e^{-a}} \tag{4.2}$$

is called the sigmoid function and it will be used a number of times in the following sections. For later convenience it is noted that

$$\frac{d\sigma}{da} = \sigma(1-\sigma) \tag{4.3}$$

The fact that $P(t=0|\boldsymbol{x})$ is modelled by the sigmoid function, may seem a bit random, but comes from the fact that (with $p_0 = P(t=0|\boldsymbol{x})$)

$$p_0 = \frac{1}{1+e^{-(\boldsymbol{w}^T\boldsymbol{x}+w_0)}}$$

can be written

$$\begin{aligned} p_0 &= \frac{e^{\boldsymbol{w}^T\boldsymbol{x}+w_0}}{1+e^{\boldsymbol{w}^T\boldsymbol{x}+w_0}} && \Leftrightarrow \\ p_0 + p_0 e^{\boldsymbol{w}^T\boldsymbol{x}+w_0} &= e^{\boldsymbol{w}^T\boldsymbol{x}+w_0} && \Leftrightarrow \\ \frac{p_0}{1-p_0} &= e^{\boldsymbol{w}^T\boldsymbol{x}+w_0} && \Leftrightarrow \\ \log\frac{p_0}{1-p_0} &= \boldsymbol{w}^T\boldsymbol{x}+w_0 \end{aligned}$$

And therefore the parametric form in (4.1), comes from the assumption that the log-odds of p_0 is linear in the input vector $\boldsymbol{x}$.

Now to find an expression for the likelihood function, the probability of getting the data in the trainingset must be expressed. For a given $\boldsymbol{x}$ the class will be class 0 with probability $P(t=0|\boldsymbol{x})$. If class 0 is defined as a "success" then t given $\boldsymbol{x}$ is Bernoulli distributed with probability parameter $p = P(t=0|\boldsymbol{x})$.

With $p_i = P(t=0|\boldsymbol{x}_i)$ the likelihood function can be written as

$$\mathcal{L}(\boldsymbol{w}) = \prod_{i=1}^{n} p_i^{t_i}(1-p_i)^{1-t_i}$$

where p_i is implicit a function of $\boldsymbol{w}$. The log-likelihood function is therefore given by

$$\ell(\boldsymbol{w}) = \sum_{i=1}^{n} t_i \log p_i + (1-t_i)\log(1-p_i) \tag{4.4}$$

The maximum of the log-likelihood function gives the maximum likelihood estimator of $\boldsymbol{w}$, so the derivative of ℓ wrt. $\boldsymbol{w}$ needs to be derived. From equation (4.3) it is seen that

$$\frac{\partial p_i(\boldsymbol{w})}{\partial \boldsymbol{w}} = p_i(1-p_i)\boldsymbol{x}_i \tag{4.5}$$

and therefore

$$\begin{aligned}\frac{\partial \ell(\boldsymbol{w})}{\partial \boldsymbol{w}} &= \sum_{i=1}^{n} t_i \frac{1}{p_i} p_i(1-p_i)\boldsymbol{x}_i + (1-t_i)\frac{1}{1-p_i}(-p_i)(1-p_i)\boldsymbol{x}_i \\ &= \sum_{i=1}^{n} t_i(1-p_i)\boldsymbol{x}_i - (1-t_i)p_i\boldsymbol{x}_i \\ &= \sum_{i=1}^{n} (t_i - p_i)\boldsymbol{x}_i\end{aligned}$$

To make sure it is a maximum the second derivative wrt. $\boldsymbol{w}$ must be found. Using (4.5), the Hessian of the log-likelihood is found as

$$\boldsymbol{H} = \frac{\partial^2 \ell(\boldsymbol{w})}{\partial \boldsymbol{w} \partial \boldsymbol{w}^T} = -\sum_{i=1}^{n} p_i(1-p_i)\boldsymbol{x}_i \boldsymbol{x}_i^T$$

which can be rewritten as an matrix equation, by introducing the matrices $\boldsymbol{X}$ and $\boldsymbol{P}$. $\boldsymbol{X}$ is an $n \times d$ matrix with the i'th row being $\boldsymbol{x}_i$ and $\boldsymbol{P}$ is an $n \times n$ diagonal matrix with $p_i(1-p_i)$ in the diagonal. The Hessian can then be written

$$\boldsymbol{H} = \boldsymbol{X}^T \boldsymbol{P} \boldsymbol{X}$$

For any non-zero d-dimensional column vector $\boldsymbol{u}$, let $\boldsymbol{u}_x = \boldsymbol{X}\boldsymbol{u}$. Then $\boldsymbol{u}_x^T = \boldsymbol{u}^T \boldsymbol{X}^T$ and therefore

$$\boldsymbol{u}^T \boldsymbol{H} \boldsymbol{u} = \boldsymbol{u}_x^T \boldsymbol{P} \boldsymbol{u}_x$$

Since $0 < p_i(1-p_i) < 1$, $\boldsymbol{P}$ is a diagonal matrix with positive elements, so $\boldsymbol{u}_x^T \boldsymbol{P} \boldsymbol{u}_x = \boldsymbol{u}^T \boldsymbol{H} \boldsymbol{u} > 0$ for any vector $\boldsymbol{u}$. Therefore the Hessian is positive definite and then $-\boldsymbol{H}$ is negative definite. From this it is seen that it is indeed a maximum that is found and that it is a concave optimization problem, so a single global maximum exists.

The maximum likelihood estimator can now be found by solving $\frac{\partial \ell(\boldsymbol{w})}{\partial \boldsymbol{w}} = 0$, but since p_i isn't linear in $\boldsymbol{w}$ the equation must be solved numerically. An effective algorithm called iterative reweighted least squares can solve this problem (Bishop, 2006, p.207).

4.3.1 The decision boundary

An important property of a classification method is how complex, the datasets the method can separate in the appropriate classes, can be. The decision boundary of a classification method gives some imformation about how well it separates input data. The decision boundary is the set of points in the input space that lays at the boundary between input points that are classified as class 0 and input points that classify as class 1.

EXAMPLE 4.2 *A trainingset consists of n d-dimensional input data* $\boldsymbol{x}_i = [x_{1i}, x_{2i}, \ldots, x_{di}]^T$ *and corresponding classes* t_i*. Using logistic regression, approximations of the posterior class probabilities* $\widehat{p}_k(\boldsymbol{x}) = \widehat{P}(t = k|\boldsymbol{x}), k \in \{0,1\}$ *are found. Using these with the Bayes classification rule gives*

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if} \quad \widehat{p}_1(\boldsymbol{x}) > \widehat{p}_0(\boldsymbol{x}) \\ 0 & \text{else} \end{cases}$$

The decision boundary is given by all $\boldsymbol{x}$ *where* $\widehat{p}_0(\boldsymbol{x}) = \widehat{p}_1(\boldsymbol{x})$*. But by the definition of the linear regression*

$$\log \frac{\widehat{p}_1(\boldsymbol{x})}{\widehat{p}_0(\boldsymbol{x})} = \boldsymbol{w}^T \boldsymbol{x} + w_0$$

So for all points at the decision boundary the following holds

$$\boldsymbol{w}^T \boldsymbol{x} + w_0 = 0$$

This shows that the decision boundary of the logistic regression is linear in the inputs.

The classification methods with linear decision boundaries, defines a family of classification methods with simple analytical and computational properties (Bishop, 2006, p.179). This simplicity comes at the cost of the models having a limited capability to separate complex data sets.

4.3.2 Model complexity

Although the logistic regression has a limited complexity, its ability to separate complex datasets can be increased by introducing feature transformations. Instead of just fitting the logistic regression to the d-dimensional

input $\boldsymbol{x} = [x_1, x_2, \ldots, x_d]$, eg. the squared transformations of the elements x_i can be added to the input. This gives the $2d$-dimensional input

$$\boldsymbol{x}_e = [x_1, \ldots, x_d, x_1^2, \ldots, x_d^2]$$

This gives nonlinear decision boundaries in the original input space, which increases the number of datasets that can be separated. Notice that the decision boundary is still linear in the extended $2d$-dimensional input space. Instead of introducing transformed, non-linear features in the logistic regression, other methods exists, that have the non-linearity build-in. One example of such a model is the Neural Network.

4.4 The Neural Network

The Neural Network originated from an attempt to create a mathematical model of information processing in the human brain (Bishop, 2006, p.226). But the Neural Network turned out to be useful in many pratical applications, and it is now seen as a standard statistical model (Hastie et al., 2009, p.392).

A neural network is made up of one or more layers and each layer consists of one or more perceptrons. The perceptron is the basic building block of a neural network and it is described next

4.4.1 Perceptron

A perceptron consists of one or more inputs represented by an input vector $\boldsymbol{x}$ with corresponding weights $\boldsymbol{w}$ and an activation function f. The activation function is assumed to be differentiable. The output from the perceptron is then

$$y = f(\boldsymbol{w}^T \boldsymbol{x} + w_0)$$

The intercept value w_0 is added to make the model more flexible. It can be seen as the weight of an extra input that is always 1. As a stand alone unit, the perceptron can implement linear classification methods, as eg. logistic regression. To emulate the behavior of a logistic regression, the activation

function is chosen as the sigmoid from equation (4.2). The output of the perceptron is modelled as the posterior class probability $P(t = 0|\boldsymbol{x})$, which gives

$$P(t = 0|\boldsymbol{x}) = \frac{1}{1 + e^{-(\boldsymbol{w}^T \boldsymbol{x} + w_0)}}$$

Compared with the logistic regression model in equation (4.1) it is seen that the two are identical. An error function can now be defined exactly as the log-likelihood in the logistic regression (see equation (4.4)) and a maximizing weight vector can be found.

The perceptron by itself isn't much more complex than the logistic regression, and the decision boundary is still linear in the input space. The non-linearity is achieved when multiple perceptrons are combined in a network.

4.4.2 Neural Network for binary classification

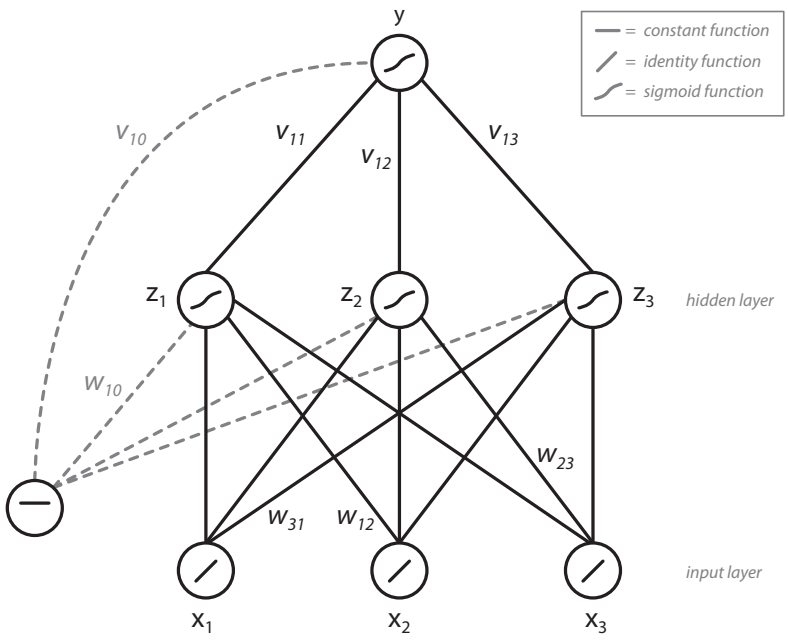


Figure 4.1: *Neural network*

In a neural network multiple perceptrons are combined, such that the output of one perceptron becomes the input of another perceptron. A lot of different network architectures exists but here the focus will be on feedforward networks with one hidden layer.

In a feedforward network the perceptrons can be divided into layers where the output of all perceptrons goes to a perceptron in the layer "above". The first layer is called the input layer and it consists of $D+1$ perceptrons, one for each feature in the input data, plus an intercept perceptron. The output of the input layer becomes input in the hidden layer, and each such connection has attached a weight w_{hi}. The hidden layer have $H+1$ perceptrons including an intercept perceptron. The output from the hidden layer, then becomes input to the output layer and each connection also has a weight v_{1h}. The output of the i'th perceptron in the input layer is x_i. The output of the h'th perceptron in the hidden layer is z_h and is given by

$$z_h = \sigma(\boldsymbol{w}_h^T \boldsymbol{x} + w_{h0}) = \frac{1}{1+e^{-(\boldsymbol{w}_h^T \boldsymbol{x} + w_{h0})}}$$

assuming that the sigmoid function is used as activation function in the hidden layer. This is a common choice as activation function in the hidden layer, but other functions could have been used. The output of the hidden layer is sent to the output layer. Since the focus is on binary classification, the output layer consists of only one perceptron y. Since the output is modelled as the conditional probability $P(t=0|\boldsymbol{x})$, the sigmoid is also used as activation function for the output layer. The output can therefore be written

$$y = \sigma(\boldsymbol{v}_1^T \boldsymbol{z} + v_{10}) = \frac{1}{1+e^{-(\boldsymbol{v}_1^T \boldsymbol{z} + v_{10})}}$$

From these expression it may be seen that the decision boundary is no longer linear in the output. To get a better sense for this, the output can also be written

$$y = \sigma\left(\sum_{h=1}^{H} v_{1h}\, \sigma\left(\sum_{i=1}^{D} w_{hi} x_i + w_{h0}\right) + v_{10}\right)$$

The increased complexity of the neural network comes at a price though. Finding the weights that maximizes the log-likelihood is no longer a concave problem, in fact many maxima exists[¶], and there are no guarantees

[¶]See Bishop (2006, p.237) for a description of how many, and Duda et al. (2001, p.297) for a couple of graphs of the likelihood function

that the global maximum will be found. Fortunately an efficient algorithm called backpropagation[‖] exists, that efficiently finds at least a local maximum.

4.4.3 Model complexity

In section 4.3.2 it was described how the complexity of the logistic regression model, could be increased by adding transformed features. This can also be done in a neural network, but some other options exists as well. First of all different activation functions can be chosen for the different layers. In binary classification it is common to use the sigmoid function in the output layer, but the activation functions in the hidden layer can be freely chosen, and this may affect the complexity of the model. Another option is to increase the number of perceptrons in the hidden layer. As more perceptrons are added, the complexity rises with it. In fact it can be shown that any continous function can be approximated arbitrarily well by a network with one hidden layer, as long as enough perceptrons are added (see Duda et al. (2001, p.287)).

4.5 Measuring classifier performance

When a classifier has been trained on a training dataset, some estimate of the classifier performance, on future datasets, needs to be estimated. To be able to estimate the performance, it first needs to be defined exactly how the performance is measured. This can be done in many ways. For the Ford Challenge a measure called AUC-score was used.

4.5.1 AUC**

The AUC-score is the area under the Receiver Operating Characteristic (ROC) curve for a classifier, so to explain the AUC-score, an introduction

[‖] See Alpaydin (2010, p.249) for an explanation of the backpropagation algorithm.

** This section is loosely based on Tan et al. (2006, p.296-301)

of the ROC curve is needed. The ROC curve is based on four measures of the performance of a binary classifier. Assume that one of the two classes of the classifier have been labeled as “positive”. Then the four measures are:

- True positives (TP) - The number of positive samples that was correctly classified as positive.
- False negative (FN) - The number of positive samples that was incorrectly classified as negative.
- False positive (FP) - The number of negative samples that was incorrectly classified as positive.
- True negative (TN) - The number of negative samples that was correctly classified as negative.

From these four measures it is easy to calculate the true positive rate (TPR) and false positive rate (FPR) as

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad \text{and} \qquad \text{FPR} = \frac{\text{FN}}{\text{TN} + \text{FP}}$$

Based on the TPR and FPR measures an ROC curve can be calculated.

4.5.1.1 Generating the ROC curve

The ROC curve assumes that the classifier outputs continous-valued output and that the ordering of the output corresponds to the classifiers ranking of the inputs connection with the “positive” class. This is exactly the case when an estimate of the posterior class probability $P(t = 0|\boldsymbol{x})$ is the output of a classifier.

To generate the ROC curve for a classifier on a specific trainingset, all rows in the trainingset are predicted by the classifier. Then the rows are ordered by these predictions. A threshold parameter is set equal to the highest prediction and all rows with a prediction lower than or equal to the threshold is predicted to belong to the negative class. For this threshold all

rows are predicted as as negative, and therefore both TPR=0 and FPR=0 and therefore the point (0,0) is plotted on the ROC curve. The threshold is now set as the next lowest prediction and TPR and FPR are recalculated and the point (FPR, TPR) is plotted on the ROC curve. This is repeated until the threshold is equal to the lowest prediction, at which point all rows are classified as positive, and therefore the point (1,1) is plotted. The curve generated is the ROC curve which plots the relationship between FPR and TPR, as the threshold varies from lowest to highest prediction.

Two extreme examples will clarify what a ROC curve for a good classifier and a bad classifier looks like. If a classifier predicts all rows correctly it will rank all positive rows higher than the negative. As the threshold is lowered the TPR will rise to 1, while the FPR remains at 0. When TPR have reached 1, the FPR starts to go towards 1. The ROC curve for a perfect classifier therefore starts at (0,0), steeply rises to (0,1) and then goes to (1,1).

If the classifier on the other hand is just randomly making prediction the behaviour is different. When the ROC curve is generated the threshold at any time during the generation divides the dataset such that $p \cdot 100\%$ of the dataset is classified as positive. For a classifier that makes random prediction on a dataset with n_+ positive rows and n_- negative rows, it would be expected that pn_+ rows will be classified positive and pn_- as negative. This gives TPR= $(pn_+)/n_+ = p$ and FPR= $(pn_-)/n_- = p$. So for all thresholds a point near (p, p) is expected and as p is updated every time the threshold is updated, the result is the straight line from $(0,0)$ to $(1,1)$.

4.5.1.2 Good and bad AUC scores

Based on the observations in the previous paragraph it is seen that good classifiers have an area under the ROC curve close to 1, while bad classifiers have areas around 0.5. This is used to measure the performance of different classifiers. When a classfier has been trained it makes prediction on a testset and the area under the ROC curve is calculated, and this is the AUC score of the classifier.

Some properties of the AUC score makes it appealing to use. It is a single number and no parameters are chosen by the person calculating the AUC, which means that different people, calculating the AUC of the same classifier on the same dataset should get the same value. Some problems with the AUC score exists though.

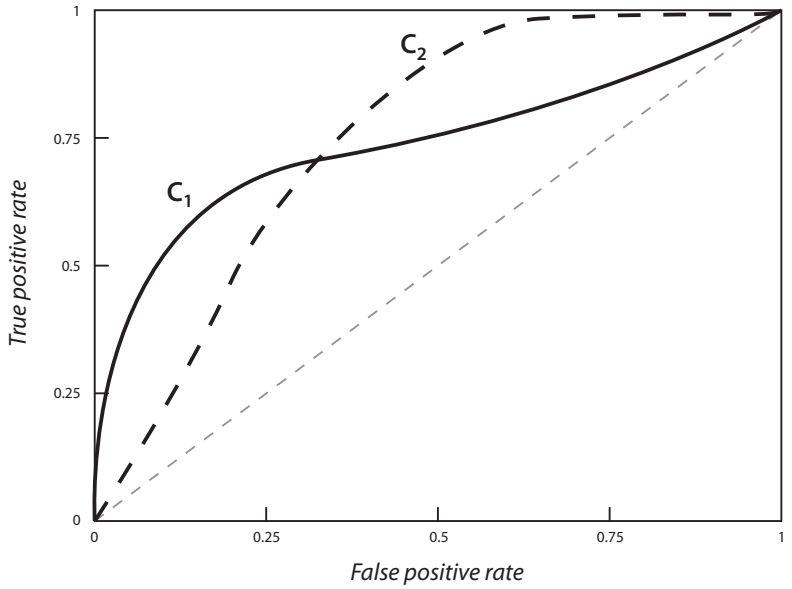


Figure 4.2: *Some examples of ROC curves. Although C_2 achieves a slightly higher AUC, C_1 may be prefered as discussed section 4.5.2*

4.5.2 Critiques of AUC

The AUC for a classifier is calculated by letting the threshold value change from the lowest to the highest prediction. When a final model is chosen this threshold is set to a specific value, which means that the AUC score is influenced by behavior of the classifier that may never be of pratical interest. Take for example the ROC curves in figure 4.2. The classifier C_2 achieves the highest AUC, but C_2 performs worse than C_1 when FPR< 0.35. In a pratical application a false positive rate above 0.35 may be totally unacceptable and then C_1 would be the prefered classifier. One

obvious solution to this problem would be to only calculate the AUC for FPR< 0.35, but this introduces a parameter that must be subjectively chosen, thus losing one of the advantages of the AUC.

A more profound critique of the AUC score shows that the relative cost of making a wrong positive prediction compared with the cost of a wrong negative prediction, depends on which classifier method the AUC is calculated for. See the paper Hand (2009) for an indepth discussion.

The competition winner also mentions that the AUC score might be inapropriate for The Ford Challenge (Inference, 2011a, p.5)

4.5.3 Cross validation

When a measure has been chosen there still needs to be found a way to estimate the expected measure-score on future datasets, for the classifier. An emperical way to do this, is called cross validation.

One straight forward way to get an estimate of the expected performance on future datasets for a classification method, would be to measure the classifiers performance on the dataset used to train the classifier. The problem is that this will give an overly optimistic estimate, since the classifier has been trained to take into account some of the inevitable noise in the trainingset. To get a better estimate of the classifier performance, the complete dataset is split into smaller dataset, such that the same dataset is never used for both training and estimation of performance. This is the core idea in cross validation. Split the dataset to avoid using the same dataset for training and estimation.

The easiest way to split a dataset is to split it in two parts: a trainingset and a validationset. The classifier can then be trained on the traningset and its performance is estimated on the validationset. Although this technique works it use the dataset only once, and only one estimate of the performance is produced. This means that the variability in the estimate can't be estimated. If the validationset is large, it can be split in smaller subsets, and the performance can then be tested on each subset. This is the procedure used for this report, since the competition dataset had

plenty of datarows. If only a small dataset exists a set of estimates can still be calculated by using K-fold cross validation.

4.5.3.1 K-fold cross validation

In K-fold cross validation the dataset is split into K-parts. Then each part is selected as validationset once and the classifier is then trained on the $K-1$ remaining parts. This way K values of the classifier performance is calculated, and these can be used to both estimate the expected classifier performance and the variablility in this estimate. It is worth noticing that this is not exactly the same as splitting a large validationset and calculating performance on each part. In that procedure only one model is trained on the validationset, while K different models are trained in K-fold cross validation.

4.5.4 Confidence interval of estimated performance

When k measures of the performance of a model have been calculated, a confidence interval for the true value of the expected performance can be found. The k measures are seen as a sample from the population of performance scores on future datasets, and this population is assumed normal distributed. For a small sample ($k < 30$) with sample mean m and sample standard deviation s, a 95% confidence interval for the true mean μ can be found as[††]

$$m - t_{0.025,k-1} \cdot \frac{s}{\sqrt{n}} < \mu < m + t_{0.025,k-1} \cdot \frac{s}{\sqrt{n}}$$

where the value of $t_{0.025,k-1}$ can be found in a statistical table or in software as eg. R. In this report the sample size will be $k = 10$, which gives $t_{0.025,9} = 2.2622$.

[††]Section based on Johnson (2005, p.232-233)

4.6 Model selection

As mentioned in the section about logistic regression and neural networks, the complexity of a model can be tweaked in different ways. A natural question is how complex the model should be chosen. Should the complexity be as high as is computational possible? Or should the complexity be kept as low as possible? As is often the case, the answer is that the complexity should be somewhere between low and high, which is explained by the concept of over- and underfitting.

If the model complexity is made as high as possible, the model will probably suffer from what is called overfitting. This means that the model has learned so many details from the training data, that it has adapted to the noise in the training data. Although this gives a great performance on the trainingset, it also means that the model will perform worse on future datasets. That is, the generalization of the model gets worse, when the fit to the trainingdata gets too high[‡‡].

At the opposite end of the scale is a model that is too simple. When a model is to simple it can't adapt to the patterns in the training dataset, that is shared with the future dataset. That way the model will neither fit the trainingset nor any future datasets.

One way to determine the right level of complexity is by using cross validation. By training models at different complexity levels and comparing their estimated performance, the right complexity can be chosen for the model.

4.6.1 Forward feature selection

One way to tweak model complexity, that is independent of the classification model used, is feature selection. In feature selection a subset of the whole set of features is selected for the classification model. Since the performance can depend on combinations of features, all possible sub-

[‡‡]See Hastie et al. (2009, p.220) for a good illustration of overfitting.

set of features should ideally be tested to find the optimal subset. For a high-dimensional input space this is not possible.

EXAMPLE 4.3 *For the Ford Challenge the input data is 30-dimensional. This gives* $2^{30} \approx 10^9$ *unique subsets to test. Since the model must be trained for each unique subset this is not a feasible solution.*

Instead of trying every combination of features, different heuristics exists, that find a solution that is better than many alternatives. But optimality isn't guaranteed. One such heuristic is the forward feature selection. The procedure is simple. Start with a model that consists of no features and a set of candidate features that could be included in the model. Add the first candidate to the model and test performance. Then exchange the first candidate with the next and test performance, etc. The candidate that gives the best performance is removed from the candidate set and permanently added to the model. Recursively apply this procedure until the performance is no longer improved by adding features.

The forward feature selection is a greedy algorithm that at each stage selects the feature that gives the largest performance improvement. This doesn't guarantee an optimal performance but the worst case number of model training is (with d features in the candidate set)

$$\sum_{i=1}^{d} i = \frac{d(d-1)}{2}$$

The algorithmic complexity is therefore reduced from exponential $O(2^d)$ to polynomial $O(d^2)$.

Chapter 5

Recreating winning approach

In this chapter, the classification method of the Ford Challenge winner will be attempted to be recreated. The goal is to get a result – as measured by the AUC – that is close to the winners approach.

5.1 The winning approach

The competition winner has described his approach in Inference (2011a), which will just be called "the paper" in the rest of this section.

In the paper two key observations are mentioned. First of all, the winner observed that in most of the trials, the driver was either alert most of the time, or not alert for most of the time. Secondly the winner observed that there was a mismatch between the AUC-scores achieved on the Ford trainingset and on the Ford testset. Since it wasn't possible to get access to the Ford testset, the second observation isn't of any direct use for this report. The two observations though, gives an explanation of why the winner chose the model he did.

Since most trials was either alert-trials or not-alert-trials, the winner thought it would make sense to calculate aggregated features within trials. He therefore included the mean and the standard deviation of all features, aggregated backwards, within a trial. As a standard the mean features was prefixed "m" and the standard deviation features "sd".

EXAMPLE 5.1 *In the k'th observation of trial t, the value of feature* `mP6`*, is the mean of feature* `P6`*, for observation* $0,1,2,\ldots,k-1$ *in trial t*

The fact that there was a mismatch between the Ford trainingset and the Ford testset, made the winner focus on simple models, since he expected that the trainingset wasn't capturing all aspects of the testset, and extrapolation was therefore needed.

5.1.1 Classification method

The method used by the winner was a logistic regression, trained on the three features `sdE5`, `V11` and `E9`. The three features was selected "... based on diagnostics of the logistic regression ..." (Inference, 2011a, p.3). The final model was a logistic regression with the following weights

$$\log \frac{P(t=0|\boldsymbol{x})}{P(t=1|\boldsymbol{x})} = -392.4317 \cdot \mathrm{sdE5} + 0.2209 \cdot \mathrm{V11} + 3.6544 \cdot \mathrm{E9}$$

The winner do not mention any intercept parameter, although it seems unlikely that no intercept was included in the model (Mørup, 2011). Based on the above parameters, the winner achieved an AUC-score of 0.8492 on the Ford testset. As mentioned in the previous section, the AUC-score of the Ford testset differed from the AUC-score on the Ford trainingset. In figure 5.1 (a reproduction of a figure from Inference (2011a)) it is seen that an AUC-score of 0.8492 on the Ford testset, corresponded to an AUC-score of 0.82 ± 0.01 on the Ford trainingset. Since only the Ford trainingset is available for this report, the goal is to acieve an AUC-score of approximately 0.82 ± 0.01.

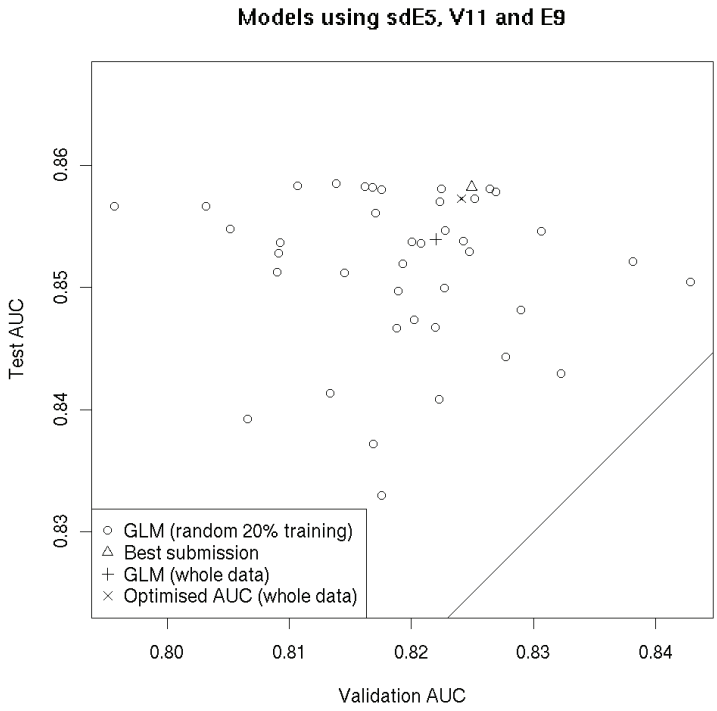


Figure 5.1: *Reproduction of figure 2 in Inference (2011a). The models that achieves a Test AUC higher than 0.8492 are using future observations to predict the* `IsAlert` *feature, which wasn't allowed in the competition. See Inference (2011a) for more information.*

Run	AUC
1	0.8082
2	0.8050
3	0.8114
4	0.8088
5	0.8063
6	0.8020
7	0.8128
8	0.8116
9	0.8057
10	0.8104

Table 5.1: *Results of calculating the AUC-score for the model in equation (5.1), for 10 parts of the testset.*

5.2 Recreating the winning approach

It is now time to recreate the logistic regression of the winner. Using the same features as the winner a logistic regression is trained on the trainingset. This resulted in the following model

$$\log \frac{P(t=0|\boldsymbol{x})}{P(t=1|\boldsymbol{x})} = -87.2753 \cdot \text{sdE5} + 0.2244 \cdot \text{V11} + 3.6545 \cdot \text{E9} - 5.3645 \quad (5.1)$$

The model is much like the winning model. The highest deviation from the winning model is in the weight of feature `sdE5`. To make it possible to calculate a confidence interval for the AUC-score of the new model, the testset is split in 10 parts, and an AUC-score is calculated for each part. The results can be seen in table 5.1 and the source code used to calculate the results can be found in appendix B.11

From the results table it is seen that the AUC-score achieved by the model lays in the interval [0.805;0.813] which is very close to the goal of 0.82 ± 0.1. To get a better idea of the results of the model, summary statistics and a confidence interval are calculated.

5.2.1 Statistics on the AUC score

As described in section 4.5.4 a 95% confidence interval can be calculated from the sample mean and standard deviation. These are found as

$$m = 0.8082 \qquad \text{and} \qquad s = 0.003456$$

which gives the confidence interval

$$\begin{aligned} 95\%\,\text{CI} &= [0.8082 - 2.2622\frac{0.003456}{\sqrt{10}}, 0.8082 + 2.2622\frac{0.003456}{\sqrt{10}}] \\ &= [0.8058, 0.8107] \end{aligned}$$

that should be interpreted as

$$\mathbb{P}(\mu_{AUC} \in [0.8058, 0.8107]) = 95\%$$

CHAPTER 6

Improving the winning approach

In this chapter it is examined whether the winning approach can be improved or not. The classification method will still be logistic regression, but it is tried to improve the results, by doing feature selection across all features. Two feature selection techniques are tried. First a forward selection is performed, on a subset of the whole trainingset, and then a L^1 regularization.

6.1 Forward selection

TODO: Mention that it isn't standard to use AUC as error function for Forward selection A forward selection is performed to find the features that gives the best classification. The logistic regression model starts out with no features, and then the feature that improves the performance the most is added to the regression model. This is repeated until the performance is no longer increased. The features, that the forward

Feature added	AUC
V11	0.6978
E9	0.8089
sdE1	0.8333
mE6	0.8500
sdE4	0.8539
⋮	⋮
mP1	0.8858
sdE5	0.8857

Table 6.1: *The first five and the last two features added in the forward selection.*

selection chooses between are all the features; both the original, and the running mean and running standard deviation. This gives a total of 90 features to choose between. Since all candidates for features to add to the model, must be tested in every iteration, a worst case scenario is that a total of

$$\sum_{i=1}^{90} i = \frac{90 * 89}{2} = 4005$$

logistic regression fittings must be done. Since a standard library (scikits.learn) is used for the training, the training dataset is restricted to 10000 rows and the validation dataset to 5000 rows. And only a simple hold-out cross validation is performed. The script for the forward selection can be seen in appendix B.12 and all the results are in appendix B.13. The first five selected features and the last two are shown in table **??** A total of 48 features out of the 90 features was selected. The AUC that is achieved, when all 48 features are in the model is quite high (0.8857). The 10000 datarows for the trainingset and the 5000 datarows of the validationset was chosen uniformly across the whole trainingset**TODO: Find good names for the different datasets**, so there is no reason to believe that any connection between the trainingset and the validationset, makes the AUC score artificially high. It is worth noticing that the top two features (`V11`, `E9`) are also used in the winning model. To get a better idea of the precise performance of the model with the 48 selected features, the model is now trained on the trainingset and the performance is tested on the testset.

Run	AUC
1	0.8914
2	0.8956
3	0.8985
4	0.8931
5	0.8951
6	0.8876
7	0.8958
8	0.8930
9	0.8963
10	0.8894

Table 6.2: *Results from testing the model, with 48 features, on 10 different parts of the testset.*

6.1.1 Testing performance on the original testset

By forward selection a model with 48 features have been selected. The AUC score achieved when the 48'th feature was added was 0.8857. But this score was achieved on a small validationset of 5000 rows. The model with 48 features was trained on the trainingset and the resulting model was then tested on 10 separate parts of the testset. The source code is almost identical with the source code used to recreate the winning approach, and this can be found in appendix B.11. The results can be seen in table **??**.

The AUC score of the 10 runs falls in the interval [0.888,0.899], which is close to the 0.8857 that was achieved when running the forward selection (see table **??**).

6.1.1.1 Statistics on the AUC-score

First the sample mean and sample standard deviation are calculated

$$m = 0.8936 \qquad \text{and} \qquad s = 0.003366$$

which gives that

$$\mathbb{P}(\mu_{AUC} \in [0.8912, 0.8960]) = 95\%$$

Run	AUC
1	0.8333
2	0.8291
3	0.8314
4	0.8264
5	0.8299
6	0.8362
7	0.8308
8	0.8438
9	0.8338
10	0.8234

Table 6.3: *Results from calculating the AUC-score on 10 different parts of the report testset, with the top 3 features selected by forward selection.*

6.1.2 Testing performance on the top 3 features

With a model with an AUC score of 0.89, the goal of improvement of the winner model, has been achieved. But this improvement was created by a much more complex model than the winning model, and one of the reasons the winning model was kept simple, was exactly the problem with models scoring high on validationsets, but low on the ford testset. To make a (maybe) more fair comparison, the performance of a model with the top 3 features from the feature selection is calculated. The procedure is exactly as with all 48 features and the new results are seen in table 6.3

The results hint at what seems to be a slight improvement of the winning model. Since two of the three features was shared between the winning model and this model, the improvement is achieved by substituting feature `sdE5` from the winning model with feature `sdE1` in this model. The parameters of the new model are given by

$$\log \frac{P(t=0|\boldsymbol{x})}{P(t=1|\boldsymbol{x})} = -0.0988 \cdot \mathrm{sdE1} + 0.2019 \cdot \mathrm{V11} + 3.6418 \cdot \mathrm{E9} - 4.2076$$

6.1.2.1 Statistics on the AUC-score

First the sample mean and sample standard deviation are calculated

$$m = 0.8318 \qquad \text{and} \qquad s = 0.005572$$

which gives the 95% confidence interval

$$\mathbb{P}(\mu_{AUC} \in [0.8278, 0.8358]) = 95\%$$

Chapter 7

Neural Network

In this chapter the logistic regression models from the previous chapters, are tried to be improved. Specifically a neural network is trained on the features of winning model as well as on the top 3 features from the forward selection in section 6.1. It wasn't possible to train a neural network on all 48 features from the feature selection, due to limited computational resources. Instead it was tested how the number of hidden units as well as the number of iterations of the optimization algorithm, affected the performance. For each model training a random subset of 100000 rows from the trainingset was used.

7.1 The winning model features

In this section the features of winning model are used as input for neural networks with various numbers of hidden units, and trained with a different number of optimization iterations.

Run	AUC
1	0.8132
2	0.8074
3	0.8116
4	0.8040
5	0.7959
6	0.8070
7	0.7975
8	0.8046
9	0.8035
10	0.8006

Table 7.1: *Results from training a neural network with 3 hidden units for 8 iterations on the features of the winning model*

7.1.1 3 hidden units - 8 iterations

The basic setting for the network training was a network with 3 hidden units and 8 weight optimization iterations. It is worth noticing that 8 iterationsisn't many iterations, but the trainingset was large (100000 rows) and the computational resources few.

A network was trained for the basic settings and the performance was then tested on the testset. The results from the validation are shown in table 7.1

The results of the neural network is close to the results achieved by the logistic regression in section 5.2, although the network seems to be doing a bit worse. A confidence interval is calculated to get an better idea.

7.1.1.1 Statistics on the AUC-score

First the sample mean and sample standard deviation are calculated

$$m = 0.8045 \qquad \text{and} \qquad s = 0.005583$$

which gives the 95% confidence interval

$$\mathbb{P}(\mu_{AUC} \in [0.8005, 0.8085]) = 95\%$$

Run	AUC
1	0.8060
2	0.8087
3	0.8048
4	0.8032
5	0.8068
6	0.8025
7	0.8046
8	0.8060
9	0.8070
10	0.8020

Table 7.2: *inference-features - 5 hidden units - 8 iterations*

this is seen to overlap with the confidence interval of the logistic regression calculated in section 5.2.1

7.1.2 5 hidden units - 8 iterations

The number of hidden units is increased from 3 to 5. The features are still from the winning model and the number of iterations is still 8. Training and then testing the network gives the results seen in table 7.2.

No obvious improvements in the AUC score is achieved by raising the number of hidden units. The AUC-score is still a little above 0.8. The usual statistics are calculated on the results.

7.1.2.1 Statistics on the AUC-score

First the sample mean and sample standard deviation are calculated

$$m = 0.8052 \qquad \text{and} \qquad s = 0.002150$$

which gives the 95% confidence interval

$$\mathbb{P}(\mu_{AUC} \in [0.8036, 0.8067]) = 95\%$$

which overlap with the confidence interval of the network with 3 hidden units.

Run	AUC
1	0.7989
2	0.8073
3	0.7995
4	0.8053
5	0.8052
6	0.8046
7	0.8080
8	0.8042
9	0.8103
10	0.8060

Table 7.3: *inference-features - 3 hidden units - 20 iterations*

7.1.3 3 hidden units - 20 iterations

Instead of raising the number of hidden units, the number of iterations is increased from 8 to 20. The hidden units are returned to 3. As usual the results are gathered in a table, namely table 7.3.

No increase in performance is seen and maybe the number of iterations is simply to low for any effect to be seen. For completeness the usual statistics are calculated.

7.1.3.1 Statistics on the AUC-score

First the sample mean and sample standard deviation are calculated

$$m = 0.8049 \qquad \text{and} \qquad s = 0.003525$$

which gives the 95% confidence interval

$$\mathbb{P}(\mu_{AUC} \in [0.8024, 0.8075]) = 95\%$$

7.1.4 5 hidden units - 20 iterations

For completeness a network was also trained with 5 hidden units and a total of 20 iterations. The results can be seen in table 7.4 Again a consis-

Run	AUC
1	0.8010
2	0.8017
3	0.8072
4	0.8033
5	0.8027
6	0.8131
7	0.8064
8	0.7991
9	0.8050
10	0.8029

Table 7.4: *inference-features - 5 hidden units - 20 iterations*

tent performance around the 0.8 is achieved. Not much to declare and the statistics gives no surprises.

7.1.4.1 Statistics on the AUC-score

First the sample mean and sample standard deviation are calculated

$$m = 0.8042 \qquad \text{and} \qquad s = 0.003966$$

which gives the 95% confidence interval

$$\mathbb{P}(\mu_{AUC} \in [0.8014, 0.8071]) = 95\%$$

7.2 The forward selection features

The focus is now turned to the top 3 features selected by forward selection in section 6.1. The same 4 combinations ($[3,5] \times [8,20]$) of hidden units and number of iterations are tested on the forward selection features. When a logistic regression model was trained on the forward selection features (see section 6.1.2) an increased performance, compared to the winning model features, was observed. It is hoped that the same increase is seen in the neural network models.

Run	AUC
1	0.8286
2	0.8336
3	0.8345
4	0.8354
5	0.8272
6	0.8344
7	0.8370
8	0.8339
9	0.8385
10	0.8320

Table 7.5: *forward-selection - 3 hidden units - 8 iterations*

7.2.1 3 hidden units - 8 iterations

First a network with 3 hidden units is trained for 8 iterations. The results are shown in table **??**.

As hoped for the performance is increased compared to the winning models features. Compared to the performance of the logistic regression on the forward selected features no noticeable difference is observed.

7.2.1.1 Statistics on the AUC-score

First the sample mean and sample standard deviation are calculated

$$m = 0.8335 \qquad \text{and} \qquad s = 0.003463$$

which gives the 95% confidence interval

$$\mathbb{P}(\mu_{AUC} \in [0.8310, 0.8360]) = 95\%$$

7.2.2 5 hidden units - 8 iterations

The results for the network model with 5 hidden units trained for 8 iterations are shown in table 7.6.

Run	AUC
1	0.8301
2	0.8301
3	0.8332
4	0.8381
5	0.8320
6	0.8411
7	0.8290
8	0.8340
9	0.8325
10	0.8355

Table 7.6: *forward-selection - 5 hidden units - 8 iterations*

The performance is more or less identical with the performance for 3 hidden units. It may be due to the fact that 8 iterations are not enough training for the additional complexity to show up. Or maybe there is just not much more structure to be learned in the data.

7.2.2.1 Statistics on the AUC-score

First the sample mean and sample standard deviation are calculated

$$m = 0.8336 \qquad \text{and} \qquad s = 0.003800$$

which gives the 95% confidence interval

$$\mathbb{P}(\mu_{AUC} \in [0.8308, 0.8363]) = 95\%$$

7.2.3 forward-selection - 3 hidden units - 20 iterations

The network returns to 3 hidden unit, but is now trained for 20 iterations. The results are shown in table 7.7

From the results it is seen that the performance is increased i little compared to the previous two runs. The difference isn't clear cut though, so a confidence interval may give a clearer picture of how the performances compares.

Run	AUC
1	0.8373
2	0.8432
3	0.8443
4	0.8395
5	0.8382
6	0.8441
7	0.8377
8	0.8382
9	0.8431
10	0.8285

Table 7.7: *3 hidden units - 20 iterations*

7.2.3.1 Statistics on the AUC-score

First the sample mean and sample standard deviation are calculated

$$m = 0.8394 \qquad \text{and} \qquad s = 0.004759$$

which gives the 95% confidence interval

$$\mathbb{P}(\mu_{AUC} \in [0.8360, 0.8428]) = 95\%$$

It is seen that the confidence interval just barely overlaps with the confidence intervals from the 2 previous run. It seems as if this network performs slightly better than the previous two networks.

7.2.4 forward-selection - 5 hidden units - 20 iterations

The last neural network test is for a network with 5 hidden units, that are trained for 20 iterations. As a slight increase in performance was observed when the training was run for 20 iterations, similar good performance may be expected from this test. The results are shown in table 7.8.

As is seen from the results, the performance have decreased noticeably and is the worst of all the networks trained on the forward selected features. Either the optimization got stuck in a local minimum or the network has been overfitted to the trainingdata. Overfitting could make

Run	AUC
1	0.8181
2	0.8203
3	0.8230
4	0.8169
5	0.8227
6	0.8206
7	0.8209
8	0.8188
9	0.8223
10	0.8159

Table 7.8: *5 hidden units - 20 iterations*

sense, since this network has more complexity and have had more time to fit this complexity to the trainingset. It is worth noticing though, that no similar decrease in performance was observed in the networks trained on the winning features.

7.2.4.1 Statistics on the AUC-score

First the sample mean and sample standard deviation are calculated

$$m = 0.8199 \qquad \text{and} \qquad s = 0.002463$$

which gives the 95% confidence interval

$$\mathbb{P}(\mu_{AUC} \in [0.8182, 0.8217]) = 95\%$$

The right endpoint of this interval is about 0.01 below the lowest left endpoint of the confidence intervals, for the tests on the networks trained on the feature selected features.

Chapter 8

Discussion

Bla, bla, bla.

Chapter 9

Conclusion

More bla, bla, bla.

CHAPTER A

Workflow, tools and project evaluation

In this chapter the workflow used while working on this project, is introduced. As the workflow is tightly connected with the tools (software) used, these will be introduced when they are first mentioned. Before the start of this project, I* decided on some "metagoals" for this project

- I should spend some time on setting up a workflow that can be used for future data analysis projects.
- I should be able to set up my working environment on any linux/mac within an hour.
- All project related files should be managed by a version control system
- Anyone that wishes to work on my project should be able to start working on it, within an hour or two.

*The narrative style changes in this chapter, as some personal thoughts and opinions will be expressed

- My workflow should encourage me to write documentation of all the work done
- Find machine learning software libraries that shows potential to become really great in the future.

Reflecting on well the different goals was achieved is the subject of the rest of this chapter.

A.1 Workflow

In this section the workflow that was used throughout this project will be described. A simple session concept, that turned out to work great, is introduced along with some suggestions for improvements. Documentation of the work done is an important part of a project, but it is always neglected, since it is pretty boring to write. A great tool to write documentation is introduced and it is seen how it naturally fits with the session concept.

A.1.1 The session concept

One of the problems when working with any scientific project, that produces some kind of output (data, plots etc.) is where to put all the stuff that is produced. And also where to write down all the notes about things to remember about the output produced (why is this plot extremely interesting etc.). An attempt to solve this problem is the idea of a (working) session.

A session is just an amount of work done in a time period (usually continuous). Every session is given a unique number and a name so that it can easily be identified. In this project all sessions was represented as folders that was named '<sessionnumber>-<sessionname>'. All session folders is placed in the "sessions" folder in the root folder of this project (you can go to https://github.com/alphabits/ford_challenge to explore the root

folder of this project). Anytime any work is done on this project it belongs to a session and all documentation, plots, data and source code generated in a session, can be found in that sessionfolder.

EXAMPLE A.1 *It is time to make forward selection of the features of The Ford Challenge dataset. Session number 18 is created and named "Forward selection", which means that a folder called* `18-forward-selection` *is created in the* `sessions`*-folder. Inside the* `18-forward-selection` *folder the folders* `plots`, `data`, `scripts` *are created. In the folder of session 18 a textfile for documentation is created, and a few lines about the goal of the session is written. After a lot of debugging code, waiting on calculations to end etc. the forward selection has been created. Code and data produced are saved in the folders created for this, but the code and data are also included in the documentation using a tool named Sphinx.*

A.1.2 Documentation with Sphinx

As explained in the previous example documentation is a core part of the session. Two goals are achieved by writing the documentation. First of all by always starting a session by writing a couple of lines about the goals of the session, it helps you to define what you should be working on for the next hours. Secondly there is a place to look up details, when the report is written 3 months later, and all details about forward selection is forgotten. But the documentation is only written if it is easy to write and update, and if it is possible to easily create a nice representation of the documentation (eg. a html- or latex-document). A tool that can help create nice documentation is the library Sphinx (`http://sphinx.pocoo.org/`). Sphinx is a Python library that creates documentation from textfiles written in a format called reST (`http://docutils.sourceforge.net/rst.html`). The Sphinx program is called from the root folder of the project, and it then automatically scans all subfolder for reST-files, and then creates a complete website, with link between your documents. Latex code can be included in the documentation giving documentation with nice equations. All the source code developed in a session can be included in the documentation and it will automatically be syntax higlighted. Todo-boxes, notes etc. can also be included to make a more useful documentation.

A.1.3 How it worked in the real world

In the previous sections the session concept used for this project was introduced, and a tool to easily create useful documentation was briefly presented. But this was the sunny-side presentation. How did it work out when used in the project?

First of all the concept of always working within a session worked out really well. At no time was it a problem to find the right place to save a plot or write a note. It is also worth noticing that the session concept is pretty flexible, so every meeting with my **TODO: Vejleder?** was recorded as a session, and a dummy session called "next meeting" was always present to write down questions for the next meeting. When the meeting was over, answers to the questions was filled in.

One thing that didn't work that well was the time registration of each session. All session documentation was started with a start time and end time for the session. The hope was to create a timeline of the project to include in the report or presentation. But to make this idea work most had to be done sequentially and that didn't happen. Especially in the last few weeks, many hours have been spend in old sessions, and this broke the idea of just saving a start and end time for a session.

Also the session concept requires much discipline, and discipline is inversely proportional to the to the time left before deadline. Almost all work done in the last weeks is undocumented.

All things considered the session concept was a success and it will be reused, and hopefully enhanced, in the next project to come.

A.2 Version control

One of the stated goal of this project was to keep all files in version control. This was achieved. Using the version control system Git, and the free online GitServer-service called github, all project files are available online at `https://github.com/alphabits/ford_challenge`. Before this project

my experience with version control systems was limited. But after this project, most future projects will probably be kept in a version control system. Two examples shows some of the benifits of version control.

EXAMPLE A.2 *It is 3am and the last few plots out of a total of 150 plots of features have just been created. The next meeting with the* ***TODO: Bejleder*** *is at 9pm and these plots is the main focus of the meeting. Issuing the commands*

```
$ git commit -a -m "Created plots of features for 20 random trails"
$ git push origin master
```

uploads all files that have been updated or created, to the projects github page, including all the plots and documentation with some questions for the meeting. These files can then be browsed in the webinterface of github.

EXAMPLE A.3 ***TODO: ?????***

A.3 NumPy/SciPy, scikits.learn and pybrain

A central decision for this project was to choose which scientific computation software to use. The most popular tool at DTU seems to be Matlab, and a machine learning library with all the functionallity needed for this project, have been developed at IMM. The problem with Matlab though, is that it isn't open-source. Since one of the goals listed at the beginning of the chapter was that any person should be able to start working on the project within an hour or two, this more or less implies that open-source software should be used. On a more personal note, I find the programming model of Matlab really clumsy and inflexible, and as soon as anything that doesn't naturally involves a matrix, needs to be done, the language is pretty much a hindrance.

Instead of choosing Matlab, I decided to use Python[†]/NumPy[‡]/SciPy as the scientific computation platform. Python is a very popular programming

[†] http://xkcd.com/353/
[‡] http://numpy.scipy.org/

language with a very simple syntax and NumPy/SciPy is a pretty mature scientific computation library for Python, with efficient array and matrix data types, which pretty much matches Matlab in functionallity[§]. Apart from NumPy/SciPy the plotting library matplotlib (http://matplotlib.sourceforge.net/) as well as the interactive console ipython (http://ipython.scipy.org/moin/), are needed to get a capable scientific computing environment. All these libraries combined gives an open-source environment with almost all the functionallity neede for this project. The few missing pieces is the machine learning specific libraries. For this the scikits.learn library (http://scikit-learn.sourceforge.net/) and the pybrain library (http://pybrain.org/) was installed. The scikits.learn library is pretty well documented and has a nice api, but it lacks a neural network implementation. Pybrain was therefore selected for the neural network training.

A.3.1 Comments on the choice of Python instead of Matlab

Matlab wasn't really an option at any time as I think that open source software should be used as much as possible. And since the requirements for this project could be fulfilled with the Python solution, there was little doubt about the decision. If the decision to use Python is evaluated with only this project in mind, the decision may very well have been wrong. Much time was used to find, install and learn libraries and the documentation was't always the best. But the cost declines with time, and I'm sure that when it is time to write my Bachelor, the decision will turn out to be the right.

A.4 General reflections on the project

[§]See http://www.scipy.org/NumPy_for_Matlab_Users

Chapter B

Appendices

A little introduction and then a new page

B.1 Calculating common statistics - Code

```
# sessions/9-data-exploration/src/calculate_summary_statistics.py

from json import dump

from src.data_interface import trd, L
from src.utils import get_path


sess_root = get_path(__file__) + '/../scr'

summary_statistics = ['min', 'max', 'mean', 'std']
features_to_calculate = L[2:]
calculations = {}

for feature_name in features_to_calculate:
    calculations[feature_name] = {}
    for statistic in summary_statistics:
        stat_function = getattr(trd.get_feature(feature_name), statistic)
        calculations[feature_name][statistic] = stat_function()

f = open(sess_root + '/summary_statistics.json', 'w')
dump(calculations, f, indent=4)
f.close()
```

Code Listing B.1: *https://github.com/alphabits/ford_challenge/tree/master/sessions/9-data-exploration/src/calculate_summary_statistics.py*

B.2 Calculating common statistics - Result

Feature	Mean	Min	Max	Std.Dev
E1	10.54	0.00	243.99	14.00
E10	63.38	0.00	127.00	19.23
E11	1.37	0.00	52.40	5.37
E2	105.05	0.00	360.00	128.33
E3	0.30	0.00	4.00	1.03
E4	-3.99	-250.00	260.00	34.80
E5	0.02	0.01	0.02	0.00
E6	358.50	260.00	513.00	27.84
E7	1.81	0.00	25.00	2.95
E8	1.39	0.00	9.00	1.62
E9	0.87	0.00	1.00	0.33
IsAlert	0.58	0.00	1.00	0.49
P1	35.45	-22.48	101.35	7.36
P2	12.01	-45.63	71.17	3.74
P3	1028.00	504.00	2512.00	309.70
P4	63.99	23.89	119.05	19.76
P5	0.18	0.04	27.20	0.39
P6	843.73	128.00	228812.00	2795.32
P7	78.40	0.26	468.75	18.79
P8	0.00	0.00	0.00	0.00
V1	75.58	0.00	129.70	44.94
V10	3.28	1.00	7.00	1.27
V11	11.59	1.68	262.45	8.43
V2	-0.04	-4.79	3.99	0.42
V3	569.78	240.00	1023.00	299.02
V4	21.24	0.00	484.49	67.21
V5	0.18	0.00	1.00	0.38
V6	1699.34	0.00	4892.00	626.27
V7	0.00	0.00	0.00	0.00
V8	12.46	0.00	82.10	11.54
V9	0.00	0.00	0.00	0.00

B.3 Calculate unique values of features - Code

```
from json import dump

from src.data_interface import trd, L
from src.utils import get_path


sess_root = get_path(__file__) + '/../src'

features_to_calculate = L[2:]
unique_values = {}

for feature_name in features_to_calculate:
    unique_values[feature_name] = trd.get_feature(feature_name).unique_values()

f = open(sess_root + '/unique_values.json', 'w')
dump(unique_values, f, indent=4)
f.close()
```

Code Listing B.2: *https://github.com/alphabits/ford_challenge/tree/master/sessions/9-data-exploration/src/calculate_unique_values.py*

```
from json import dump
import numpy as np

from src.data_interface import trd, L
from src.utils import get_path


sess_root = get_path(__file__) + '/..'

features_to_calculate = L[2:]
trials = list(trd.trial_id_list)
calculations = {}

for feature_name in features_to_calculate:
    tmp = {"trial_results": {}}
    for trial_id in trials:
        unique_values = np.unique(
                trd.get_trial(trial_id).get_feature(feature_name).view())
        tmp["trial_results"][trial_id] = unique_values.size
    tmp["max"] = max(tmp["trial_results"].values())
    tmp["min"] = min(tmp["trial_results"].values())
    calculations[feature_name] = tmp

f = open(sess_root + '/src/unique_values_pr_trial.json', 'w')
dump(calculations, f, indent=4)
f.close()
```

Code Listing B.3: *https://github.com/alphabits/ford_challenge/tree/master/sessions/9-data-exploration/src/calculate_unique_values_pr_trial.py*

```
import json

from src.utils import get_path


path = get_path(__file__)

def get_dict(file_name):
    f = open(path + '/' + file_name, 'r')
    d = json.load(f)
    f.close()
    return d

unique_pr_trial = get_dict('unique_values_pr_trial.json')
unique_all_data = get_dict('unique_values.json')


values_combined = {}

for k in unique_all_data:
    tmp = {}
    tmp["all_data"] = unique_all_data[k]
    tmp["min_pr_trial"] = unique_pr_trial[k]["min"]
    tmp["max_pr_trial"] = unique_pr_trial[k]["max"]
    values_combined[k] = tmp

f = open(path + '/unique_values_combined.json', 'w')
json.dump(values_combined, f, indent=4)
f.close()
```

Code Listing B.4: *https://github.com/alphabits/ford_challenge/tree/master/sessions/9-data-exploration/src/unique_values_combined.py*

B.4 Calculate unique values of features - Result

Feature	Min in trial	Max in trial	Whole dataset
E1	1	131	12265
E10	1	86	121
E11	1	79	116
E2	1	129	28502
E3	1	3	3
E4	1	97	249
E5	1	134	254
E6	1	116	247
E7	1	15	25
E8	1	10	10
E9	1	2	2
IsAlert	1	2	2
P1	967	1208	105715
P2	1155	1207	128666
P3	24	106	406
P4	24	106	406
P5	34	114	1070
P6	3	99	406
P7	3	99	406
P8	1	1	1
V1	1	969	12374
V10	1	5	5
V11	875	1206	166455
V2	1	76	90
V3	1	29	33
V4	1	204	324
V5	1	2	2
V6	1	775	2727
V7	1	1	1
V8	1	239	323
V9	1	1	1

B.5 Creating boxplots - Code

TODO: Include source code

B.6 Creating boxplots - Result

TODO: Include boxplots

B.7 Creating layered featuer plots - Code

B.8 Creating layered feature plots - Result

B.9 Scatterplots - Code

```
import itertools as it

import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
import numpy as np

from src.dataloaders import trainingset
from src.utils2 import get_path, c, L

L = list(L[4:])
D = trainingset()

path = get_path(__file__) + '/..'
savepath_template = '{0}/plots/scatterplots/{1}-{2}.pdf'
num_points = 100

rows = np.random.random_integers(0,D.shape[0]-1, num_points)
data = D[rows,:]

colors = map(lambda x: 'blue' if x==1 else 'red', data[:,c('IsAlert')])
blue = Rectangle((0,0),1,1,fc='b')
red = Rectangle((0,0),1,1,fc='r')

exclude = ['V7', 'V9', 'P8', 'E3', 'E7', 'E8', 'E9', 'V3', 'V5', 'V10']
features = [f for f in L if f not in exclude]

for f1, f2 in it.combinations(features, 2):
#for f1, f2 in [['E4', 'E5']]:
    plt.title('Feature {0} vs {1} ({2} points)'.format(f1, f2, num_points),
            {'size': 20})
    plt.legend((blue, red), ('Alert', 'Not Alert'))
    plt.scatter(data[:,c(f1)], data[:,c(f2)], c=colors)
    plt.gca().set_xlabel(f1, {'size': 18})
    plt.gca().set_ylabel(f2, {'size': 18})
```

```
    plt.savefig(savepath_template.format(path,f1,f2), format='pdf',
            papertype='a4')
    plt.cla()
```

Code Listing B.5: *https://github.com/alphabits/ford_challenge/tree/master/sessions/24-writing-helper-scripts/scripts/rewrite-scatterplot-script.py*

B.10 Scatterplots - Result

B.11 Recreating the winning approach - Code

```
from __future__ import division

import json
import random

import numpy as np
import matplotlib.pyplot as plt

from src.utils2 import c_ex as c, get_path, get_bins
import src.dataloaders as d
from src.logistic import fit_logistic_regression


path = get_path(__file__) + '/..'

D = d.trainingset_extended()
#D = d.testset_extended()

cols = c('sde5', 'v11', 'e9')

C = 1000000

num_bins = 10
bins = get_bins(D.shape[0], num_bins)

X = D[:, cols]
y = D[:, c.isalert]

auc = num_bins*[0]
fpr = num_bins*[0]
tpr = num_bins*[0]
w = num_bins*[0]
b = num_bins*[0]
for i in range(num_bins):
```

```
    train = [item for j in range(num_bins) if j != i for item in bins[j]]
    test = bins[i]
    auc[i], fpr[i], tpr[i], w[i], b[i] = fit_logistic_regression(
            X[train,:], y[train,:], X[test,:], y[test,:], C=C)

def save_result(save_path):
    with open(save_path, 'w') as f:
        json.dump(get_result(), f, indent=4)

def get_result():
    wl = [a[0].tolist() for a in w]
    save_data = dict(zip(
        ['weights', 'auc', 'intercepts', 'C', 'num_bins', 'dataset_size', 'generator'],
        [wl, auc, b, C, num_bins, D.shape[0], __file__]
    ))
    return save_data
```

Code Listing B.6: *https://github.com/alphabits/ford_challenge/tree/master/sessions/17-recreating-winning-entry/scripts/inferences_features_cv.py*

B.12 Forward selection - Code

```
import json
import random

import numpy as np

from src.utils2 import c_ex as c, get_path, L_ex, LabelIndex
import src.dataloaders as d
from src.logistic import fit_logistic_regression


path = get_path(__file__) + '/..'

D = d.trainingset_extended()

a = range(D.shape[0])
random.shuffle(a)

num_train_rows = 10000
num_test_rows = 5000

tr_rows = a[:num_train_rows]
ts_rows = a[num_train_rows:(num_train_rows+num_test_rows)]

X = D[:, 4:]
X = X[tr_rows, :]
y = D[tr_rows, c('isalert')]
```

```
Xt = D[:, 4:]
Xt = Xt[ts_rows, :]
yt = D[ts_rows, c('isalert')]

auc = np.zeros((90,90));

# Remove P3, P6, P8, V7 and V9 and
# the corresponding running features.
# See session 9 on data exploration
# for details
cc = LabelIndex(L_ex[4:])
exclude = cc('p3', 'p6', 'p8', 'v7', 'v9',
             'mp3', 'mp6', 'mp8', 'mv7', 'mv9',
             'sdp3', 'sdp6', 'sdp8', 'sdv7', 'sdv9')
candidates = [i for i in range(90) if i not in exclude]

chosen = []

for i in range(90):
    for c in candidates:
        features = chosen + [c]
        result = fit_logistic_regression(X[:, features], y, Xt[:, features], yt)
        auc[i, c] = result[0]

    chosen_feature = auc[i,:].argmax()

    if auc[i,chosen_feature] <= auc[i-1,:].max():
        break

    candidates.remove(chosen_feature)
    chosen.append(chosen_feature)
```

Code Listing B.7: *https://github.com/alphabits/ford_challenge/tree/master/sessions/18-forward-selection/scripts/forward_selection.py*

B.13 Forward selection - Results

Feature added	AUC
V11	0.6978
E9	0.8089
sdE1	0.8333
mE6	0.8500
sdE4	0.8539
mV3	0.8566
sdV8	0.8597
E8	0.8612
mE10	0.8623
mP5	0.8635
sdP5	0.8687
mP7	0.8705
mV11	0.8717
V10	0.8727
sdV1	0.8737
mE4	0.8744
sdP2	0.8759
sdE6	0.8766
sdP7	0.8771
sdP1	0.8778
sdE2	0.8781
mV2	0.8785
mE8	0.8790
sdV10	0.8793
continues ...	

Feature added	AUC
... continued	
sdV3	0.8799
sdE9	0.8802
sdE8	0.8811
mE7	0.8818
mE9	0.8822
mV4	0.8827
mV5	0.8832
mE1	0.8833
E6	0.8835
V4	0.8839
V2	0.8839
E10	0.8840
V5	0.8841
sdE3	0.8843
sdE11	0.8843
P4	0.8844
mE11	0.8846
sdP4	0.8848
sdV2	0.8848
sdV4	0.8849
mV8	0.8851
mV6	0.8855
mP1	0.8857
sdE5	0.8858

Bibliography

Abou-Nasr, M. (2011a). Kaggle forum: About the parameter - Reply 3. URL http://www.kaggle.com/c/stayalert/forums/t/317/about-the-parameter/1861#post1861.

Abou-Nasr, M. (2011b). Kaggle forum: Discrete or continuous values - Reply 2. URL http://www.kaggle.com/c/stayalert/forums/t/266/discrete-or-continuous-values/1635#post1635.

Abou-Nasr, M. (2011c). Kaggle forum: Were units changed? - Reply 2. URL http://www.kaggle.com/c/stayalert/forums/t/268/were-units-changed/1641#post1641.

ACM (2011). KDD Cup Center. URL http://www.sigkdd.org/kddcup/index.php.

Alpaydin, E. (2010). *Introduction to Machine Learning*. The MIT Press, 2 edition.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer Science + Business Media LLC.

David (2011). Methods/Tips from non-top 3 participants. URL http://www.kaggle.com/c/stayalert/forums/t/328/methods-tips-from-non-top-3-participants/1967#post1967.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification*. John Wiley & Sons, Inc.

Hand, D. J. (2009). Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Machine Learning*, 77(1), 103–123.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*. Springer Science + Business Media LLC.

Inference (2011a). First place in the stay alert! competition. URL http://www.kaggle.com/blog/wp-content/uploads/2011/03/ford.pdf.

Inference (2011b). Kaggle forum: Ford - Reply 3. URL http://www.kaggle.com/c/stayalert/forums/t/295/ford/1743#post1743.

Johnson, R. A. (2005). *Miller and Freund's Probability and Statistics for Engineers*. Pearson Education, Inc.

Kaggle.com (2011). The Ford Challenge Data Files. URL http://www.kaggle.com/c/stayalert/Data.

Labs, Y. (2011). KDD Cup 2011. URL http://kddcup.yahoo.com/.

Mørup, M. (2011). Meetings with Morten Mørup.

Netflix (2011). Netflix Prize Leaderboard. URL http://www.netflixprize.com/leaderboard.

Pardos, Z. (2011). Kaggle forum: Top two teams with same AUC - Reply 1. URL http://www.kaggle.com/c/stayalert/forums/t/327/top-two-teams-with-same-auc/1937#post1937.

Rosanne (2011). Kaggle forum: Top two teams with same AUC - Reply 4. URL http://www.kaggle.com/c/stayalert/forums/t/327/top-two-teams-with-same-auc/1957#post1957.

Tan, P.-N., Steinbach, M., and Kumar, V. (2006). *Introduction to Data Mining*. Pearson Education, Inc.

Wasserman, L. A. (2004). *All of Statistics: a concise course in statistical inference*. Springer Science + Business Media Inc.

Wikipedia (2011). Netflix Prize. URL http://en.wikipedia.org/wiki/Netflix_Prize.