NTNU

Norwegian University of
Science and Technology

Early Warnings of Corporate Bankruptcies Using Machine Learning Techniques

Jostein Gogstad
Jostein Øysæd

Master of Science in Computer Science
Submission date: June 2009
Supervisor: Helge Langseth, IDI
Co-supervisor: Tore Anders Husebø, SpareBanken 1 SR-Bank

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

We develop a model, using machine learning techniques, capable of recognizing patterns in transactions to and from a company's tax withdrawal account that indicates a financially unhealthy company. The model is easily integrated into an early warning system capable of flagging Norwegian companies which represent a high risk for defaulting on a loan or going into bankruptcy.

Assignment given: 26. January 2009
Supervisor: Helge Langseth, IDI

Abstract

The tax history of a company is used to predict corporate bankruptcies using Bayesian inference. Our developed model uses a combination of Naive Bayesian classification and Gaussian Processes. Based on a sample of 1,184 companies, we conclude that the Naive Bayes-Gaussian Process model successfully forecasts corporate bankruptcies with high accuracy. A comparison is performed with the current system in place at one of the largest banks in Norway. We present evidence that our classification model, based solely on tax data, is better than the model currently in place.

Preface

This report constitutes our master thesis, written during the 10th semester of the Master of Science studies in Computer Science at the Norwegian University of Science and Technology (NTNU). It was carried out at the Department of Computer and Information Science (IDI). The work started January 26th, 2009 and ended June 10th, 2009.

First, we would like to thank Tore A. Husebø and Geir Tjentland who first came up with the idea behind this thesis. Without their invaluable feedback and tutoring this thesis would never been possible. We are also grateful to BearingPoint, Oslo, for taking the initiative for this thesis, and thus giving us the opportunity to work with such an interesting field as the intersection between machine learning and economics.

We will especially like to thank our professor and supervisor Helge Langseth. Without his advice, feedback, support and passion for lengthy discussions, this thesis would never see the light of day. Albert Einstein once said: "*It is the supreme art of the teacher to awaken joy in creative expression and knowledge.*" A quote summarizing Helge Langseth's contribution to this thesis, as he has provided encouragement and expertise in all phases of this project.

At last, a big thank you goes to C. E. Rasmussen at University of Cambridge for providing us with hours of video lectures, and not to mention the implementation of Gaussian Processes (`http://www.gaussianprocess.org/gpml/`).

NTNU, **June 10**, 2009.

Jostein Gogstad

Jostein Øysæd

Contents

1 Introduction

The prediction of corporate bankruptcies is an important and widely studied topic since it can have significant impact on bank lending decisions and profitability. Banks need to predict the probability of a loss when extending loans; accurate estimates of risk associated with loans leads to sounder lending decisions and can result in significant savings. After the financial crisis hit in September 2008, the topic of estimating a company's financial health is even more relevant.

A common feature of the various studies in bankruptcy prediction is the use of industry-relative ratios: comparing company financial ratios with industry ratios [Platt and Platt, 1991]. Altman and Izan pioneered the field of industry-relative ratios in [Altman, 1973], among the six ratios presented where sales/total assets, working capital/total assets and market capitalization/total debt. These ratios has gained extensive popularity in the research on bankruptcy prediction; Neural Networks has been especially popular [Atiya, 2001; Odom and Sharda, 1990; Wilson and Sharda, 1994; Zhang et al., 1999].

The ratios proposed by Altman depends on industry numbers and company information which is (1) only available every quarter for public companies and once a year for other companies, and (2) the numbers may be inaccurate or not available for all companies, e.g. market capitalization is only available for public companies.[1] The contribution of this thesis is a prediction model which can—at any given time—provide an estimate of the financial health of a company independent of accounting and industry information.

Specifically, we base our estimator on transactions to, and from, the "Tax Withdrawal Account" explained in the next chapter. We theorize that *the financial health of a company is reflected in the management of the Tax Withdrawal Account, especially before a bankruptcy or other actions leading to a loss for the bank.*

The model we propose for prediction is two-folded: We first perform a coarse grained separation of companies in Part I. The result is two sets of companies: one with healthy companies which is left out of further analysis, the second with companies that has a higher probability of incurring a loss for the bank and which will by analysed in Part II. By partitioning the companies in this way we hope to achieve a homogeneous set of "hard" cases which can be analysed apart from the rest.

All data used in this thesis was provided by one of the largest banks in Norway.

1.1 Terminology

Before continuing, we must introduce a few words about the terminology used by banks in the context of loans and credit rating.

A *default* occurs when a debtor (company in our case) fails to meet the obligations according to a loan agreement. Defaults together with bankruptcies are important when considering the financial health of a company.

Companies in our dataset (see Chapter 3) are marked accordingly if the bank has reckoned a loss in connection with that company. A *financially healthy* (or simply healthy) company, is a company that has no defaults nor any loss reckonings for the past four years. An *unhealthy* company is a company that has defaulted on a loan and/or the bank has reckoned a loss.

A *loss* associated with a loan for a bank is not restricted to bankruptcies, we therefore refrain from using the term "bankruptcy" explicitly, but instead use the term "loss" or "loss-reckoning".

[1] Market capitalization = share prize × outstanding shares

1.2 Purpose

The purpose of this thesis is to create a *classification model*—or classifier—which is able to accurately estimate the financial health of a company at any given time. A classification model is a mapping of instances to a certain class/group. Our classification model will assign a real value to a company indicating the financial health of that company. The health-indicator is based on events that directly or indirectly leads to losses associated with loan agreements. An example of the former would be bankruptcies, an example of the latter is defaults.

The resulting classifier has the following characteristics:

- *Reactive*: It is able to estimate the financial health of a company at any time.
- *Proactive*: A potential bankruptcy or default is flagged before the actual event takes place.
- *Independency*: It is independent of accounts and other information which is (1) only updated certain times a year, or (2) hard to measure accurately.
- *Consistency*: No subjective measures are taken into account.

The classifier is deemed successful if it can describe more accurately the financial health of a company than the PD-rating.

1.2.1 The PD-rating

Our banks current credit rating system outputs a "PD-rating" in the range $[0, 1]$ where low values indicates healthy companies. In addition, the bank operates with *risk classes* derived directly from the PD-rating:

$$\text{Risk class} : \text{PD-Rating} \to \{A, B, C, D, E, F, G, H, I\} \cup \{J, K\}$$

If a company defaults on a loan, the company is automatically rated with PD = 1.0 and the risk class `J`. If some other loss-reckoning is made, the company is rated with PD = 1.0 and the letter `K`. Else, the company is rated with a PD rating in the range $[0, 1)$ and a risk class from the set $\{A, B, C, D, E, F, G, H, I\}$ (*cf.* Appendix A.1 for details on the mapping).

The PD-rating is calculated by a semi-automatic process which is subject to both objective and subjective measures. Our model should beat the PD-rating in identifying companies which will incur a loss for the bank. In other words, the PD-rating acts as a measurement of how good our model is.

1.3 Scope

We are only concerned with the corporate prediction problem in this thesis. For the private consumer problem, we refer the reader to [Šušteršič et al., 2009; Hand and Henley, 1997 or Serrano-Cinca, 1996] for a review.

More specifically, only companies with the following characteristics are used to train and evaluate the classifier:

1. The company has to be public. In bank terminology, this is the same as specifying that the company is in "sub-sector 710". The opposite of a public company is a privately owned company.
2. We only use companies from the following industries: *Construction*, *Hotel*, *Retail* and *Industry*. This is by request from the bank.

3. The company need to be larger than a certain threshold. Small companies are not considered.

4. The company has to pay tax; we require at least half a year of payments—without any defaults or bankrupts—before any estimation can be done.

The list above was advised by the bank. By restricting our analysis to public companies in the industries mentioned in point 2, we hope to find more homogeneous patterns in the deposits and withdrawals from the Tax Withdrawal Account. It should be noted however that public companies are not a requirement for the method, it is merely a choice for testing purposes. The same is true for the industry requirement in point 2.

We expect small companies with only one or a few employees to behave significantly different from the rest; the analysis is therefore restricted to companies of a certain size. We need a certain amount of data to perform prediction, by manual testing we have determined that we need *at least* six months of data, preferably a year, to correctly classify a company.

1.4 Success Criteria

The success criteria for the thesis is summarized below

1. The resulting model is easily integrated with a system capable of automatically supervise corporate customers of the bank and flag companies which will default or incur a loss in near future.
2. The performance (measured in accuracy, ROC or any other suitable measure) of the classification model should be better than the performance of the existing system—the PD-rating.

2 Background

SpareBank 1 is an alliance of 24 Norwegian Banks. In addition to administrative responsibility for common processes among the banks, the alliance also do research development on—among other things—estimation of credit ratings. A "Credit rating" assesses the credit worthiness[1] of an entity, it can be a private person, a company or any other establishment that can undertake a loan. A high credit rating indicates high risk of defaulting, and thus leads to more restrictive covenants in the loan agreement. On the other hand, a low credit rating indicates that risk of defaulting is low so the debtor may loan more money or get less restrictive covenants. The *credit rating estimation problem* is to calculate the credit rating of an entity with only limited information.

Traditionally, banks use quantitative and subjective factors such as leverage, earnings, reputation or ratios suggested by [Altman, 1973] to calculate credit ratings through a scoring system [Treacy and Carey, 1998]. The problem with this approach is the subjective aspect which makes it hard to achieve consistent estimates.

SpareBank 1 is seeking a replacement for the system currently in place, the resulting model from this thesis may in the future be used as a part of the new system. Today, SpareBank 1 does credit assessment of all its customers—both private consumers and corporate companies—using an in-house system. SpareBank 1's current system is complex and relies on many parameters to estimate an entity's credit worthiness. Some of these parameters may not be available at any time, others are in-frequently updated, maybe only a few times a year (e.g. accounts and returns). There is also a certain degree of subjectivity in the ratings.

We base our credit rating estimate on a special bank account which is required by Norwegian law to be maintained by all corporations. The "Tax Withdrawal Account" is described in more detail in the next section; in short it provides information about how much tax is payed off wages every month.

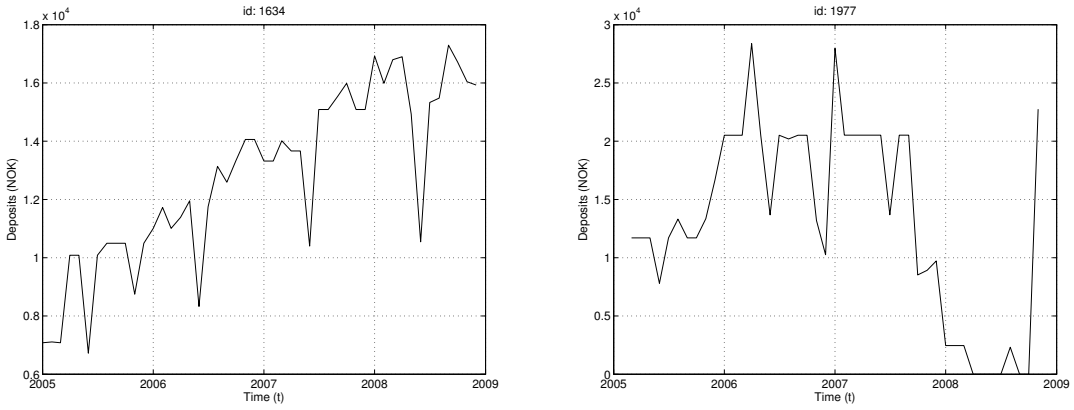
2.1 The Tax Withdrawal Account

According to the Norwegian tax law, a company is liable to pay tax off the wages payed to workers. This tax is transferred from the company to the authorities every *second* month (starting January) and is reserved on a special tax account each month. According to the law, a company is required to maintain a balance on the Tax Withdrawal Account equal to wage taxes every month. In addition, the money on the Tax Withdrawal Account is reserved and cannot be used for anything else (such as paying other creditors).

We hypothesize that the management of this account reflects to a certain degree the state of the company. For example, if the company is transferring more and more money to the tax withdrawal account, it means that it is paying more wages (more wages leads to more tax). This might be because an increase in staff or salary. Either way, it is a positive sign for the health of the company. Conversely, if a company is paying less and less tax to the tax withdrawal account, one might suspect that the company is in trouble and that the bank should take steps to secure any liabilities the company might have.

In the figure on the following page we see deposits to the Tax Withdrawal Account for two different companies. In Figure 2.1 (b) we make two observations: there is an overall negative trend in the deposits; the company is for some reason transferring less and less money to the account. The balance plot for this company (not shown) shows that there is no extra money on the account, so we interpret this as a bad sign for the health of the company.

[1] The risk of loss due to debtor's non-payment of a loan.



(a) (b)

Figure 2.1: Two trivial examples. (a) A healthy company's deposits to the tax withdrawal account. (b) A non-healthy company's deposits.

2.2 Challenges

Classifying the two companies above is a trivial task and form the basis for our assumption that there is enough information in the Tax Withdrawal Account to predict financially troubled companies. In the general case however, such well-behaved companies as those shown in Figure 2.1 are extremely rare. Consider the figure on the facing page.

For the healthy companies in Figure 2.2 (a) and (b) we notice that there is very little structure in the deposits. This is fairly common for all the companies in the data set. Second, we notice that the company in (b) several times does not transfer money to the Tax Withdrawal Account, there is also a lack of trend in the deposits for any of the companies.

Figure 2.2 (c) and (d) shows deposits for two companies that has economical problems; they have defaulted or a loss was reckoned some time during the period. Figure (d) displays the deposits up to a bankruptcy. The company suddenly transfer very little money (but not zero) to the account before they go bankrupt, this is typical hard-case. The default in figure (c) also appear somewhat out of thin air, even though we might accept a *slight* downward trend during 2006.

As a last note we note that by focusing solely on the Tax Withdrawal Account we neglect global events that may affect the amount of tax a company is expected to pay. For example, seasons and the current unemployment rate will impact the Tax Withdrawal Account and should be considered. While we certainly could incorporate such numbers in our model, we have decided not to in order to keep the model as simple as possible.

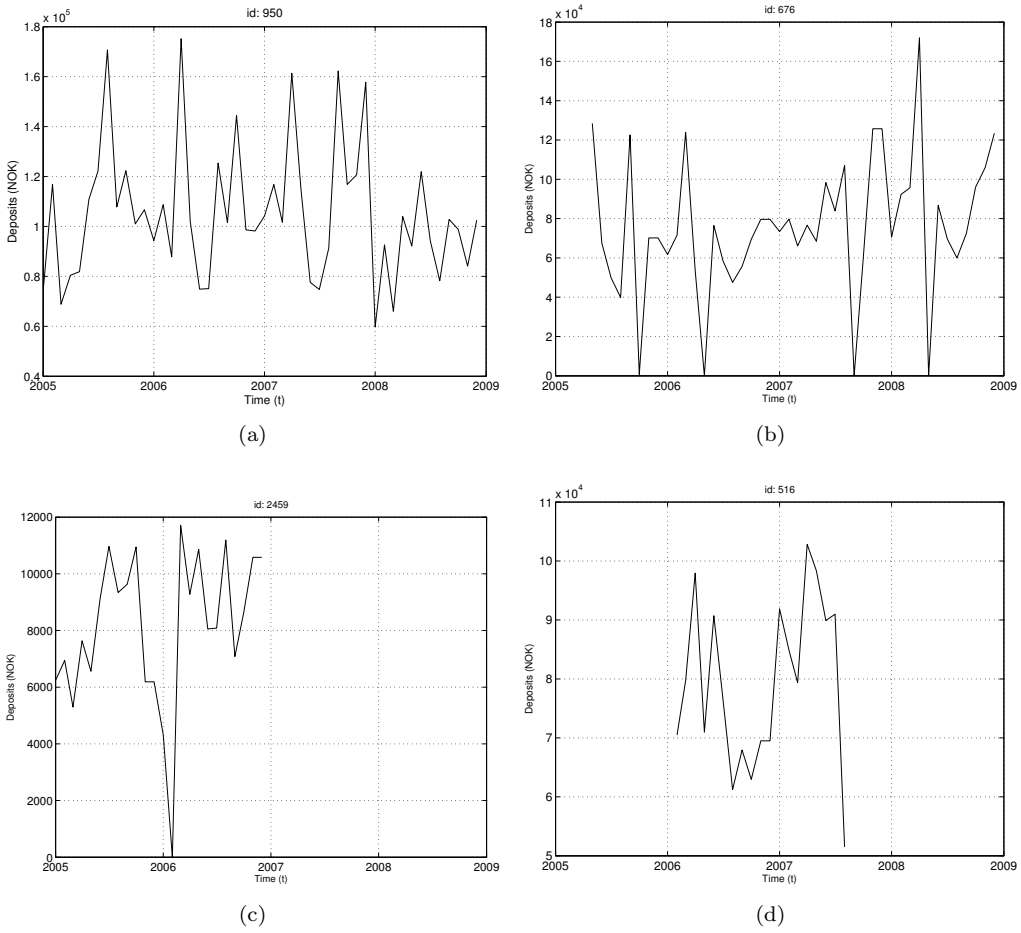


Figure 2.2: Four less trivial examples. In (a) and (b) we see deposits to the Tax Withdrawal Account for two healthy companies. In (c) and (d) we see deposits for two companies that defaulted and went bankrupt respectively. Only deposits up to the default and bankrupt is plotted. Note the deposits are $\times 10^4$ and $\times 10^5$.

3 The Dataset

The dataset used in this thesis is a collection of deposits and withdrawals from the Tax Withdrawal Account from 2005 to (and including) 2008 for various companies. All which are (or have been in case of bankruptcy) customers of SpareBank 1. For each company we have the following information available:

1. The company's Industry code[1] in the Brønnøysund register.
2. The company's Sector code in the Brønnøysund register.
3. Date of establishment
4. Monthly deposits to the Tax Withdrawal Account from 2005 to 2008
5. Monthly withdrawals to the Tax Withdrawal Account in the same period.
6. The bank's estimated credit rating for that company, also in the same period.
7. Annual revenues from 2005 to 2007. 2008 is not yet available.

Point 6 in the list above is the PD-ratings for the company estimated for every month. Both Point 4 and 5 consists of time series data. A *time series* is a statistical term for a sequence of data points measured at successive times, spaced often (but not necessarily) at uniform time intervals. Point 3–7 are used as inputs for the classification algorithm presented in Part I and II.

3.1 Training and Validation Data Sets

In order to make sure that the classifier is able to generalize[2], we divide the data set in two: a training set and a validation set. The purpose of the validation set is to provide new data to a tweaked classifier to ensure a performance measurement not biased by the effect of overfitting. Thus, the validation set is only used once. It is not used for tweaking the classifier, but to ensure that the classifier generalizes.

The training set is used to tweak parameters of the features and the classifier itself. When the classifier displays good performance on the training, the validation set is used to validate or invalidate the performance.

The validation set is constructed by removing 25 % of the healthy companies from the original data set, along with 25 % of the companies that defaulted or were the bank reckons a loss. This way we ensure an equal ratio of healthy/non-healthy companies in both the training set and the validation set. The validation set is only used *once*.

3.2 Class Distribution

The main problem with the dataset is the number of unhealthy companies. In the past four years (including 2008) only 137 out of a total 3,689 companies were considered as companies who caused a loss to the bank. After filtration of companies, which do not conform with the specifications on page 3, the number of loss-reckonings are considerably smaller. The companies are separated into three classes based on the PD-rating:

1. *J-rating*: The company has defaulted on a loan some time during the period.
2. *K-rating*: A loss where reckoned during the period.

[1] Norwegian: "Næringskode"

[2] That the classifier works in the general case and not just on the data we are working with

3. $PD(company) \subset [0, 1)$: The company has not defaulted and has not caused a loss for the bank.

Of course, a company may be in both Class 1 and 2 at the same time. Therefore, we treat Class 1 and 2 as the same class and Class 3 as a separate disjunct class. In machine learning terminology, this is called a *binary classification problem*. We shall refer to Class 1 and 2 as "unhealthy" companies and Class 3 as "healthy" companies. Table 3.1 shows the distribution of companies in Sub-sector 710 before applying restriction 2–4 in the Scope section, page 2.

Class	Training-Set	Validation-Set
Healthy	2,669	883
Non-healthy	99	38
Total	2,768	921

Table 3.1: Distribution of companies in Sub-sector 710 before filtration.

Applying restrictions 2–4 we get the following class distribution:

Class	Training-Set	Validation-Set
Healthy	918	287
Non-healthy	41	11
Total	959	298

Table 3.2: Distribution of companies in Sub-sector 710 after filtration.

We loose almost 2,000 companies by restricting our analysis to companies in the selected industries which are above a certain threshold. Performing filtration or not, healthy companies are represented by a far larger number of instances than non-healthy companies, resulting in a skewed distribution.

When one class is represented by a large number of examples while the other is represented only by a few, the dataset is *imbalanced* [Japkowicz et al., 2000]; a feature which is quite common in practice. When learning from imbalanced data sets, machine learning algorithms tend to produce high predictive accuracy over the majority class, but poor predictive accuracy over the minority class [Maloof, 2003]. We discuss how to tackle this problem later in Part I.

3.3 Noise and Phased-out Companies

There are two groups of companies in addition to those above: (1) Companies which do not have any record of defaults or bankruptcy, but which have been phased out and do not longer exist. (2) Companies that has received an erroneous PD-rating based on subjective measures.

These two classes of companies presents a problem for us. The phased out companies are in a grey area between healthy and non-healthy companies. They are clearly not non-healthy as they have always payed their debt and no bankruptcy or other evidence or economical problems are present. On the other hand, if no problems were present, the company probably would not have been phased out. After discussion with the bank, all companies which are phased out have been removed from the dataset.

While phased out companies can be handled, those which have been manually, and erroneously, marked with a J- or K-rating are worse. These companies are part of the already sparse set of non-healthy companies and may significantly disturb the classification. Fortunately, it is not impossible to identify such mistakes, to do this however, one need access to bank internals which we do not have.

Identifying mistakes in the PD-rating requires manual analysis of the company in question; it is not reasonable to analyse *all* the non-healthy companies for erroneous PD-ratings. Instead we tune the classifier on the noisy set of healthy and unhealthy companies. When a certain error-threshold has been met, we send the (hopefully small) set of false negatives[1] for further analysis by SpareBank 1. If some of the companies are erroneously marked they are manually re-classified and new parameters is evaluated for the classifier.

The process of removing erroneous rated and phased out companies was done after the training of the classifier (Chapter 6), but we present the final class distribution here for the sake of completeness:

Class	Training-Set	Validation-Set
Healthy	873	273
Non-healthy	31	7
Total	904	280

Table 3.3: Final distribution of companies

3.4 Samples from the Dataset

In order to give the reader a taste of what we are working with, we present some samples of healthy and unhealthy companies. In the following figures, defaults are marked with yellow dotted circles and loss-reckonings are marked with solid red circles. Deposits to the Tax Withdrawal Account is shown in the left column; withdrawals from the same account is shown in the right column.

A few comments is needed for the withdrawals plot. First, we only plot *actual* withdrawals from the Tax Withdrawal Account, since companies are only expected to transfer money from the account every second month (Section 2.1) there are usually only six points in the plot for each year. There are, however, some exceptions: Some companies transfer money from the account even though they are not expected to, also, some companies do *not* transfer money from the account even though they are expected to. These exception are also plotted.

[1] Companies that were classified as healthy, but which is labeled with `J` or `K` in the dataset.

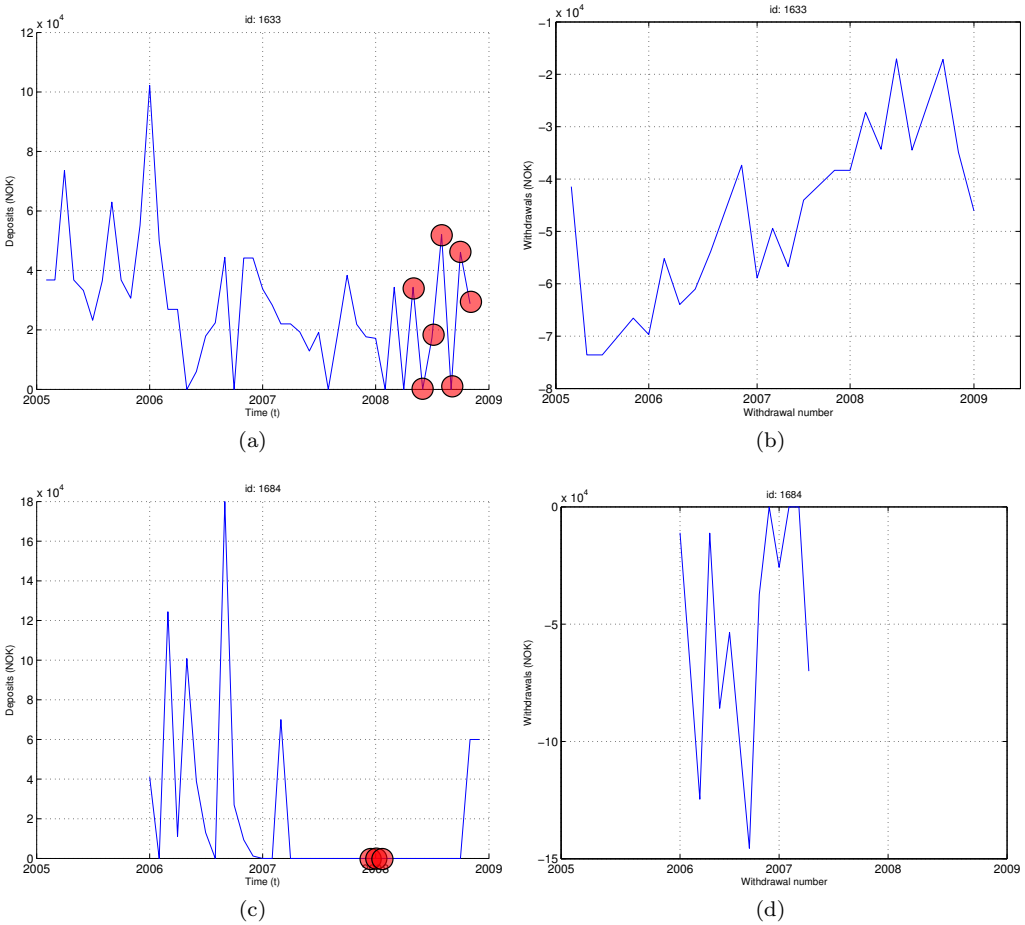


Figure 3.1: Two unhealthy companies. Notice in Figure (b) that there is a dominant up-trend. This means that the company is transferring less and less money to the government. In Figure (c) we see an example of a delayed loss reckoning. The company is not bankrupt as proved by the last deposit in 2008. Notice that three times around the shift from 2006 to 2007 the company does not transfer money from the account. More importantly, the company suddenly stops transferring money to and from the account—we have no explanation for such behaviour.

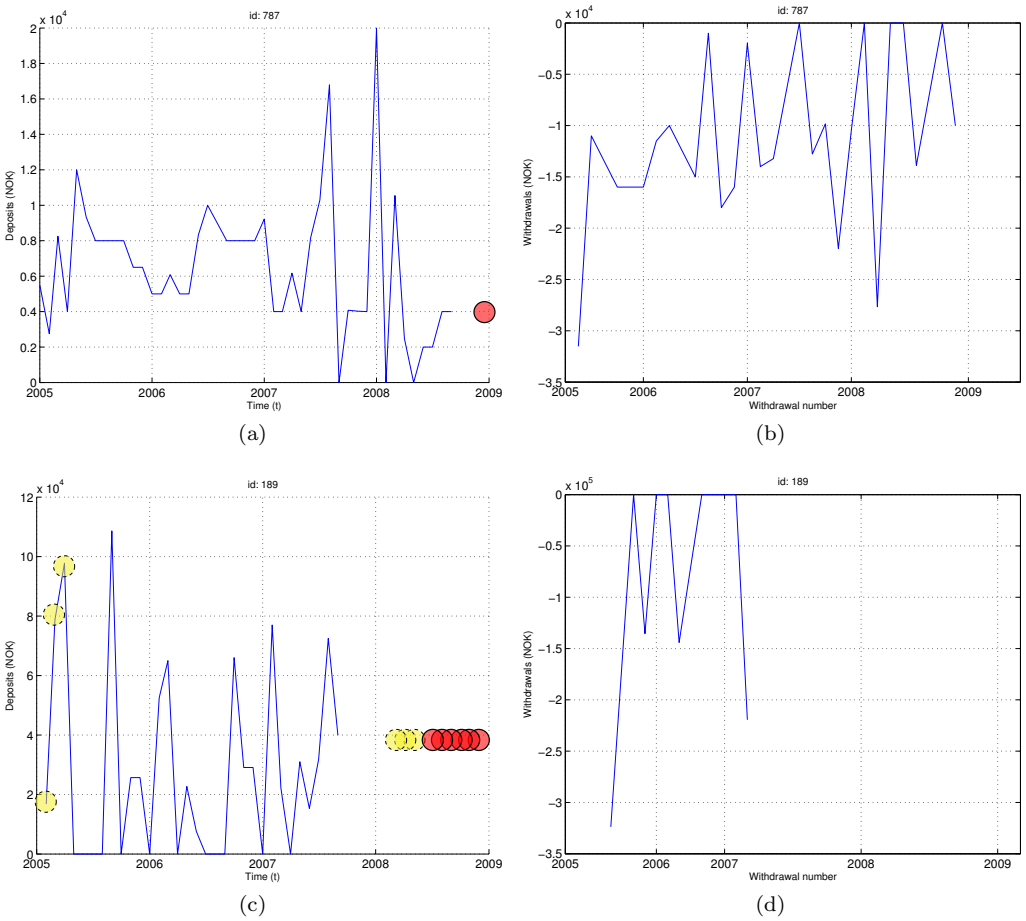


Figure 3.2: Again we see examples of delays before loss reckonings for both companies. Certainly, not *all* information about a company's health is reflected in the Tax Withdrawal Account. Notice in both withdrawals plot that several times the companies omit transferring money from the Tax Withdrawal Account.

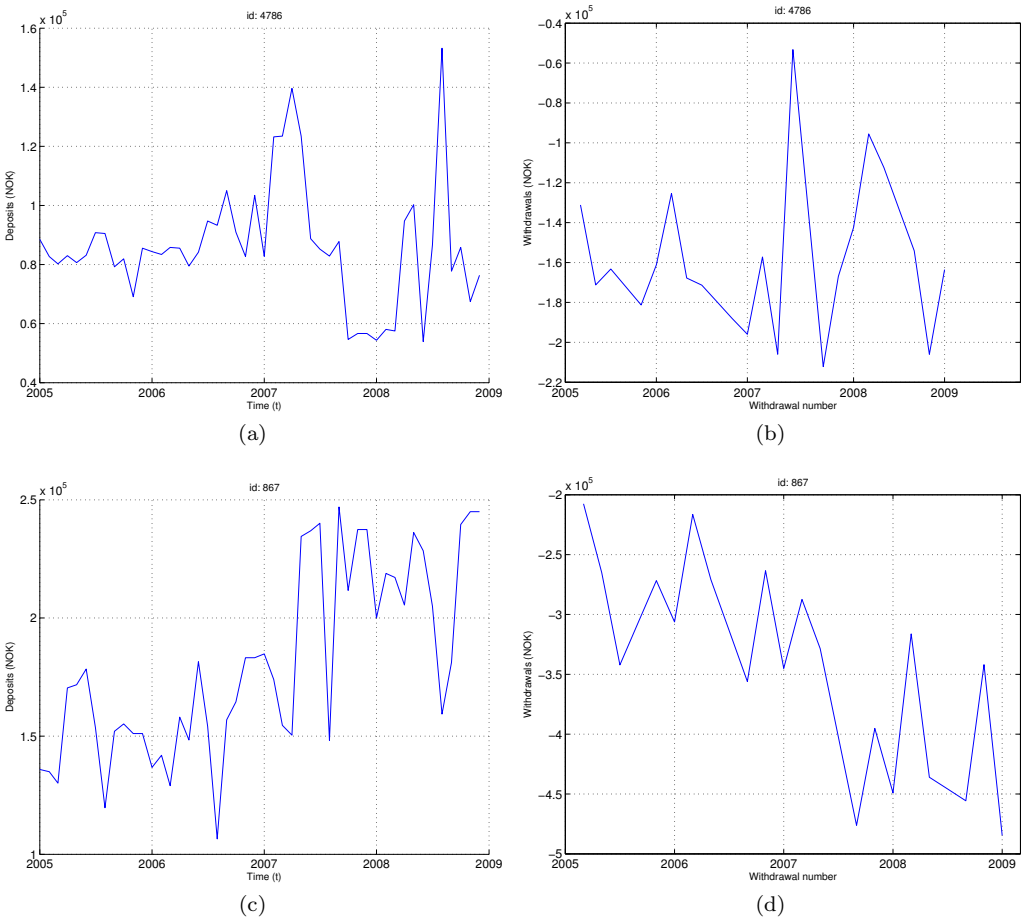


Figure 3.3: These companies are classified in the dataset at healthy: they have no registered defaults and there has not been reckoned a loss on any loans they have in SpareBank 1. In contrast to some of the non-healthy companies in Figure 3.1 and Figure 3.2, these companies (mostly) transfer money from the account when they are supposed to, and there are few zero-deposits.

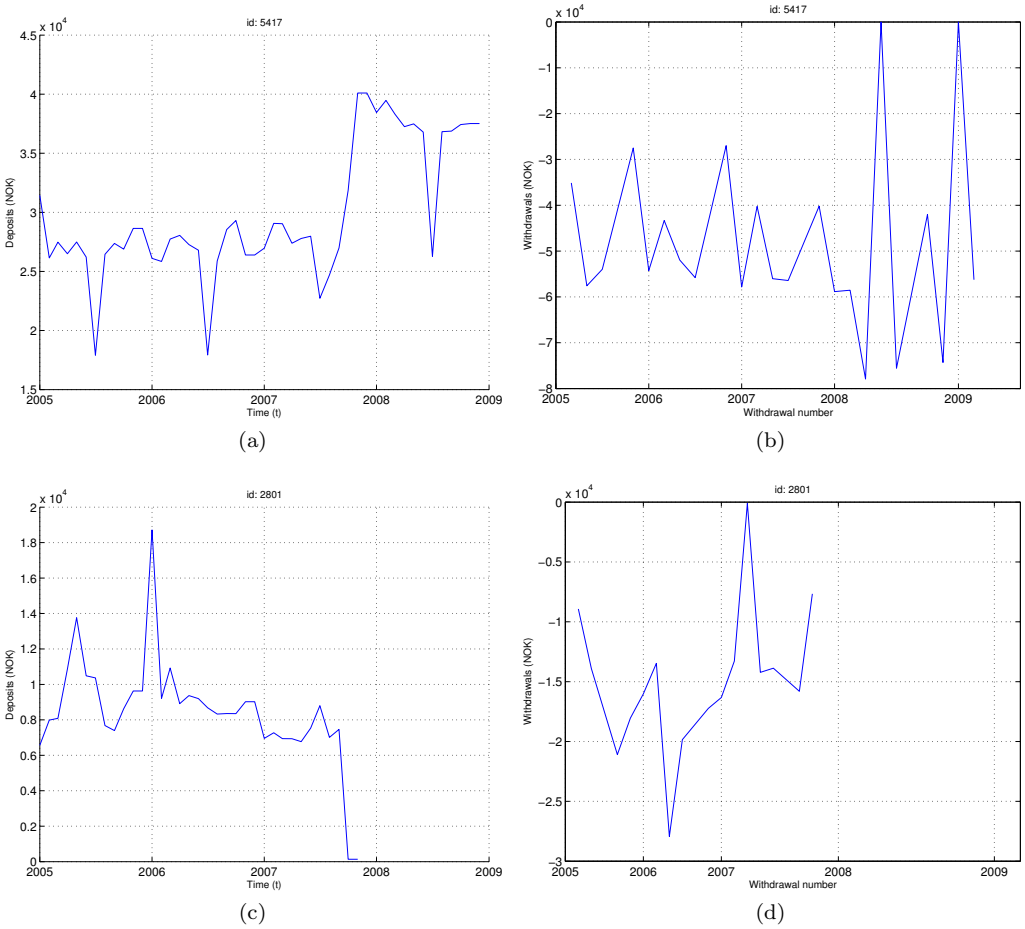


Figure 3.4: Here we present some exceptions to the rule. Both companies has some time during the period not transferred money from the Tax Withdrawal Account even though they were supposed to. The company in (c) and (d) is specially awkward in that the deposits and withdrawals suddenly stops (the company is not registered in the Brønnøysund registers as phased out). We can think of no good explanation for such behaviour, but it happens now and then.

I *Coarse-Grained Separation*

The heterogeneity of the dataset presents a problem when searching for *one* algorithm to solve the corporate bankruptcy problem. While this is in theory possible, we take a two-step approach. We first separate the obvious healthy companies from the rest to obtain a "grey area" of companies were the probability for a default or loss-reckoning is above a certain threshold; this grey area is then analyzed at higher resolution than the previous others. We perform the first step—the coarse grained separation—by using abstractions of the time series forming deposits and withdrawals. These abstractions are derived through an extensive domain-oriented data study resulting in a small finite set of features, see Section 5.1.

Figure 3.5: The different phases conducted in Part I

As presented in Figure 3.5, the feature generation phase is conducted after the preprocessing phase, which is discussed in detail in Chapter 4. The preprocessing phase consists of normalizing the data and adjusting them due to special tax rules. In the two latter phases, Chapter 5 to 7, we integrate the features by a classification algorithm to separate healthy companies from the rest.

4 Preprocessing the time series

Preprocessing is the process of altering a set of data prior to analysis. The goal of preprocessing is to transform a noisy set of measurements into a set of data values more easily and effectively processed by a later procedure. In this chapter we discuss how to exploit domain knowledge to smooth and adjust the curves forming deposits and withdrawals from the Tax Withdrawal Account in order to make them easier to analyse and later on classify. This includes incorporation of regulations in the Norwegian tax law as well as techniques for trend extraction and "safety buffers".

4.1 Holiday Tax Adjustments

The Norwegian tax law requires every wage-earner (with some few exceptions) to pay tax off their wages.[1] The amount of tax a worker is required to pay is distributed over 10.5 months due to the following rules:

- No tax is required off vacation pay in June/July. This accounts for *one* month of pay, however the employer is free to choose which month.
- Only half tax is required in December.

Since we have 12 months of deposits and withdrawals some correction is needed. By adjusting the raw data accordingly we achieve more accurate trend estimates later on. In the figure below we see an example of the results after adjusting from the rules in the list above.

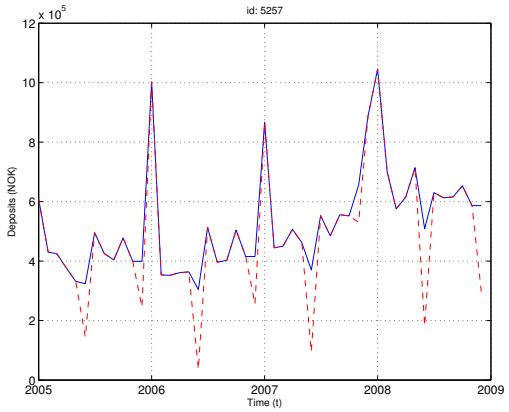


Figure 4.1: Holiday adjustments in action. The dotted red line is the raw data, the solid blue line is the result after adjusting. The result after adjusting is a smoother function.

We first adjust the vacation tax in June/July. If the company pays less tax in June than July we set $Tax_{June} = average(Tax_{May}, Tax_{June}, Tax_{July})$. If the company pays less tax in July, then the July deposits is set to the average of June, July and August. The deposits for December is adjusted similarly: $Tax_{December} = average(Tax_{November}, Tax_{December})$. Notice however that we do not use the average of November and January, this is because we have frequently observed spikes in January (probably because of the new year), see for instance Figure 4.1.

[1] Consult "Skatteloven" §2-1 for details on exceptions

The adjustments just outlined are not mathematically correct with respect to the tax regulations outlined in the list on the preceding page. For example, adjusting for December month should be achieved by simply multiplying by two. The problem is that not every company have as evident tax reductions as the company in Figure 4.1, consider for example the companies in Figure 2.2 on page 7. Multiplying the December deposits by two in these cases will result in distinguished spikes. By just averaging we achieve more robust adjustments.

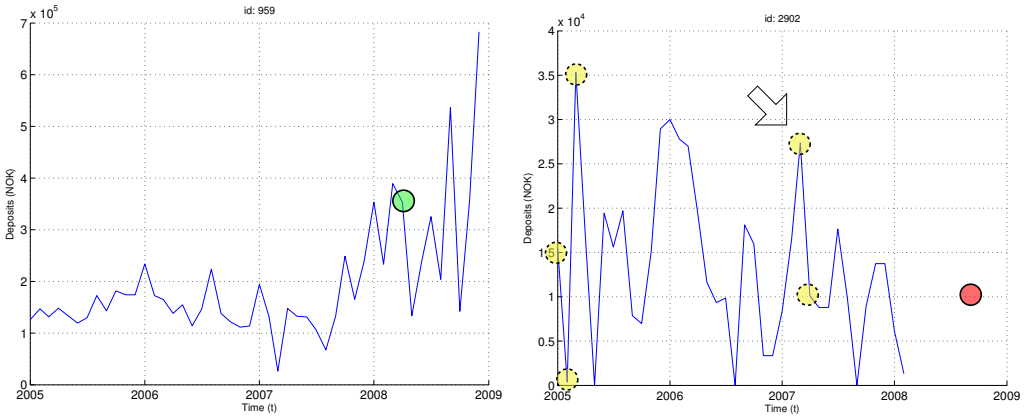
4.2 Safety Buffer

When analyzing the time series forming deposits and withdrawals we need to determine an interval to consider. This interval might seem obvious, but consider the following:

1. We only have data up to and including December 2008. This means that if a healthy company exhibits behaviour that indicates a default or loss-reckoning, but the actual loss or default occurs in early 2009, the company will be marked as "healthy" in the dataset. In this case, the classifier should mark the company as "will have a default or loss reckoned" and not healthy.

2. There is no reason to look at deposits/withdrawals data after a default or loss-reckoning since we want to recognize the patterns forming *before* a default or loss-reckoning. We therefore need to determine an interval for non-healthy companies to consider when extracting features.

3. In cases where several defaults has taken place we need to determine which default to use as an end of the interval to analyze. See Figure 4.2 (b) on this page.



(a) (b)

Figure 4.2: (a) Solid green circle marks end of the interval considered for healthy companies. (b) Defaults marked with dotted yellow circles, loss-reckoning marked with a solid red circle. The first yellow circle of the second defaults group is used as the start point of the safety buffer. Only data points up to, but not including, this point are used.

With respect to point 1 in the list above, we require at least six months without evidence of a default or loss reckoning in order for the company to be healthy. The start of this "safety buffer" is marked with a solid green circle in Figure 4.2 (a). When analyzing healthy companies, only data points up to and including the circle are used.

In Figure 4.3 on the facing page we see an example were parts of the time series are left out of the analysis. Only the points from January 2005 up to, but not including, the first dotted yellow

circle are analysed. By discarding data points after the first loss-reckoning or default we also discard all companies that only have defaults or loss-reckonings in the first half of 2005. This is because we require at least six data points for the analysis.

Regarding the last point in the list on the preceding page we have decided to use the first evidence received, after the required 6 months have passed, as the end point for the interval under analysis. Consider Figure 4.2, where defaults occurs in the first month of the time series. If we were to only analyze the data points up to the first default we would have too few data points for a proper analysis (in this case none). Instead we use the time series up to the next "block" of evidence of defaults (early 2007). Specifically, the first yellow circle in 2007 is used as the end point of the interval to analyze.

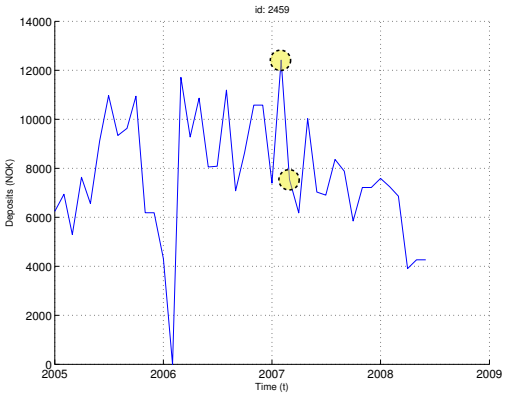


Figure 4.3: An example of a time series where data points are left out of the analysis. In this example we only use the data point up to (but not including) the first dotted yellow circle.

4.3 Normalization

In order for the classifier to compare deposits and withdrawals for different companies some normalization is needed. All deposits and withdrawals are normalized to the range $[0, 1]$, this ensures that trends are described the same regardless of whether a company pays 100,000 a month or 1,000 a month in taxes.

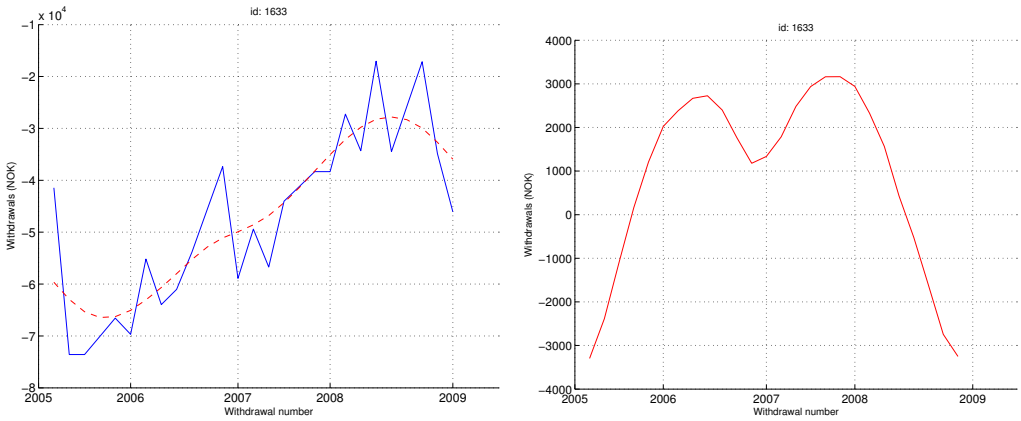
There is nothing fancy about the normalization process. To normalize a timeseries ts to the range $[0, 1]$—be it deposits or withdrawals—we simply divide by the maximum value:

$$n(ts) = \frac{ts}{max(ts)}$$

4.4 Identifying Trends

A trend is a *prolonged period of time where deposits or withdrawals rise or fall faster than their historical average* [Turner, 2007]. Identifying trends in a function f is usually done in two steps: (1) Filter f to obtain a function g that express "roughly" the shape of f. (2) Analyze the first difference of g: trends exists in intervals where the sign of the first difference is constant. Of course additional constraints may be added, such as the minimal amount of data points in a trend. See Figure 4.4 on the following page for an illustration.

We use local and global trends as part of the model explained in the next chapter. There is not *one* specific way to estimate a trend given a set of data points; in our context, the following characteristics are important for the algorithm identifying trends:



(a) (b)

Figure 4.4: (a) Withdrawals from a Tax Withdrawal Account. Actual withdrawals are plotted with solid blue, a filtered version is plotted in dotted red. (b) The first difference of the dotted red function in (a). Notice that the sign of the first difference is positive from the end of 2005 and half way through 2008, hence a trend exists in this period.

- The algorithm must be able to recognize several trends given a set of data values. For instance, a time series may start in an up-trend and end in a down-trend. Such cases are interesting when determining whether there exists a global trend or not.
- The algorithm should not be overly sensitive to short term fluctuations. I.e. spikes should not disturb the trend.

Next, we discuss different ways of fitting a trend to a set of data points.

4.4.1 Least Squares

In the most generic case, fitting a trend is done with the least squares method: Given a set of data points X, and data values Y, one choose values for a and b so that

$$\sum_X([ax_i + b - y_i]^2) \qquad x_i \in X, y_i \in Y \tag{4.1}$$

is minimized. We minimize the sum of offsets—the residuals—of the points from the curve. Because every point is weighted equally, outlying points may have a disproportionate effect on the fit. The result is a straight line describing the major trend.

4.4.2 Moving Average

A *moving average* is used to analyze a set of data points by creating a series of averages of subsets of the full data set. The *simple moving average* (SMA) of size n is the unweighted mean of the past n data points. Formally, given a data set X:

$$SMA_{x_i} = \frac{x_i + x_{i-1} + x_{i-2} + \cdots + x_{i-n-1}}{n} \qquad x_i \in X$$

we calculate the mean of the previous n points for every $x_i \in X$. By using the sign of the first derivative of the SMA we can extract several trends from a single time series (change of sign equals change of trend).

4.4.3 Fourier Analysis

By analyzing the frequency domain of a signal—deposits and withdrawals in our case—we can make certain qualitative assertions about it. The low frequency components are especially interesting as they contain information about the general curvature of the signal, ignoring abrupt variations.

In the discrete case we have a collection of points (x_k, y_k) and a total of K points. Let x_k denote month and y_k the deposit to the Tax Withdrawal Account. Then for all points in the function we define a complex number $s(k) = x_k + jy_k$, where j is the imaginary unit. The Discrete Fourier Transform (DFT) is defined as:

$$a(u) = \frac{1}{K} \sum_{k=0}^{K-1} s(k) e^{-j2\pi uk/K}$$

for $u = 0, 1, 2, \ldots, K-1$. $a(u)$ is referred to as a "Fourier component" and low frequency means low values for u. The low frequency components carry general information about the signal, while high frequency components accords for details. By analyzing the slope of the low frequency components we can make assertions about trends in the deposits or withdrawals.

4.4.4 Hodrick-Prescott Filter

The Hodrick-Prescott filter is a mathematical tool proposed in [Hodrick and Prescott, 1997] and used in macroeconomics. It is used to produce a representation of a time series that is less sensitive to short term fluctuations than long term.

Let Y be a time series, the Hodrick-Prescott filter assumes that Y can be expressed as a sum: $y_t = \tau_t + c_t$ for all $y_t \in Y$. Here, τ is the "trend" component of the time series and c is the "cycle" component. The filter is parameterized by a λ-parameter controlling its sensitivity to short term fluctuations. Formally, given an adequately chosen positive value for λ there exists a value τ that will minimize the following function:

$$\min \sum_{t=1}^{T} (y_t - \tau_t)^2 + \lambda \sum_{t=2}^{T} [(\tau_{t+1} - \tau_t) - (\tau_t - \tau_{t-1})]^2 \qquad y_t \in Y \tag{4.2}$$

Usually the logarithm of the data points Y is used instead of the actual values. The cycle component c is obtained simply by subtracting τ from y: $c_t = y_t - \tau_t$. The right part of Equation 4.2 is a way to formulate the second difference for a discrete function. The second difference penalizes abrupt changes in the trend-function and the λ parameter adjusts the effect on the overall minimization problem.

4.4.5 Choosing a Method for Identifying Trends

The least squares method in its original form is only appropriate in cases where you always expect that the time series in question is in fact trending, and that there is only *one* dominant trend. Neither of this is true for our case so the least squares method is discarded. The reason for the least squares inability to detect several trends is because the linear polynomial used in Equation 4.1. By using a non-linear polynomial the least squares method will in theory be able to detect several trends. It is however not very robust to determine such a polynomial, meaning one polynomial will not fitt all functions very well.

The Simple Moving Average is able to detect several trends in a time series, but since each point is equally weighted the fit might be disproportionally affected by single points. This can be solved with a "weighted moving average". The amount of smoothing by the moving average filter is controlled by the size parameter, i.e. how many points are considered when averaging. The more points, the more smoothing.

In Figure 4.5 on the next page the moving average (dotted red) is plotted for the deposits (solid blue) for a company. Here we have smoothed the original function by averaging the past

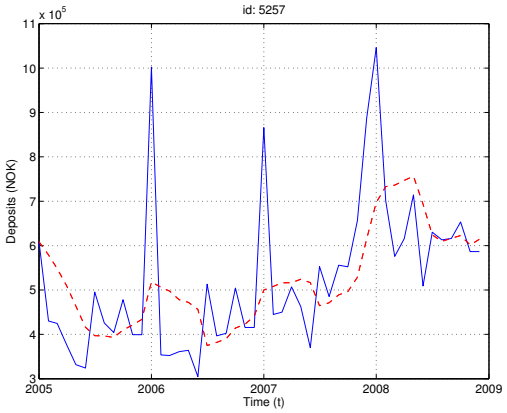


Figure 4.5: The solid blue line denotes deposits to the Tax Withdrawal Account; the dotted red line is the Simple Moving Average (SMA) of the deposits. The size of the filter is 6 months.

six months for every data point. This requires at least six months of data for the smoothing to have maximum effect; we let the data points before 2005 equals the values at January 1st, 2005, in order for the smoothed function to be defined over the entire interval. Considering the figure we notice three important things:

- There is a trend from mid-2006 to 2008. It is not possible to identify this trend by analyzing the first difference of the moving average since the original function is not smoothed enough.
- It is not reasonable to increase the size of the filter as this would require too much data.
- The moving average is "lagging". Notice that the tops and valleys in the moving average is delayed from the original function.

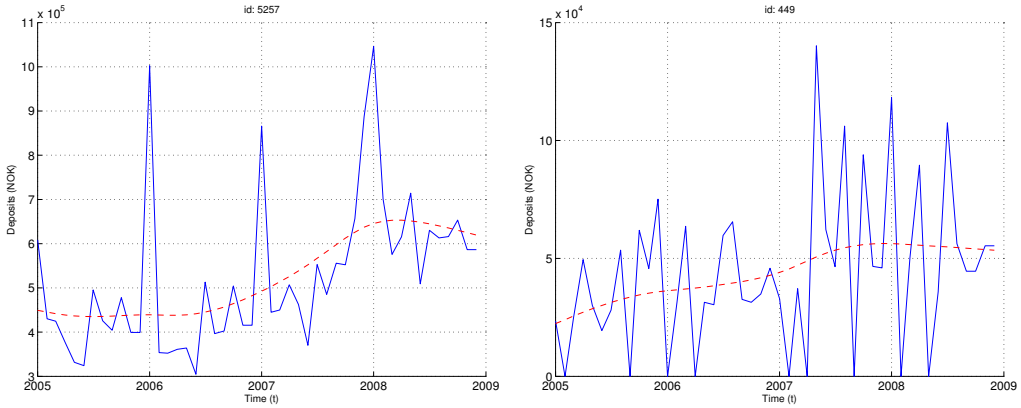
The lagging behaviour of the moving average filter is proportional to the size of filter; this is another reason to keep the size low. The lagging behaviour will cause a delay when flagging bankrupts. Thus, because of insufficient smoothing and the lagging we discard the use of moving averages.[1]

The Hodrick-Prescott filter suffers from none of the problems mentioned so far. The λ-parameter in Equation 4.2 controls how sensitive τ is to high frequency components (abrupt variations) in the original function. In other words, λ controls how smooth τ is.

In Figure 4.6 on the facing page we see two examples of the Hodrick-Prescott filter in action: Figure 4.6 (a) is equal to Figure 4.5. We see that the Hodrick-Prescott filter successfully identifies the trend from mid-2006 to early 2008 (the sign of the first difference is the same throughout the period). In Figure 4.6 (b) we see an example were the filter identifies less obvious trends. According to the first difference of the output, the deposits (solid blue) is constantly in an uptrend until the end of 2007. If we consider only the data points where actual deposits where made (ignoring zero-deposits), this might well be true. But when considering zero-deposits, it is more questionable whether a trend exists or not. Notice that the function is never zero over a period of time, zero deposits are only caused by short term fluctuations which are exactly what the Hodrick-Prescott filter is designed to be less sensitive to.

Fourier descriptors—like the Hodrick-Prescott filter—provides a well smoothed, correctly shifted trend in accordance to the original function. The only problem is that the amount of deposits and withdrawals varies from company to company (depending on whether the company went bankrupt or not). This makes it harder to choose the optimal amount of low frequency

[1] The "Weighted Moving Average" will perform better with respect to both lagging and smoothing, but will eliminate neither of the problems.



(a) (b)

Figure 4.6: Two examples of the Hodrick-Prescott filter (dotted red, $\lambda = 500$). The original function is plotted with solid blue. (a) Previous example with Hodrick-Prescott filter instead of Moving Average. (b) Deposits to the Tax Withdrawal Account for a non-bankrupt company where the Hodrick-Prescott filter perform less well.

components to consider. While not impossible, due to the success of the Hodrick-Prescott filter, we have discarded Fourier descriptors.

5 A Model for Coarse-Grained Classification of Companies

As stated in Chapter 3, our dataset consists of time series data describing deposits and withdrawals to, and from, a company's Tax Withdrawal Account. These time series are so called non-stationary: their statistical properties depend on time. Hence, we have highly varying and shifting time series with trends, seasonal changes and other irregularities, such as large month-to-month changes. Most of the irregularities and variations originate from unknown systems, for instance accounting methods, and does not necessarily reflect the economical state of the company. By including the time series in our model as is, we rely entirely on the classifiers ability to neglect the irrelevant information, and focus on the relevant information in the time series.

Additionally, the length of time series usually varies from series to series. This difference in dimensionality is incompatible with the majority of classifiers, which only accepts input of the same dimensionality. In this chapter we will present an approach that handles not only the issue with dimensionality but also the issue with information overload in the time series. We perform a domain-oriented data study resulting in a finite set of features. The features represent only the relevant information in the time series. All of the features, introduced underneath, are derived from discussion with experts within the credit risk field. By assuring that the features are backed by experts within the domain, we decrease the potential for overfitting. The features, being a compact representation of the time series, enable the use of a simple algorithm for classification.

In Part II of this thesis we will, despite its complications with information overload and dimensionality, explore an approach performing classification over the raw non-stationary time series. To tackle this task we will bring in the statistically heavier method of Gaussian Processes and see how it performs on the data.

Another approach, commonly used for classifying non-stationary time series, involves using the Box-Jenkins approach presented in [G.E.P. Box and Reinsel, 1994]. The Box-Jenkins approach applies Autoregressive Integrated Moving Average (ARIMA) models to find the best fit of a time series to past values of this time series. These models are then used to forecast the future values of the time series. Even though the Box-Jenkins approach is quite popular, especially in the field of econometrics, we have not explored this approach in our research due to the requirement of sufficiently long series. [Chatfield, 1996] recommends at least 50 observations. Many others would recommend at least 100 observations, while we only have 48 observations (monthly data for four years). Additionally, the Box-Jenkins approach is not a complete solution to our problem. The forecasted time series still need to be analyzed to decide whether or not a company is financially sick; a non-trivial task.

In the following section we will walk thoroughly through the features we derived via our data study, before we see how a simple classifier performs on the abstracted dataset. Recall, as stated in Chapter 3, the dataset we use is imbalanced. To tackle this imbalance, we use a cost-sensitivity inducing wrapper around the classifier, called MetaCost [Domingos, 1999]. What a cost-sensitive classifier is and how we benefit from it will be discussed in Section 5.4.

5.1 Feature Generation

Feature generation means creating a set of features based on the original data. From this set of features only the best performing features are selected. In order to identify such features we have conducted a domain-oriented data study, which is the main contribution from Part I of this thesis. For a study to be considered as domain-oriented, it is required that the findings are backed by

domain knowledge. The objectives of this study has been to extract features and select a finite set of discriminating feature functions that represents the relevant information in the time series, compactly. A "discriminating feature function" is identified as a feature function that separates the two classes of companies well. Additionally, a relevant feature will discriminate the two classes entirely based on information that we reckon ought to separate the classes. The relevance of the feature has been assured through extensive discussions together with experts within the credit risk field, currently working at SpareBank 1's own research department for credit risk models. Selecting only relevant features decreases the possibilities for overfitting, since the features are backed by knowledge not only learned from the small world composed by our dataset.

We have two ways of extracting features: a few features are extracted from company metadata, while the rest of the features process the time series associated with the company. The latter can be further divided into two groups, namely local and global time series features. The difference is, as the name indicates, the former type of features only processes a window, the last n months from current date, while the latter feature type process the whole time series, from the beginning of the time series to the current date. In contrast to the differences between the features, they all return a real number. Overall, the feature functions can be summarized as

$$r = \text{featureFunction}(\textit{company}, n)$$

where *company* is a structure encapsulating all data regarding a company, n is the number of months to look back (n will be ignored by features not concerned with the time series), and r is a real number with different meaning for each feature function. The following list enumerates the proposed features:

1. Local Deposit Trend
2. Local Withdrawal Trend
3. Global Withdrawal Trend
4. Global Deposit Trend
5. Global Balance Trend
6. Consecutive Zero Deposits
7. Consecutive Zero Withdrawals
8. Consecutive Zero Balance
9. Zero Deposits
10. Zero Withdrawals
11. Age
12. Company Size

Feature 1–10 abstracts the time series representing deposits and withdrawals, while the last two features, Company Size and Age, are extracted directly from the company metadata. The time series are mainly described in two ways, by various trends and by various counts of incidents when a company refrains from making a deposit or withdrawal. An performance evaluation of the features will be comprehended in Chapter 6 and discussed in Chapter 7.

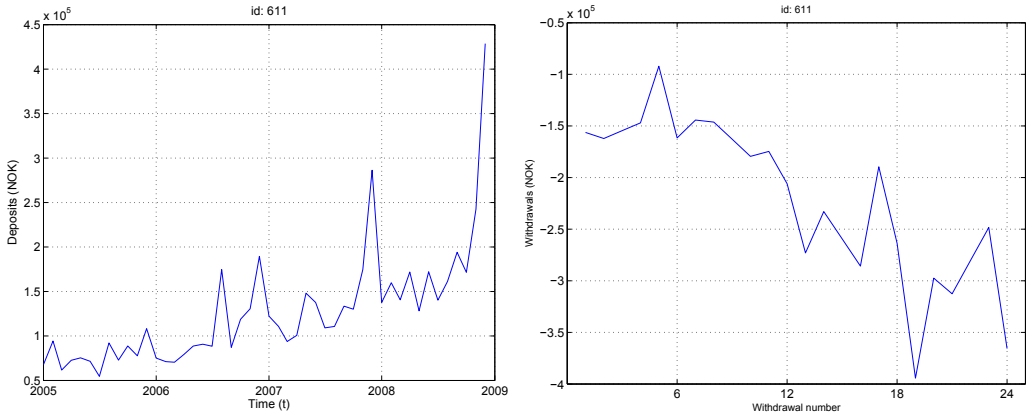
5.1.1 Extracting Trend Features

What a trend is, and how we can extract the trend component from a time series, have already been covered in Section 4.4 on page 21. In this section we analyze the trend component and output a real number summarizing the relevant information regarding the trend.

First, let us explore the rationale behind analyzing trends in time series describing transactions to and from the Tax Withdrawal Account. Our hypotheses propose that the observed transaction trend is a mirror of the financial situation in the company. Consider a company in growth. By hiring more people and rising wages to employees, the company pay increasingly more and more wages. Due to the increasing amount of paid wages, a corresponding increase with respect to the amount of tax deposited to the Tax Withdrawal Account is expected. A tightly coupled negative

trend is expected to be manifested in the withdrawals from the same account, since the money deposited needs to be further transferred to the government. See Figure 5.1 for an example of a financially healthy company with both positive deposit trend and negative withdrawal trend.

A financially sick company will seldom increase wages or workforce, so a positive deposit trend, or a negative withdrawal trend, is less likely to happen. More likely, a negative or stagnant trend is observed when studying the deposits. Both types are believed to be a sign of financial weakness in the company. A negative trend may indicate that the company is lowering wages or getting rid of employees, while a stagnant trend may be conceived as slightly negative sign when adding inflation to the picture; the employees receive the same wages, while the prices are increasing.



(a) (b)

Figure 5.1: (a) Plot of deposits by Company 611 over the four year period. A strong positive trend is revealed. (b) The opposite trend is observed when plotting the actual withdrawals done by Company 611 over the same period.

It should be noted that there is a substantial amount of uncertainty tied to the theories presented in this section. For instance, a negative deposit trend may as well be a result of a couple of highly paid senior employees going into retirement. Additionally, a company getting rid of a bad-performing department is not necessarily a sign of bad health, but may as well be a sign of a healthy company being focused on cost efficiency. Notice, the examples above are all examples of relatively short-term trend movements. The retired seniors may soon be replaced by younger employees, and the absence of the bad-performing department may give room for new positions in departments that are profitable. As stated in [Gordon and Rosenthal, 2003], growth is imperative for firms in the capitalistic world. This observation, that economical healthy companies may suffer a short term negative tendency but economically growth in the long run, calls for different trend features observing trends over different time periods. Only studying short-term trends are not sufficient.

Local and global features, as introduced earlier in this section, are utilized to cope with the demand of trend features observing trends over different time intervals. Global trend features, such as feature 3–5 in the list above, extracts a single global trend over the whole time series, until current date. Local trend features extracts the trend prevailing in the last n months, from current date.

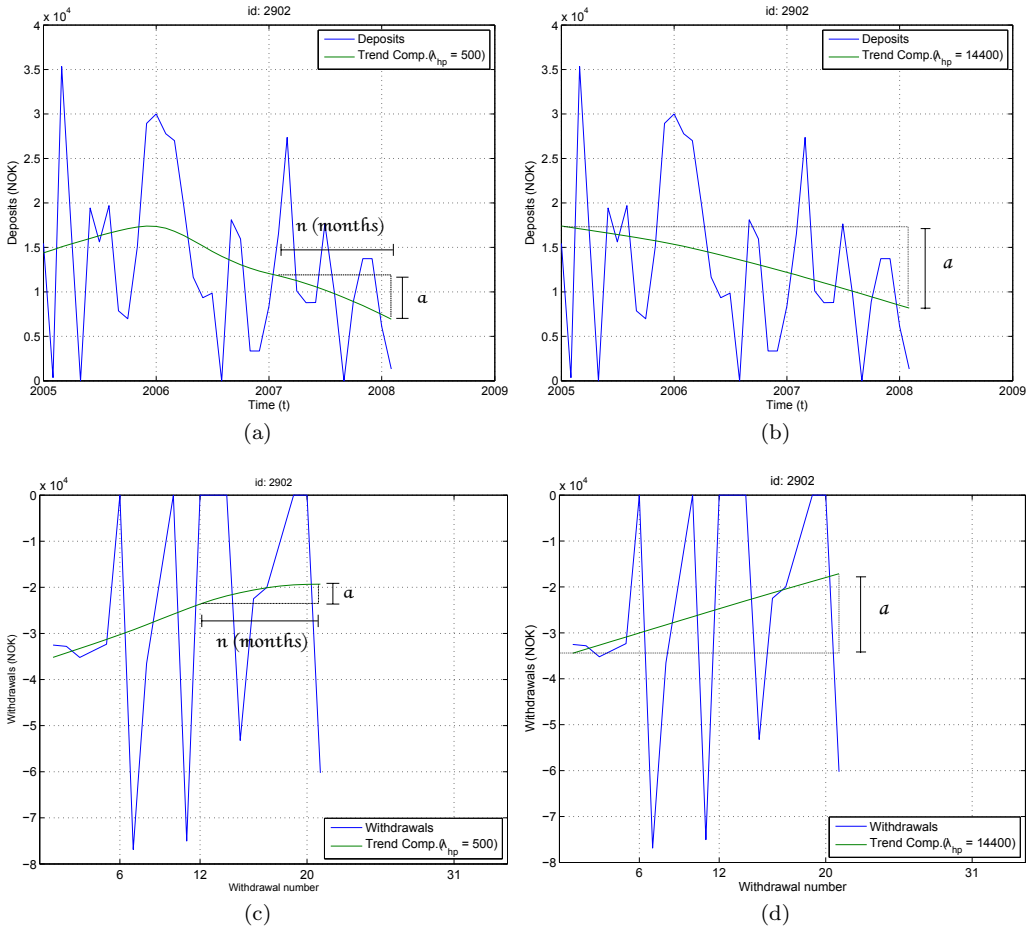


Figure 5.2: (a)–(d) shows deposits and withdrawals from/to Company 2902's Tax Withdrawal Account. In (a) and (c) the local trend is extracted using a Hodrick-Prescott filter with $\lambda = 500$. From a window of 12 months we calculate the slope, a, of the trend. In (b) and (d) the global trend is extracted using a Hodrick-Prescott filter with $\lambda = 14400$.

There are two trend measurements of interest, whether the trend is negative or positive, and the strength of the trend. Fortunately a single metric covers both points of interest, namely the slope. A negative slope indicates a negative trend, and vice versa. The magnitude of the slope indicates the strength of the trend. In Figure 5.2 on the preceding page, global and local trends are extracted from time series describing both deposits and withdrawals. Which value to use for the smoothing parameter, λ, is decided due to the frequency of the data. Our time series have a frequency of one month. While [Ravn and Uhlig, 2002] suggest using $\lambda = 129600$ for monthly data, the "industry standard", or macro economical best practices, uses $\lambda = 14400$. We suggest using $\lambda = 14400$ for global trends, see Figure 5.2(b) and (d), and $\lambda = 500$ for local trends, see Figure 5.2(a) and (c). Notice, by using a lower λ when extracting local trends we extract a less smoothed trend component, following the time series more, compared to the global trend component. This lower λ is necessary in order to extract a trend that may change direction several times during our four year period.

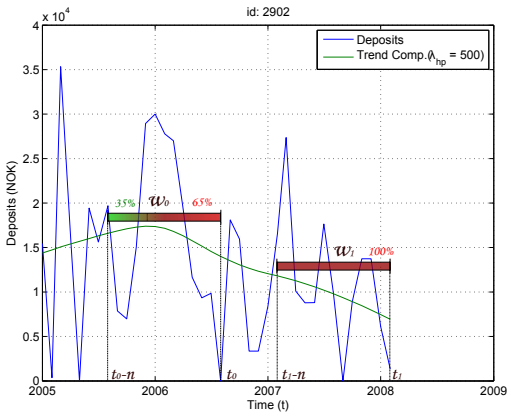


Figure 5.3: Here two different time windows, w_0 and w_1, are displayed. Notice in w_1 how the trend is negative during the whole period. The story is different in w_0, which contains two different trends. In 35% of the period there is a positive trend present, which then turns into a negative trend. Due to our trend-ratio requirement we need at least a presence of 80% for one trend type in order to report a trend. Thus, the local deposit trend feature will return $a = 0$ over w_0.

The low λ used when extracting local trends poses an issue when the current window is located over an extremal point of the trend component. At an extremal point the trend direction switches. To handle these cases we have defined a trend ratio threshold, demanding a presence of 80% of the time for one trend in the current window. If no trend has a presence of 80% of the time in the window, we say the trend is horizontal and returns a slope equal to zero. See Figure 5.3 for an example of how the trend ratio threshold impacts what we appreciate as a trend.

The last trend feature is the global balance trend, which is extracted in the same manner as the global deposit and withdrawal trend. This feature captures whether or not the company builds up reserves on the Tax Withdrawal Account. If a company has a negative global balance trend, it means that the company is paying more to the government than it is depositing to the Tax Withdrawal Account. This may be considered as a negative sign, since the practice can only sustain as long as there still are reserves left on the Tax Withdrawal Account.

5.1.2 Counting Zero-Transactions

We have five features, feature 6–10, detecting whenever a company omits transferring tax to the Tax Withdrawal Account (deposit), omits transferring tax to the government (withdrawal), or whenever the balance of a company's Tax Withdrawal Account is zero. The two latter features

counts every incident of zero deposits and withdrawals, while the three former counts the longest series of consecutive zeros in the time series.

The most important of the zero-transactions is the zero withdrawal, in other words, the incident of skipping a transfer of tax to the government. As introduced in Section 2.1 a company is liable to transfer tax to the government every second month. Omitting to do so may indicate serious liquidity issues within the company. A one time incident of skipping a withdrawal is serious enough, but skipping several consecutively payments is even more interesting to detect. It may indicate the beginning of the end, meaning the company may have stopped paying wages. Such incidents is captured by the consecutive zero withdrawal feature. See Figure 5.4(b), where a company skips several withdrawals and goes into bankruptcy during 2008.

There is usually a correspondence between a zero deposit and a zero withdrawal; the company did not deposit any money to the Tax Withdrawal Account thus there is not enough funds on the account to follow up on its obligations to the government. This link is not always a fact. Zero deposits are more frequent than zero withdrawals since companies are not liable to deposit every month, as long as they keep the balance on the Tax Withdrawal Account on a high enough level to cover the usual withdrawals. Hence, companies are allowed to skip deposits without breaking the law. If a company omits depositing tax to the Tax Withdrawal Account several times in a row, the same argument as for withdrawals applies; it may be the beginning of the end. Again, such events is detected by the consecutive zero deposits feature. Figure 5.4(a) and (c) depicts two different companies omitting transferring tax to their Tax Withdrawal Account.

The rationale behind the consecutive zero balance feature is the following: keeping funds on the Tax Withdrawal Account may indicate sloppy economical management in a company since it is suboptimal. The interest are very low, compared to regular corporate bank accounts, and the funds deposited to the account may not be used for anything else than paying taxes. In Figure 5.4(d) the consecutive incidents of zero balance on the Tax Withdrawal Account precedes company 1977's defaulting on loan obligations towards the bank.

5.1.3 Extracting Age and Company Size

The Age and Company Size features are included with the following rationale: The older and larger a company is, the smaller the probability is for a company to default on a loan obligation, or to go into bankruptcy. A company with some years on the track has already passed the test of time. Start ups, on the other hand, have been initiated with only a theory that there is money to be made from their product. Hence, young companies seem to be more risky than older companies. The Age feature is extracted as the number of days since the date it was established. In Figure 5.5(a) we present a cumulative plot showing the age of financially sick companies. The average age for all companies, meaning the average period from a company's establishment day until today, May 7th, 2009, is 12 years ($\frac{4494}{365} \approx 12$). By studying the plot we observe that approximately 23 out of totally 31 sick companies are on, or below, the average age. This fact supports the rationale behind the age feature.

The size feature is derived from annual income numbers. Specifically, we calculate the size feature as the mean of yearly income over the last three years. Companies of larger size generally have its own accounting department, best practices for how to do business, and larger margins before going into periods with financial struggle. Hence, the size feature is included in our model. The rationale behind the size feature is emphasized in Figure 5.5(b). Notice that 25 out of 31 financially sick companies have a yearly income lower than 5 million NOK. The average size over all companies is at 20 million NOK.

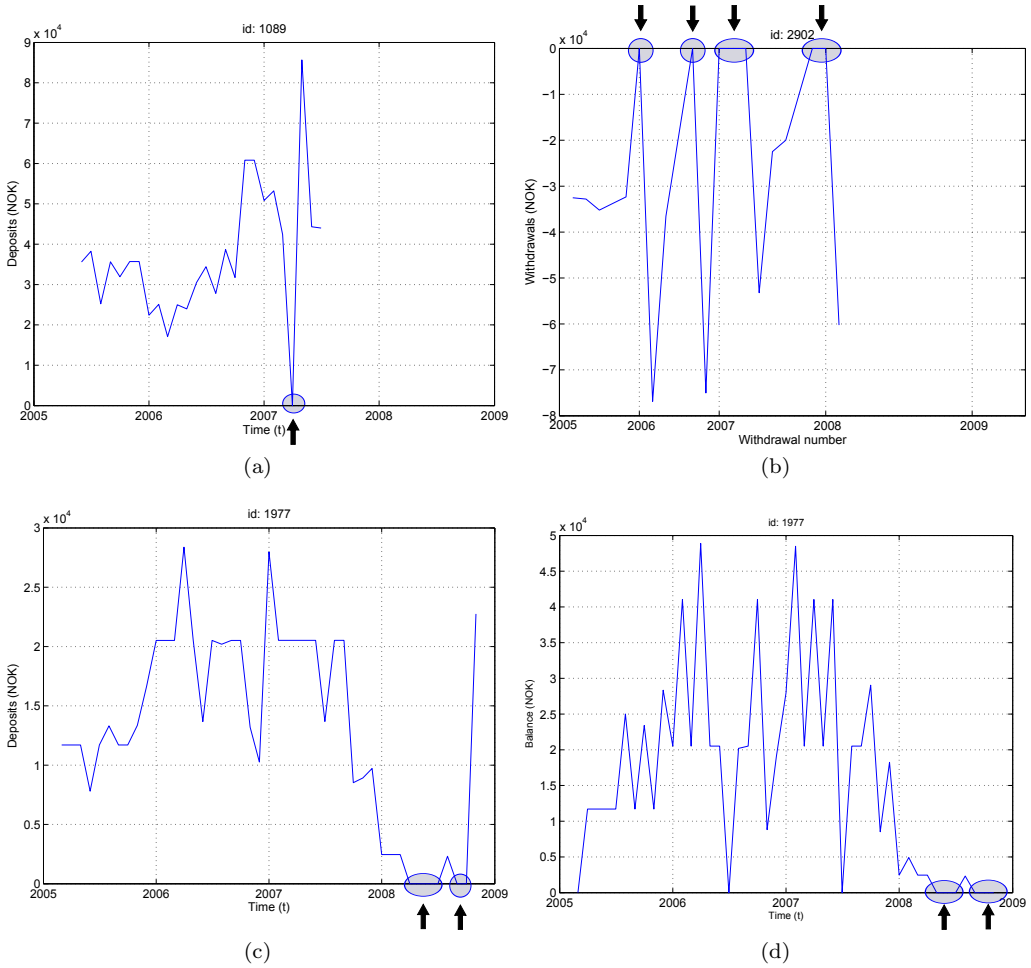
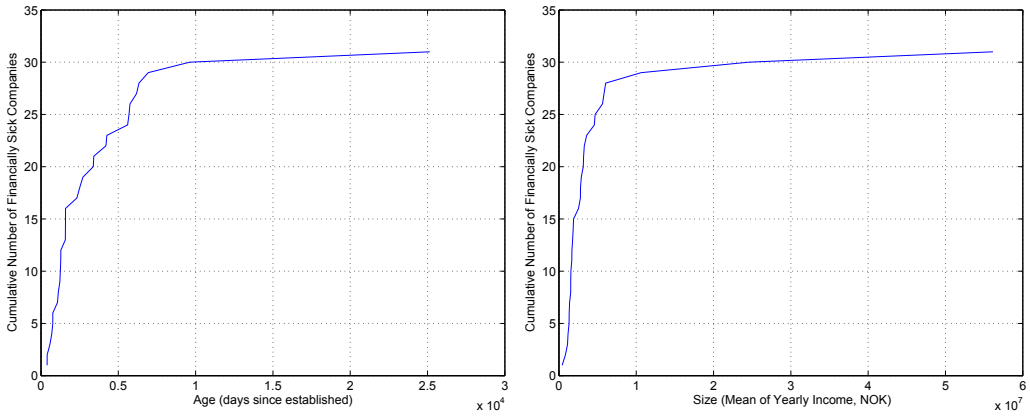


Figure 5.4: (a)–(d) displays plots of deposits, withdrawals, and balance associated with different companies' Tax Withdrawal Account. Incidents of zero transaction are marked with a blue circle. When more than one incident of a zero transaction occur consecutively, we mark it with a blue ellipse. In (a) company 1089 skips a single deposit. In (b) company 2902 first skips two withdrawals in 2006. Then, in two different periods in 2008, the company have consecutive incidents where it omits to transfer tax to the government. Company 2902 was marked as bankrupt by SpareBank 1 in September, 2008, but defaulted several times on their loan obligations throughout 2007. In (c) company 1977 omits depositing tax to the Tax Withdrawal Account during 2008. The company defaults on its loan obligation in December, 2008. As seen in (d), the same company depletes the balance of Tax Withdrawal Account during late 2008.



(a) (b)

Figure 5.5: Two plots illustrating the rationale behind size and age features. In (a), age is plotted for the financially struggling companies in our training set. The y-axis represents a cumulated count of companies, while the x-axis represents the age. In (b), a similar plot regarding the company size is presented. The y-axis represents the cumulated count of companies, while the x-axis represents the size.

5.2 Discretizing Features

The features extracted in the section on page 32 constitutes a compact representation of the time series, and the company. Through our feature generation we have abstracted away, tentatively, everything but the relevant information. Hence, the burden on the statistical abilities of a classifier is lowered to the minimum. In other words, through investing effort in abstracting the time series, we can invest less effort on the classifier. Keeping in mind that SpareBank 1 wants a technique that is easy to prove sound, and easy to implement, reinforces the argument for using a simple classifier.

We conducted a small empirical study testing several simple classifiers over the features. The Naive Bayes classifier performed best, which it surprisingly often does [Domingos and Pazzani, 1997]. See Section 6.5 and Chapter 7 for the details from this empirical study.

The Naive Bayes classifier is discussed in detail in Section 5.3. To anticipate the course of events, we will in this section focus on a minor design issue with the Naive Bayes.

Training a Naive Bayes classifier consists of counting the number of training instances for each class, and for each value of every feature. Recall from Section 5.1 that the features returns a real number. Real numbers measures continuous quantities, meaning it can have an infinite number of values. Since the task of counting the number of instances for every possible feature value is impossible, we need to discretize the features.[1]

Discretizing a numerical feature means splitting the continuous value space into a small finite set of distinct ranges. In the simplest form of discretizing, ranges are decided either by using equal-frequency binning or equal-width binning. Equal-width binning means splitting the range into a set of uniformly sized ranges. Notice, in Figure 5.6(a), how almost all the companies have a size within the first range. This is a typical result when discretizing a feature with unevenly distributed data points. Equal-frequency binning means deciding ranges by enforcing an equal amount of data points within each range, see Figure 5.6(b).

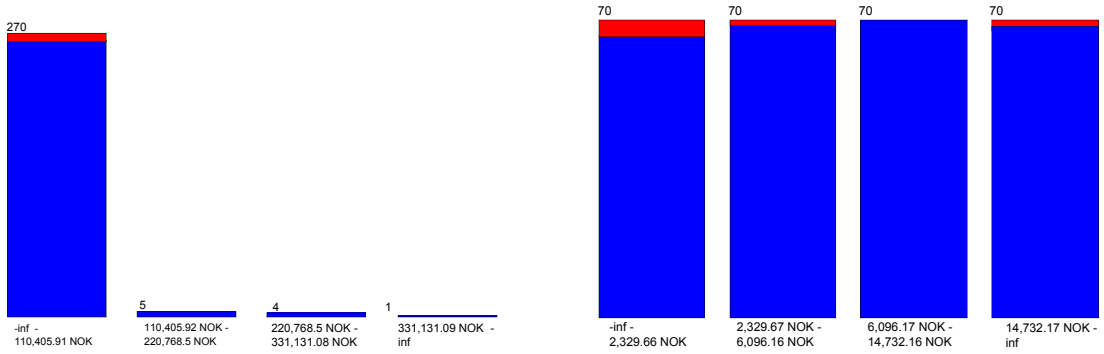


Figure 5.6: This figure presents the different types of binning used for simple discretizing. The colors indicate the number of instances from each class contained in each bin, where red constitutes the number of financially sick companies, and blue the healthy companies. In (a) the company size feature is discretized into 4 equal-width ranges. The alternative, equal-frequency, is presented in (b) over the same feature.

The two types of discretizing mentioned above are simple methods for discretizing. The optimal number of partitions need to be decided manually, for instance by visual inspection or by empirical testing. A potentially better procedure, proposed in [Fayyad and Irani, 1993], uses a entropy minimization heuristic for discretizing a range into multiple intervals. Let us first

[1] An extension of Naive Bayes enables it for numerical features by assuming they have a normal distribution. This assumption is often not very plausible in practice.

see how this procedure performs binary discretization, meaning discretizing a feature into two sub-ranges. Say, we are doing a binary discretization over the feature A, for instance the company size feature. T_{size} is then the *cut point* parting the set S of N training examples into two subset S_1 and S_2. The examples in S_1 have lower size values than T_{size} while the examples in S_2 have larger size values than T_{size}. If T_{size} is the perfect cut point, then all of the examples in S_1 are of class C_1, while the examples in S_2 are of class C_2. From being a feature ranging over a set of examples being of two classes (high class entropy), we have now partitioned the feature into two sub-features which each describes a class perfectly (low class entropy). The procedure presented underneath strives to find a discretizing which has the lowest possible amount of class entropy, given the training examples.

In this setting. the class entropy over S is translated to a measure of the degree of "randomness" that the classes, $C_1, \ldots, C_k$, of the training examples appears to exhibit. Formally, the class entropy of the set S is defined as:

$$\text{Ent}(S) = -\sum_{i=1}^{k} P(C_i, S)\ \log(P(C_i, S))$$

where $P(C_i, S)$ is the portion of the examples in S that are of class C_i. The logarithm may be to any convenient base. The best cut point is selected by sorting the examples in S by increasing values, evaluating each candidate cut point in the range of values. This totals to evaluating $N-1$ candidate cut points. To evaluate the resulting class entropy after a set S is partitioned into two subsets, S_1 and S_2, we calculate the weighted average of their resulting class entropies:

$$E(A, T; S) = \frac{|S_1|}{|S|}\text{Ent}(S_1) + \frac{|S_2|}{|S|}\text{Ent}(S_2)$$

The cut point, T_A, for which $E(A, T; S)$ is minimal amongst all the candidate cut points is selected.

So far we have only discussed how this procedure performs binary discretizing. In [Fayyad and Irani, 1993], the procedure is generalized for multi-interval discretizing by running the procedure recursively over the partitioned intervals. They also propose a criterion to be applied when deciding whether or not to refrain from applying further binary partitioning to a given interval. This criterion is based on the Minimum Description Length Principle, abbreviated MDLP, which were introduced by [Rissanen, 1978]. MDLP is defined to be the minimum number of bits required to uniquely specify an object out of the universe of all objects. In this procedure, MDLP is used as an estimate of the cost induced by partitioning an interval. The cost estimate is calculated as the number of bits needed to communicate a method that will enable the receiver to determine the class labels of the examples in the set. On a high level, the criterion estimates whether or not more information is gained through partition the set further. The MDLP criterion is defined as:

$$\text{Gain}(A, T; S) > \frac{\log_2(N-1)}{N} + \frac{\Delta(A, T; S)}{N}$$

where

$$\text{Gain}(A, T; S) = \text{Ent}(S) - E(A, T : S)$$

and

$$\Delta(A, T; S) = \log_2(3^k - 2) - [k\text{Ent}(S) - k_1 Ent(S_1) - k_2 Ent(S_2)].$$

Recall that k is the number of classes in the set. An interval is further partitioned if we gain more information, or in other words, lose more entropy, by partitioning the interval. The amount of information needed to be gained is given by the right side of the inequality sign. The details behind the derivation of the criterion are outside the scope of this thesis.

Feature	*# of partitions*	*Cut point, T*
Local Deposit Trend	1	-
Local Withdrawal Trend	2	$T = 0.0473$
Global Withdrawal Trend	2	$T = 0.0001$
Global Deposit Trend	2	$T = -0.0026$
Global Balance Trend	2	$T = -0.00465$
Consecutive Zero Deposits	2	$T = 1$
Consecutive Zero Withdrawals	2	$T = 1$
Consecutive Zero Balance	1	-
Zero Deposits	2	$T = 3.5$
Zero Withdrawals	2	$T = 1.5$
Age	1	-
Company Size	2	$T = 6107000$

Table 5.1: Table enumerating the discretized features.

When applying the procedure explained above on our training examples we get the following partitions:

In Table 5.1 we see that the majority of our features were discretized into two subintervals. Three of the features, the Company Age feature, the Consecutive Zero Balance feature, and the Local Deposit Trend feature, are not partitioned. The discretizing procedure decided that there were not enough to gain from discretizing the features any further, indicating that the features contains too much randomness to be useful in distinguishing the classes. Such distracting features often confuses the classifier. Additionally, since the binary value of the three features, with only one partition, will be equal to one for every possible example, the features may as well be excluded from the model.

The procedure of removing low performing features is called feature selection. Feature selection is beneficial due to the following three reasons [Guyon and Elisseeff, 2003]: it improves the prediction performance of the classier, it provides faster and more cost-effective classifiers, and it provides a better understanding of the underlying process that generated the data. In our case, the latter point translates to learning what kind of incidents and patterns that usually precedes a period of financially struggle.

After discretizing the numerical features into q ranges, the features will individually consist of q binary features, indicating whether or not a data point is located within the respective ranges. Training a Naive Bayes classifier is now possible, since we have decreased the sample space of each feature from being of infinite size to only q distinct values. We will exploit this fact in the next section, where we will derive the Naive Bayes classifier.

5.3 Supervised Learning with a Naive Bayes Classifier

Bayesian Learning is an approach to learning a classifier that uses Bayes rule. For the following description of the Naive Bayes classifier we have used [Russell and Norvig, 1995] as a reference.

We view each company as a vector of random variables. Let C be the random variable, also called the class variable, indicating whether or not the company is financially troubled. Let $X_1, X_2, \ldots, X_p$ be the random feature variables which we derived in Section 5.1. For a training example, values for all of these are known, including $C = c$. For a test example, only the values $X_1 = x_1$ to $X_p = x_p$ are known. Given a test example, we want the classifier to estimate the probability, $P(C = c|X = x)$ for each value that C may take. In our case we only have two classes, namely $C = 1$ if the company is financially sick, and $C = 0$ otherwise. More precisely, we want to estimate the conditional probability $P(C|x_1, \ldots, x_p)$, which is different for each example

since each example has different values for the feature random variables $X_1 = x_1$ to $X_p = x_p$ (we will in the following text for convenience denote $P(X_j = x_j)$ with $P(x_j)$).

Applying Bayes rule to the test example gives

$$P(C|x_1, \ldots, x_p) = \frac{P(x_1, \ldots, x_p|C)\, P(C)}{P(x_1, \ldots, x_p)}. \tag{5.1}$$

There are three factors in Equation 5.1, two in the numerator and one in the denominator, each of which we need to learn from training data. The easiest factor to learn is the unconditional probability $P(C = c)$. This is the overall probability of the class variable having the value $c \in \{0, 1\}$ for the whole population of companies. We estimate this as the fraction $\frac{count(C=c)}{N}$, where $count(C = c)$ is the number of instances of each class in the training set and N is the total number of training examples.

The denominator can by the law of total probability be written as

$$P(x_1, \ldots, x_p) = \sum_c P(x_1, \ldots, x_p|C)P(C)$$

Due to the fact that we only have two classes, financially sick ($C = 1$) and financially healthy companies ($C = 0$), we have that $P(x_1, \ldots, x_p)$ equals

$$P(x_1, \ldots, x_p|C = 1)P(C = 1) + P(x_1, \ldots, x_p|C = 0)[1 - P(C = 1)]$$

This leaves us with only one factor left to estimate, which also is needed to calculate the denominator, the conditional probability: $P(x_1, \ldots, x_p|C)$. So far we have not made any assumptions; the equations above are mathematically sound. The Naive Bayes classifier has earned the word "naive" in its name due to the following assumption. First, use the product rule to rewrite $P(x_1, \ldots, x_p|C)$ as

$$P(x_1|C)P(x_2|x_1, C) \ldots P(x_p|x_1, \ldots x_{p-1}C).$$

Now we make assumptions that are *not* always true. We assume that inside each class, each feature is independent of all the other features, or mathematically,

$$\begin{aligned} P(x_2|x_1, C) &= P(x_2|C) \\ &\vdots \\ P(x_p|x_1, \ldots, x_{p-1}, C) &= P(x_p|C). \end{aligned}$$

This assumption makes estimating $P(x_j|C)$ easy:

$$P(x_j|C) = \frac{count(x_j \wedge C)}{count(C)}. \tag{5.2}$$

Now we have all the factors we need, and can express Equation 5.1 as

$$P(C = c|x_1, \ldots, x_p) = \frac{P(C = c) \prod_{j=1}^{p} P(x_j|C = c))}{\sum_{v \in \{0,1\}} [P(C = v) \prod_{j=1}^{p} P(x_j|C = v)]} \tag{5.3}$$

The denominator in Equation 5.3 is usually called the partition function, or normalization function, and scales the output to a real number in the range from 0 to 1, and hence a valid probability. In principle, if we do not need probabilities, the denominator can be left out, since it is constant given the class. Equation 5.3 will then return a score, and the class with the highest score is the result of the classification.

A common pitfall with the Naive bayes classifier is revealed if a value of a feature, for example value y of feature j, is never observed in the training data for some class c. Then we will obtain

the estimate $P(X_j = y|C = c) = 0$ exactly. For every test example with value y for feature j we will have $P(C = c|\vec{X}) = 0$ by multiplication regardless of the values of all other features. This is undesirable because it gives a veto to a single feature. Our imbalanced dataset is especially vulnerable to this pitfall since the chances for a feature j having an value y that is never observer is very high for the minority class.

The solution is to make all probabilities non-zero (and also not exactly 1.0) by adding "pseudo-counts" to the real counts. Define

$$P(C = c) = \frac{count(C = c) + 1}{N + r}$$

where r is the number of different class values, and define

$$P(X_j = x_j|C = c) = \frac{count(X_j = x \wedge C = c) + 1}{count(C = c) + q_j}$$

where q_j is the number of different values of X_j. Smoothing with pseudo-counts of 1, as above, is called Laplace smoothing. In practice, this amount of smoothing is too much. We can define more general smoothing as follows:

$$P(X_j = x_j|C = c) = \frac{count(X_j = x \wedge C = c) + \lambda}{count(C = c) + q_j\lambda}.$$

A good common value for λ is 0.1 instead of $\lambda = 1$ which corresponds to Laplace smoothing.

The actual algorithm to train a Naive Bayes classifier is the following. Given the training data:

1. Calculate a vector $P(C = c)$ for each class value c
2. For each c and each feature x_j, calculate a vector $P(X_j = y|C = c)$ (Equation 5.2) where y ranges over the different possible values of X_j, the jth feature.

Since we have two classes, step 1 gives us a vector of two numbers that adds up to 1.0. If feature X_j has q alternative values (for instance $q = 2$, as we ended up with when discretizing our real valued data) then step 2 gives a vector of q numbers. Again, this vector adds up to 1.0.

After training, the algorithm for applying the classifier on a test example is the following. Note, for a test example we know the values for each feature, X_j, but not the value of C. The probability of an example company having the class value c, given the values of the features, is calculated in Equation 5.3. Through training we have estimated all the factors needed for this calculation. The test instance is classified as of the class with the highest probability.

As mentioned in Section 5.2, the Naive Bayes classifier often performs surprisingly well doing binary classification. Although the classifier separates two classes well, Naive Bayes is not known for producing good estimates for $P(C|X = x)$. This is due to the naive assumption, which limits how the distribution over the data points may look like. Recall, the naive assumption assumes that given a test example's class all features are independent of each other.

Say, we have two distributions over two types of objects measured over two features, and that there is a high dependency between the two features. Hence, the distributions may look like the two distributions in Figure 5.7(a). Due to the naive assumption, through the eyes of a Naive Bayes classifier those two distributions may look like the ones in Figure 5.7(b). Notice, the dependency, or correlation, between the features is neglected. Thus, the Naive Bayes estimate of $P(C|X = x)$ for the single cross in Figure 5.7(b) will be smaller than its actual value, depicted in Figure 5.7(a). This effect is one of the drawbacks by using the Naive Bayes classifier. On the other hand, notice that the red line separating the two classes is still appropriate in Figure 5.7(b), even though the correlation is neglected.

The Naive Bayes classifier, discussed above, is not cost sensitive. In Section 5.4 on the next page we discuss how we can add cost-sensitivity to the classifier without changing anything of what we have discussed in this section.

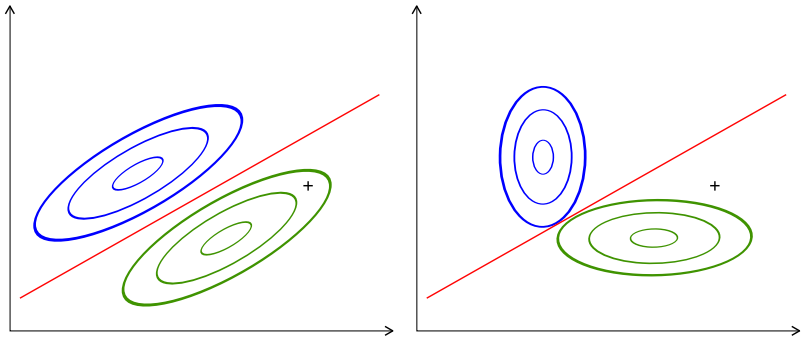


Figure 5.7: Both figures display two Gaussian distributions and one data point. In (a) the distributions incorporate dependency between the two features, indicated by the skewness in the distribution. In (b) this dependency is removed.

5.4 Adding Cost-Sensitivity to the Classifier Using MetaCost

As stated in Chapter 3 our dataset is imbalanced. Examples of unhealthy companies makes up only about 3.5% of the dataset. Classifiers usually tend to favor the major class when classifying imbalanced dataset, classifying all the instances to the major class. This is due to the way a classifier is designed. Most classifier have some means of preventing overfitting, to remove sub-concepts that are not believed to be meaningful [Kotsiantis et al., 2006]. Say, for instance, that our minority class, instead of representing a class of rare and exceptional examples, represents noisy data. The minority class would be apprehended as not meaningful, thus ignored. In our case this effect translates to classifying all companies as healthy.

The machine learning community has mostly addressed the issue of class imbalance in two ways [Karagiannopoulos et al., 2007]. One is to give distinct cost to miss-classifications [Domingos, 1999] while the other is to re-sample the original dataset, either by oversampling the minority class and/or under-sampling the majority class [Japkowicz, 2002; Kubat et al., 1998]. We have chosen to focus on the former way of handling imbalanced dataset, namely by adding cost to training instances. By adding cost sensitivity to the classifier, we not only force it to appreciate the minority class, but also to acknowledge that in real world applications the cost for making classification errors are not always equal, as most of the current-available algorithms for classification would assume. For instance, in our case, if a company goes into bankruptcy without the bank getting an early warning, called a false negative, the bank will not be able to execute any pre-emptive measures, such as making sure the bank has enough risk capital left. On the other hand, if the bank receives a warning regarding a company that is not economically unhealthy, called a false positive, the only cost is the extra labor that is needed in order to manually double-check the company.

We use a procedure called MetaCost, proposed in [Domingos, 1999], for adding cost sensitivity to our classifier. MetaCost can be applied to any error-based classifier by treating it as a black box, requiring no knowledge of its functioning and change to it. Before we dive into the details of MetaCost, let us start with some basic concepts. If, for a given example x, we know the probability of each class j, $P(j|x)$, the optimal prediction for x is the class i that minimizes the conditional risk:

$$R(i|x) = \sum_{j} P(j|x)C(i,j) \tag{5.4}$$

where the conditional risk, $R(i|x)$, is the expected cost of predicting that x belongs to class i, and

C is the cost matrix, with $C(i, j)$ being the cost of predicting that an example belongs to class i when in fact it belongs to class j. See Table 5.2 for an example of a cost matrix. Note that the diagonal of a cost matrix is always zeros since classifying correctly never should induce a cost.

	Predicted, i	
Actual, j	0	1
0	0	1
1	10	0

Table 5.2: Cost Matrix Example

The idea behind MetaCost is to relabel the training instances according to $\arg\min_i R(i|x)$, where $R(i|x)$ is given in Equation 5.4. The relabeled training instances are then used to train the error-based classifier. The difference between training the classifier on the original training set and the relabeled training set are few. Now the classifier finds the cost-optimal separation of the classes, while it before only found the error-optimal separation.

In order to relabel the training examples with their "optimal" classes, using Equation 5.4, we need to estimate $P(j|x)$. Some classifiers output estimated class probabilities, which is then used for $P(j|x)$. If the classifier only outputs the predictions, then $P(j|x) = 1$ for the class that were predicted, and $P(j|x) = 0$ for the other classes. To ensure that the estimated class probabilities are stable, MetaCost estimates $P(j|x)$ over m different re-samples of the training set and averages over them. The Naive Bayes classifier outputs class probabilities. That is, if implemented as described in Section 5.3 on page 37 where we estimate the denominator to normalize the outputs to be a real number in the range $[0, 1]$, hence a valid probability.

After training the relabeled training set on the original classifier we have concluded our model. We are now ready to test it on unseen data.

6 Results

Binary Classification can be briefly explained in two steps (1) For every instance, assign a *score value* (2) Determine a threshold such that every instance with score larger than the threshold is classified as *positive* and the others classified as *negative*. Such a threshold is called a *discrimination threshold*. This chapter presents the results after classification with the Naive Bayes classifier explained in Chapter 5 and contrasts it with the results from the PD-rating and other classifiers. The discrimination threshold is central when calculating confusion matrices and is discussed in detail for the PD-rating.

6.1 Terminology

In the classification reports in this chapter you will find terms such as "Confusion Matrix", "ROC Area" and "False Positive Rate". Before presenting the results we briefly recap the meaning of these terms.

6.1.1 Confusion Matrix

Our classification problem is binary, i.e. there are two classes and four distinct outcomes in the general case. Assume that each instance is either mapped to class p or n; if the classifier assigns the class p to an instance which is in fact of class p we call the mapping a *true positive*. In contrast, if the classifier assigns the instance to class n when it is in fact class p we call it a *false negative*. The same logic applies the other way around: If the classifier assigns an instance to class n when it is in fact class n we have a *true negative*. If the classifier assigns the class p to a company which is in fact class n it is a *false positive*.

True/false positives and negatives form the foundation for most of the following error measures. A *confusion matrix* presents true- and false positives/negatives in a matrix. Figure 6.1 is an example of a confusion matrix.

		Predicted value n'	Predicted value p'
Actual value	*n*	True Negatives	False Positives
	p	False Negatives	True Positives

Figure 6.1: A confusion matrix illustrating true/false positives and negatives.

The True Positives Rate, or *TP-Rate*, is defined as the number of true positives over the total number of predicted positives: $\frac{TP}{TP+FN}$. Likewise, False Positives Rate, or *FP-Rate*, is the proportion of negative instances that were reported being positive: $\frac{FP}{FP+TN}$. The last rate we will consider is the False Negatives Rate, *FN-Rate*, it is the proportion of positive instances that were reported as being negative: $\frac{FN}{FP+TP}$. Together, these three rates—along with precision and recall explained shortly—form the basis for various performance measures.

In statistics, two sources of error have become universally accepted, they are called *Type I* and *Type II* error. These two terms are interchangeable with "false positives" and "false negatives", but we mention them here as they are commonly found in the literature. By statistical convention,

when proposing a hypothesis (e.g. the company is not healthy), we assume by default that the opposite is true. This is called the *null hypothesis*. A common example used is the court room: the null hypothesis is that the accused is not guilty and the *speculative hypothesis* (the one to refute or not) is that he is guilty.

Type I and Type II errors (or false positives and false negatives) are defined as [Neyman and Pearson, 1928]:

- Type I errors (the "false positive"): the error of rejecting the null hypothesis given that it is actually true; e.g., a court finding a person guilty of a crime that they did not actually commit.
- Type II errors (the "false negative"): the error of failing to reject the null hypothesis given that the alternative hypothesis is actually true; e.g., A court finding a person not guilty of a crime that they did actually commit.

The speculative hypothesis in this thesis is that the company is going to have a default or incur loss for the bank in some other way. The null hypothesis (assumed by default to be true) is that the company is healthy. A Type I error occurs when a company is classified as unhealthy when it is in fact healthy. A Type II error occurs when a company is classified as healthy when it is in fact unhealthy.

6.1.2 Precision and Recall

Precision and Recall are two measures used to integrate the confusion matrix into a single number. *Recall*—also known as sensitivity—is defined as $\frac{TP}{TP+FN}$, it tells us the proportion of the actual positives that where in fact recognized by the classifier. *Precision* is defined as $\frac{TP}{TP+FP}$ and tells us how many of the positives recognized by the classifier where in fact positives.

Makhoul et al. [1999] describe precision and recall with these words:

> *"A perfect Precision score of 1.0 for a class C means that every item labeled as belonging to class C does indeed belong to class C (but says nothing about the number of items from class C that were not labeled correctly) whereas a Recall of 1.0 means that every item from class C was labeled as belonging to class C (but says nothing about how many other items were incorrectly also labeled as belonging to class C)."*

The goal of Part I is to achieve high recall: of the companies that is in fact unhealthy we should recognize as many as possible. It is better to mark some healthy companies as unhealthy, than the other way around, put another way: It is worse to commit a Type II error than a Type I error.

6.1.3 Accuracy, ROC Area and Expected Cost

The predictive performance of a classifier has traditionally been measured in "Accuracy". *Accuracy* is defined as the proportion of the dataset that where correctly classified: $\frac{TP+TN}{TP+TN+FP+FN}$. Some classifiers (like Naive Bayes) output a *confidence* in its predictions. This information is lost in the Accuracy measure as a true positive is regarded as a true positive as long as the confidence is above the discrimination threshold;[1] information about how *far* from the threshold is discarded. Ling et al. [2003] presents a proof in their paper that the area under the "Receiver Operator Characteristics"-curve is a statistically more consistent and more discriminating performance measure than Accuracy. We will therefore base our evaluation of the classifiers on the ROC curve and not accuracy; we have included accuracy in the subsequent result tables because of its prior popularity.

The ROC curve, or "Receiver Operator Characteristics" curve, shows how the number of correctly classified positive examples (TPR) varies with the number of incorrectly classified

[1] See the introduction to this chapter for a brief explanation of the discrimination threshold.

examples (FPR) when shifting the discrimination threshold. Figure 6.2 shows an example of a ROC curve. Each point in the TP-FP-plane is calculated from different discrimination thresholds.

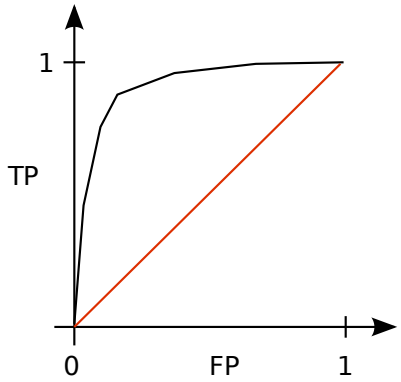


Figure 6.2: Example ROC curve. Each point on the curve is calculated for different discrimination thresholds. The diagonal line illustrates the lower bound for an acceptable classifier; i.e. $TP = FP$

Technically, the ROC curve contains all the information we need to reason about the performance of classifier. It is, however, convenient to reduce the curve to a single number: The area under the ROC curve (ROC AUC) is a common representation which we will use here. Statistically, the ROC AUC specifies the probability that, when we draw one positive and one negative example at random, the decision function assigns a higher value to the positive than to the negative example. A high value for the area under the curve indicates good performance.

Hand and Till [2001] demonstrates a way to calculate the area under the ROC curve without calculating the actual curve, but from the predictions $P(C = c|X)$. This enables us to calculate the ROC AUC for the PD rating as well as for the Naive Bayes classifier. For more information about ROC curves we refer the interested reader to [Ling et al., 2003, Fawcett, 2006 and Hand and Till, 2001].

At last we introduce the *Expected cost* or $E(cost)$ as a measure for how well the classifier performs. The ROC measure assumes equal cost for Type I and II errors. This is not the case in our situation (*cf.* Section 6.1.2), we therefore define the "Expected Cost" measure as

$$E(cost) = \frac{FP}{N} \cdot C(FP) + \frac{FN}{N} \cdot C(FN) \tag{6.1}$$

where C is the cost function defined by the cost matrix in Section 5.4, and N is the total number of examples in the dataset. The area under the ROC curve combined with $E(cost)$ provides an adequate measure of how well the classifier performs, considering both confidence and cost function. An excellent classifier should have high ROC AUC and low $E(cost)$.

6.2 Naive Bayes Results

Table 6.1 on the following page presents the performance of our classification model derived in Chapter 5. A cost-sensitive Naive Bayes classifier is trained over the abstracted training set, and tested on the validation set. We use the cost-matrix presented in Table 5.2 on page 41, which has a cost of 10 for *false negatives*, and 1 for *false positives*. Our performance is contrasted with the performance of the PD-rating. The discrimination threshold used for testing the PD-rating has been optimized on the training set, and tested on the validation set.

It is worth explaining the mapping from the real number PD-rating to the confusion matrix on the following page. SpareBank 1 uses the PD-rating implicitly through a set of "risk classes" from A to I, where A indicates excellent health and I indicates bad health. The risk class that performs

		Predicted Value	
		N	P
Actual Value	N	247	26
	P	2	5

(a) Naive Bayes

		Predicted Value	
		N	P
Actual Value	N	242	31
	P	2	5

(b) PD-Rating

Table 6.1: (a) Confusion Matrix for the Naive Bayes Classifier. (b) Confusion Matrix calculated from the PD-rating with discrimination threshold = 0.025.

best on the training set—Risk Class G—is assigned to those companies with a PD-rating between 0.025 and 0.05.[1] When calculating Table 6.1 (b) we use 0.025 as a discrimination threshold. See Table 6.2, where we have tested the different risk classes as discrimination threshold values over the training set. We measure the PD-rating calculated at the last sample point for each company. For a healthy company this will be at the start of the safety buffer; for an unhealthy company this will be before the first "usable" evidence of a default or loss reckoning.[2] See Section 4.2 on page 20 for more details about the last sample point.

Table 6.2 illustrates confusion matrices for different discrimination thresholds. Each threshold correspond to a risk class for the PD-rating.

		Predicted Value	
		N	P
Actual Value	N	854	19
	P	17	14

(a) Threshold = 0.1

		Predicted Value	
		N	P
Actual Value	N	824	49
	P	13	18

(b) Threshold = 0.05

		Predicted Value	
		N	P
Actual Value	N	752	121
	P	9	22

(c) Threshold = 0.025

		Predicted Value	
		N	P
Actual Value	N	664	209
	P	6	25

(d) Threshold = 0.0125

Table 6.2: Confusion matrices for the PD-rating with various discrimination thresholds over the training data. (a) Risk class H, (b) Risk Class G, (c) Risk Class F, (d) Risk Class E

6.2.1 Performance

Table 6.3 (a) on the next page illustrates the performance of the Naive Bayes classifier. Table 6.3 (b) illustrates how well the classifier performs on each class. The null hypothesis for the first line is $H_0 = 1$, for the second line $H_0 = 0$. I.e. the first line can be interpreted as how well the classifier recognizes healthy companies and the second line as how well the classifier recognized unhealthy companies. The first line is only included for the sake of completeness, the second line is the interesting part.

Table 6.4 illustrates the performance for the PD-rating when thresholded on Risk Class G, $PD = 0.025$.

[1] See Appendix A.1 for the complete mapping

[2] An "unsuable" default or loss reckoning occurs when there are too few data points prior to the default to make a sound analysis

Results	
E(cost)	0.164
ROC	0.877
Accuracy	0.900

(a)

Detailed Performance By Class

Class	*TP Rate*	*FP Rate*	*FN Rate*	*Precision*	*Recall*
0	0.90476	0.28571	0.095238	0.99197	0.90476
1	0.71429	0.095238	0.28571	0.16129	0.71429

(b)

Table 6.3: Statistics for the Naive Bayes Classifier.

Results	
E(cost)	0.182
ROC	0.921
Accuracy	0.882

(a)

Detailed Performance By Class

Class	*TP Rate*	*FP Rate*	*FN Rate*	*Precision*	*Recall*
0	0.88645	0.28571	0.11355	0.9918	0.88645
1	0.71429	0.11355	0.28571	0.13889	0.71429

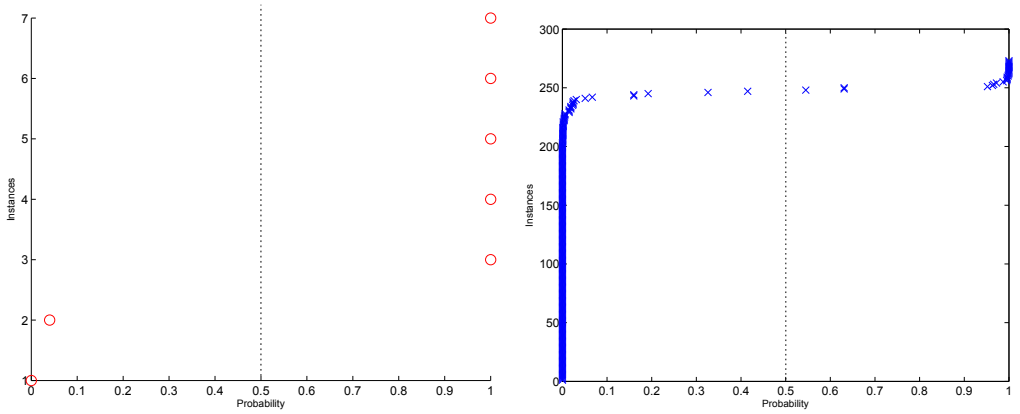
(b)

Table 6.4: Statistics for the PD-rating with a discrimination threshold equal to 0.025 (Risk Class G).

6.2.2 Confidence Plots

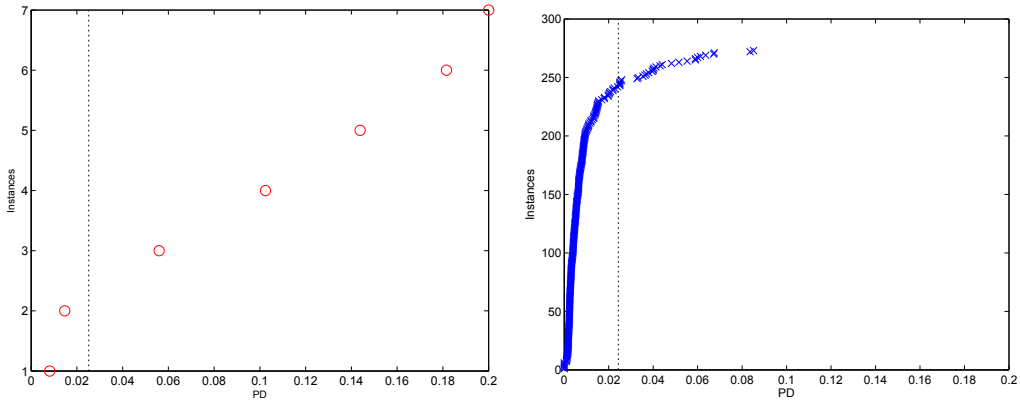
Figure 6.3 and Figure 6.4 on the following page illustrates the *confidence* of the Naive Bayes classifier and PD rating respectively for the validation set. Discrimination thresholds are marked with a dotted line. Remember that the discrimination thresholds were decided using the training set; from the particular case of the validation set, a threshold of 0.1 in Figure 6.4 on the next page would probably be better.

Ideally all the unhealthy companies in Figure 6.3 (a) would receive a score close to 1.0 and all the healthy companies is Figure 6.3 (b) would receive a score close to 0. The same is true for Figure 6.4 (a) and (b).



(a) (b)

Figure 6.3: Confidence plots for the Naive Bayes classifier. (a) Confidence assigned to each of the unhealthy companies (red circles) in the validation set. (b) Confidence assigned to each of the healthy companies (blue crosses) in the validation set. The discrimination threshold is 0.5



(a) (b)

Figure 6.4: Confidence plots for the PD rating. (a) Confidence assigned to each of the unhealthy companies (red circles) in the validation set. (b) Confidence assigned to each of the healthy companies (blue crosses) in the validation set. The discrimination threshold is 0.025

6.3 Feature Evaluation

In Section 5.1 we derived 12 features abstracting the data regarding a company. Three of these were excluded from the model in Section 5.2, due to high level of randomness. Most of the features left processes the deposit and withdrawal time series and quantifies events and properties within them. The objective by deriving these features is to remove information not relevant for classifying whether or not a company is financially sick or healthy. Although we have strived to derive features that are relevant, and provide information which seems to separate the two classes, the features still vary in relevance. Due to the implementation cost associated with implementing the features, the bank is interested in the performance impact imposed by neglecting a given feature. We have performed a feature evaluation where we identify how important each feature is.

Feature evaluation was performed by using an feature subset evaluator. We perform a 10-fold cross-validation, were we for each fold evaluate all possible subsets of features on $\frac{9}{10}$ of the training set. For every fold the best performing subset is reported back. With nine features left, we have to evaluate a total of $2^9 - 1 = 511$ possible subsets each fold. The -1 indicates that we do not need to evaluate the empty feature subset. Each subset of features is evaluated by, again, performing a 10-fold cross-validation with MetaCost (see Section 5.4) wrapping around a Naive Bayes classifier (see Section 5.3) over the feature subset.

The result from the feature evaluation is the best performing feature subset from all ten folds. We create Table 6.5, by counting the number of times a feature is a member of the best subset:

Feature	*# of Folds Member of Best Subset*
Local Withdrawal Trend	3
Global Withdrawal Trend	4
Global Deposit Trend	2
Global Balance Trend	10
Consecutive Zero Deposits	2
Consecutive Zero Withdrawals	2
Zero Deposits	6
Zero Withdrawals	10
Company Size	9

Table 6.5: Table showing the counts of how many folds each feature was a member of the best performing feature subset.

Notice in Table 6.5, that two features were members of the best performing subset ten out of ten times, i.e. the Zero Withdrawals feature and the Global Balance Trend feature. Every feature was member of the set of best performing features *at least* twice. In Table 6.6 we present the results from running our model over the validation set without the lowest performing features: the features that were member of best performing subset only twice.

		Predicted Value	
		N	P
Actual Value	N	243	30
	P	2	5

Table 6.6: Confusion matrix derived from running a stripped down model over the validation set, excluding the lowest performing features (Global Deposit Trend, Consecutive Zero Deposits and Consecutive Zero Withdrawals). Compare with Table 6.1 (a) on page 46.

6.4 Cross Validation Results

The results presented in Section 6.2 was calculated by presenting a trained classifier with new data. These results should be considered the final results from Part I, however, since the validation set ended up having only 7 unhealthy companies, the difference in performance between the two classifiers is somewhat unclear. We present in this section the results from the training phase where the sets of healthy and unhealthy companies are larger.

When training and validating a classifier using the same data source one is subject to overfitting, that is, the classifier is only able to recognize companies in the training set and is in practice worthless. In order to counter overfitting, a procedure called *Cross Validation* is used. The results presented in this section have been calculated through the use of cross validation with 10 folds.[1]

		Predicted Value	
		N	P
Actual Value	N	769	104
	P	5	26

(a) Naive Bayes

		Predicted Value	
		N	P
Actual Value	N	752	121
	P	9	22

(b) PD-Rating

Table 6.7: Confusion matrices from cross validation over the training set. (a) Naive Bayes Classifier. (b) PD-Rating with discrimination threshold equal to 0.025

6.4.1 Performance

Results	
E(cost)	0.170
ROC	0.909
Accuracy	0.879

(a) Naive Bayes

Results	
E(cost)	0.233
ROC	0.856
Accuracy	0.856

(b) PD rating

Table 6.8: Performance metrics for Naive Bayes and the PD-rating on the training set with cross validation

Detailed Performance By Class					
Class	*TP Rate*	*FP Rate*	*FN Rate*	*Precision*	*Recall*
0	0.881	0.161	0.119	0.993	0.881
1	0.839	0.119	0.161	0.200	0.839

(a) Naive Bayes

Detailed Performance By Class					
Class	*TP Rate*	*FP Rate*	*FN Rate*	*Precision*	*Recall*
0	0.861	0.290	0.139	0.988	0.861
1	0.710	0.139	0.290	0.154	0.710

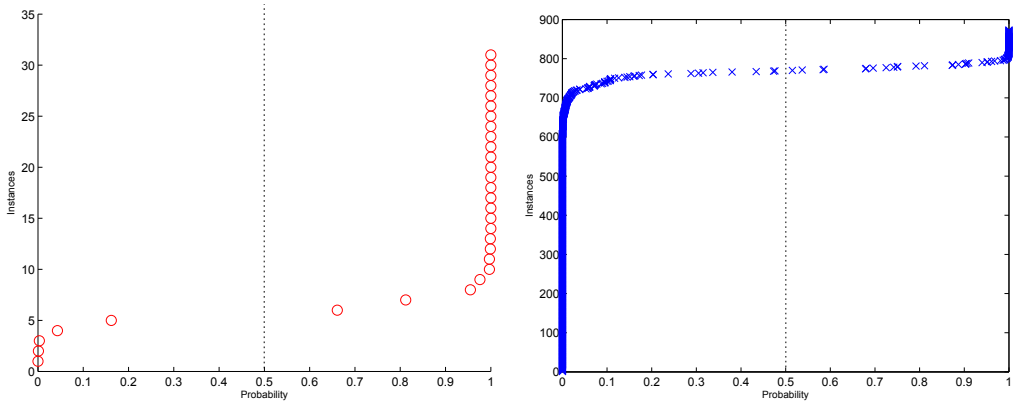
(b) PD Rating

Table 6.9: Detailed statistics from the training set (a) The Naive Bayes classifier. (b) The PD Rating.

[1] In short, the cross validation method segments the dataset in n folds where $n-1$ folds are used for training and the last fold for testing. The average of all permutations of folds is then used for calculating results.

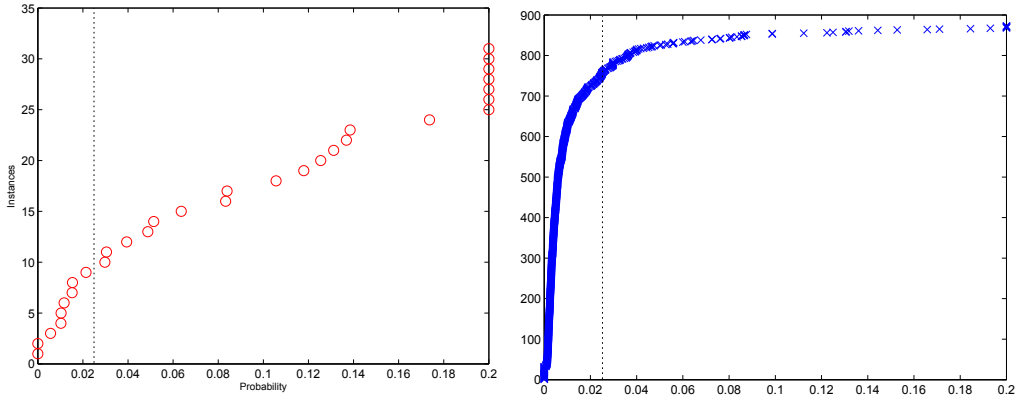
6.4.2 Confidence Plots

Figure 6.5 and Figure 6.6 illustrate the confidence plots for the cross validation results.



(a) (b)

Figure 6.5: Confidence plots for the Naive Bayes classifier. (a) Confidence assigned to each of the unhealthy companies (red circles) in the validation set. (b) Confidence assigned to each of the healthy companies (blue crosses) in the validation set. The discrimination threshold is 0.5



(a) (b)

Figure 6.6: Confidence plots for the PD rating. (a) Confidence assigned to each of the unhealthy companies (red circles) in the validation set. (b) Confidence assigned to each of the healthy companies (blue crosses) in the validation set. The discrimination threshold is 0.025

6.5 Performance of Other Classifiers

The decision to use the Naive Bayes classifier in preference to any other were not coincidental. Several other classifiers where evaluated prior to deciding for Naive Bayes. This section briefly summarizes the performance displayed by various machine learning algorithms. The algorithms were (naturally) evaluated using the training set only, and not the validation set. The results in this section is therefore calculated from the training set with cross validation, 10 folds. Page 52 illustrates the performance of K-Nearest Neighbor and Logistic Regression; page 53 illustrates the performance for a RBF-Network and Feed Forward Neural Network.

6.5.1 K-Nearest Neighbor

		Predicted Value	
		N	P
Actual Value	N	818	55
	P	14	17

(a)

Results	
E(cost)	0.2157
ROC	0.7427
Accuracy	0.9237

(b)

Detailed Performance By Class

Class	*TP Rate*	*FP Rate*	*FN Rate*	*Precision*	*Recall*
0	0.9370	0.4516	0.0630	0.9832	0.9370
1	0.5484	0.0630	0.4516	0.2361	0.5484

(c)

Table 6.10: Performance measure of K-Nearest Neighbor. (a) Confusion Matrix, (b) Results, (c) Performance table.

6.5.2 Logistic Regression

		Predicted Value	
		N	P
Actual Value	N	809	64
	P	15	16

(a)

Results	
E(cost)	0.2367
ROC	0.7214
Accuracy	0.9126

(b)

Detailed Performance By Class

Class	*TP Rate*	*FP Rate*	*FN Rate*	*Precision*	*Recall*
0	0.9267	0.4839	0.0733	0.9818	0.9267
1	0.5161	0.0733	0.4839	0.2000	0.5161

(c)

Table 6.11: Performance measure of Logistic Regression. (a) Confusion Matrix, (b) Results, (c) Performance table.

6.5.3 RBF Network

	Predicted Value		
		N	P
Actual Value	N	794	79
	P	10	21

(a)

Results	
E(cost)	0.1980
ROC	0.7935
Accuracy	0.9015

(b)

Detailed Performance By Class

Class	*TP Rate*	*FP Rate*	*FN Rate*	*Precision*	*Recall*
0	0.9095	0.3226	0.0905	0.9876	0.9095
1	0.6774	0.0905	0.3226	0.2100	0.6774

(c)

Table 6.12: Performance measure of a Radial Basis Function Network (a) Confusion Matrix, (b) Results, (c) Performance table.

6.5.4 Feed Forward Neural Network

	Predicted Value		
		N	P
Actual Value	N	795	78
	P	12	19

(a)

Results	
E(cost)	0.2190
ROC	0.7618
Accuracy	0.9004

(b)

Detailed Performance By Class

Class	*TP Rate*	*FP Rate*	*FN Rate*	*Precision*	*Recall*
0	0.9107	0.3871	0.0893	0.9851	0.9107
1	0.6129	0.0893	0.3871	0.1959	0.6129

(c)

Table 6.13: Performance measure of a Feed Forward Neural Network trained with backpropagation. (a) Confusion Matrix, (b) Results, (c) Performance table.

7 Discussion

7.1 Robustness and Adequacy of Evaluation Measures

The Area Under the ROC Curve, abbreviated ROC AUC, provides a measure of the "goodness" in the classification; erroneous classifications affects the ROC AUC as a function of how far from the discrimination threshold the classifications are. Assuming that instances that were only *just* misclassified are better than those that were misclassified by a large margin. In other words, the area under the ROC curve provides a good measure of how *robust* the classifier is. If we weight Type I and Type II errors equally, then the ROC AUC contains all the information needed to validate or invalidate a classifier.

In our case, we are willing to accept less Type II errors (false negatives) at the cost of Type I errors (false positives), we therefore need an additional measure to the area under the ROC curve. We have introduced the *expected cost*, or $E(cost)$, measure to capture the effect of the cost matrix, consult Section 6.1.3 on page 44 for details. In addition to these two measure, *Recall* is of particular interest for unhealthy companies. This measure tells us the proportion of unhealthy companies that were recognized as unhealthy companies by the classifier. The classifier should be validated or discarded with regard to these three measures.

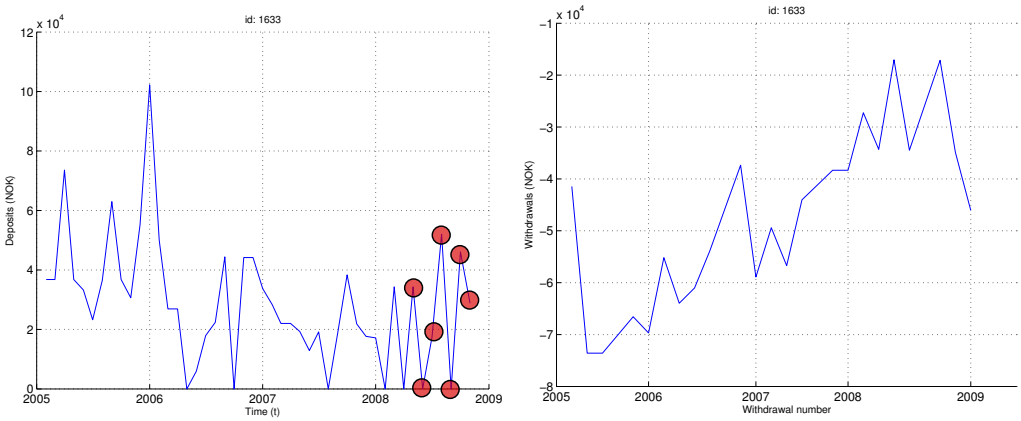
It is important to distinguish between the validation set results and the training set results. The validation set results should be considered more important than the training set results with cross validation. Cross validation is solely used for choosing which algorithm to use, and to tweak the configuration of this algorithm. Thus, the result from cross validation is gained from test examples used to choose the configuration in the first place. We are more interested in seeing how good our model generalizes, i.e. how good our model performs on unseen data. This is gauged by testing our model over the validation set.

The size of the validation set ought to be large enough to distinguish between results from different algorithms. In our case the validation set ended up being fairly small with few unhealthy companies. The cross validation set provides an *illustration* of the classifier's performance in the face of more data and is therefore mentioned briefly.

7.2 Evaluation of Final Results

Table 6.1 on page 46 illustrates the confusion matrices for the Naive Bayes classifier and the PD rating. From these matrices it might seem like the Naive Bayes classifier outperforms the PD rating because of a small edge on the amount of false positives. However, if we consider the ROC value we see that the PD rating is rated higher than the Naive Bayes. The explanation for this is evident in the confidence plots, Figure 6.3 and 6.4 on page 48. In the case of unhealthy companies, Naive Bayes assigns more erroneous confidence to the misclassified companies than the PD-rating. The same is true for the healthy companies. By "erroneous", we mean erroneous relative to the total distribution of companies, i.e. a healthy company rated as 0.2 by PD-rating is just as bad as a healthy company rated as 1.0 by the Naive Bayes since in both cases 100 % of the dataset is rated lower. We touched into Naive Bayes inability to return accurate confidences in Section 5.3. Thus, by only considering the ROC it is not a big surprise that Naive Bayes does not come out on top. If we, on the other hand, consider the $E(cost)$, we realize that Naive Bayes have less expected cost than the PD-rating (0.164 vs. 0.182).

Interestingly, the Naive Bayes classifier and the PD rating does not agree on all the misclassified false negatives. Starting with the false negative they have in common; both classifiers erroneously



(a) (b)

Figure 7.1: This company is misclassified as a false negative by both the Naive Bayes classifier and the PD rating. Months were the bank reckoned a loss is marked with red circles. (a) Deposits, (b) withdrawals.

classify the company in Figure 7.1. This is very surprising, especially in our case since almost all the time series features (except zero withdrawals) indicate that this company is unhealthy. A close look yields that the company has very large mean income (26 MNOK) while paying a fairly small amount of tax. We estimate the size of a company solely as a function of the incomes; the size of this company is overestimated and the other features does not have strong enough response to overturn the size estimate.

Next, we turn to the remaining two companies misclassified as false negatives by the Naive Bayes and PD rating; illustrated in Figure 7.2 on the facing page.

The deposits and withdrawals graphs in Figure 7.2 on the next page (c) and (d) is quite similar to the ones in Figure 7.1, the only difference is that the amount of deposits and withdrawals in this case are consistent with the yearly incomes (mean 2.6 MNOK) yielding a lower size measure. Combined with the other features, the Naive Bayes classifies this company as unhealthy with confidence 1.0. The PD rating for this company is 0.015 (discrimination threshold at 0.025).

The PD-rating classifies the company in Figure 7.2 (a) and (b) as unhealthy with a confidence 0.102. This is a very high confidence considering the PD-rating. Naive Bayes classifies the company as *healthy* with confidence 1.0. This emphasizes the point already made: The PD-rating is more variable in its confidence, while the Naive Bayes is not (*cf.* Figure 6.3 and Figure 6.4)

7.2.1 Cross Validation Results

The cross validation results illustrates a broader difference between the Naive Bayes classifier and the PD rating. Considering the confusion matrices (Table 6.7 on page 50) and performance measures (Table 6.8 and Table 6.9 on page 50), the Naive Bayes classifier excels the PD rating in every metric.

The confidence plots in Figure 6.5 and Figure 6.6, on pages 51–51, confirms the plots calculated from the validation set: The PD rating is more willing to use the whole confidence spectrum than the Naive Bayes.

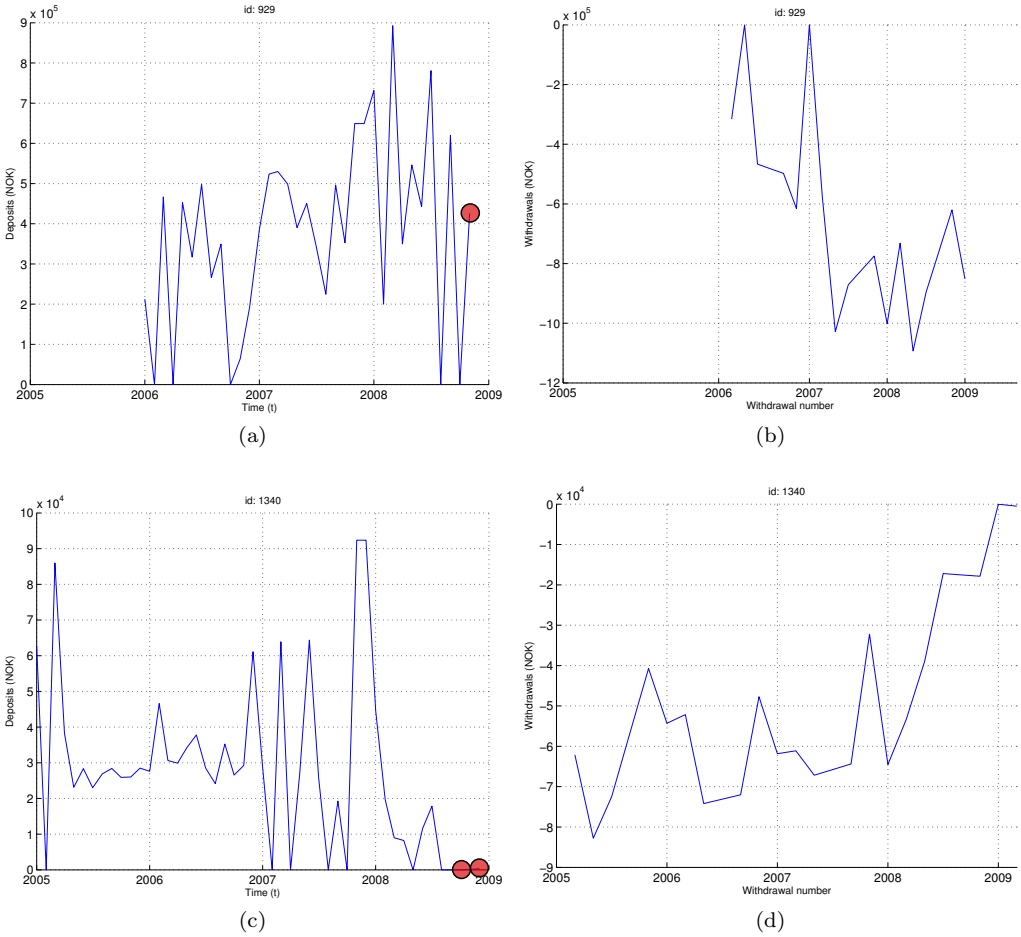


Figure 7.2: (a) and (b): Deposits and withdrawals for company misclassified as a false negative by Naive Bayes, but correctly classified by the PD-rating. (c) and (d): Deposits and withdrawals for a company misclassified as a false negative by the PD rating, but correctly classified by Naive Bayes.

7.3 Feature Evaluation

Our main contribution in Part I has been the features abstracting away irrelevant information present in the time series. The results presented in Chapter 6 suggests that we have succeeded in capturing relevant information to discriminate fairly well between the two classes of companies. By "fairly well" we mean good enough to perform a coarse-grained separation, and barely beat the PD-rating. Even though our classification model's performance on the data sets speaks for itself, we have for completeness performed an additional feature evaluation. The feature evaluation provides a more detailed view of which features were most important for performance. The result from this evaluation is presented in Section 6.3 and form the foundation for this section.

The most important finding from the feature evaluation is that none of the features seems to be dispensable without tolerating a loss in performance. As we can see in Table 6.5 on page 49, each feature is at least member of the best performing subset twice. The most vital features seems to be the global balance trend feature and the zero withdrawals feature, which were part of the best performing subset 10 out of 10 folds.

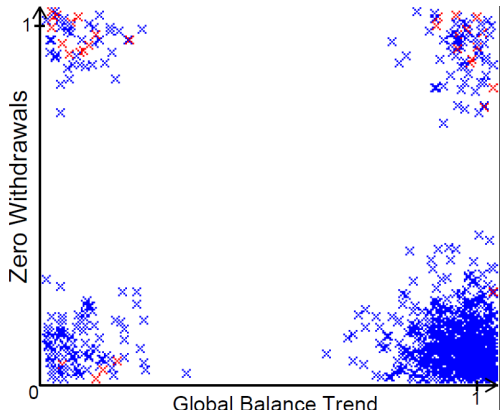
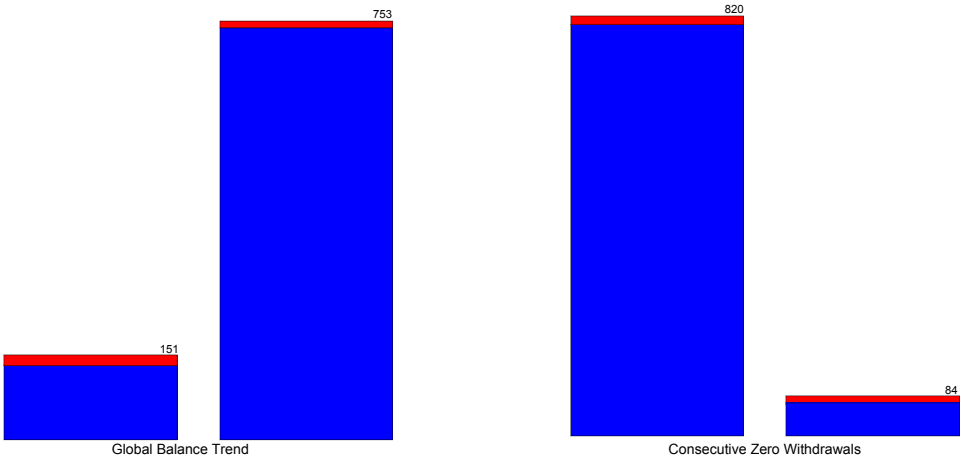


Figure 7.3: A combined plot of the two best performing features over the training set. The red crosses represents unhealthy companies and the blue crosses represents healthy companies.

In Figure 7.3, a plot of the two best performing features is included. The plot illustrates how well the global balance trend feature combined with the zero withdrawals feature separates the training set. Healthy companies are indicated with blue crosses, unhealthy companies are indicated with red crosses. Initially, since the features are discretized into binary features, all data points are located at four different coordinates $(0, 0)$, $(0, 1)$, $(1, 0)$, and $(1, 1)$. By adding jitter to the data points, that is zero mean Gaussian noise, we artificially scatter the points to see the different values, contrary to just see four crosses. Notice that the majority of financially sick companies is located in the second range of the zero withdrawals feature (zero withdrawals = 1), while the majority of the financially healthy companies are located in the lower right corner (zero withdrawals = 0 and global balance trend = 1).



(a) (b)

Figure 7.4: The global balance trend feature and the consecutive zero withdrawal feature after discretization.

The characteristics discriminating a good performing feature from a bad performing feature are subtle. For instance, the features displayed in Figure 7.4 appears to be of similar quality when studying the ratio between blue (healthy) and red (sick) companies. Both features have a range containing the majority of the companies and a range containing the minority of the companies. The ratios within the corresponding ranges are similar. Despite the resemblance, the

global balance trend feature is a top performer while the consecutive zero withdrawal feature is on the bottom, due to the feature evaluation. One common characteristic among the good performers is the low level of redundancy that particular feature provides to the classifier. Some of the low performers are redundant given another feature. For instance, a positive global deposit trend usually comes along with a negative global withdrawal trend (unless they are building up the Tax Withdrawal Account balance). Another example: consecutive zero deposits usually induces consecutive zero withdrawals (unless they have a solid balance on the Tax Withdrawal Account). These observations points in direction of the naive assumption behind the Naive Bayes classifier. As stated in Section 5.3, the Naive Bayes classifier treats the features as they were independent. When two, or more, of the features are redundant, the concept that is described by the redundant features will get more influence in the decision process, see [Witten and Frank, 2005, page 96]. Thus, the learning process is skewed, resulting in a negative impact on the classification performance. This may explain why seemingly good features are rated low by the feature evaluation: they are victims of the naive assumption.

As long as we are utilizing the Naive Bayes classifier some of the lowest performing features may be excluded without a substantial loss in performance. In Table 6.6 on page 49, we present the result from running a subset of our model over the validation set. This stripped down model consists of only the features which were a member of the best performing subset more than two times. Results from this run indicates a minimal loss in performance. The exclusion of the lowest performing features resulted in only four more *false positives*. In light of these results, we suggest that features that were member of the best performing subset only two or three times may be removed without any substantial impact on performance.

8 Conclusion

The results presented in Chapter 6 and discussed in Chapter 7 suggest that we have accomplished our goal to beat the PD-rating in the task of giving early warnings regarding unhealthy companies. Our model, based on a simple Naive Bayes classifier over a set of abstracted features, outperforms the PD-rating when running cross-validation over the training set and receives a lower $E(cost)$ than the PD-rating over the validation set. The PD-rating, on the other hand, is better to utilize the whole range when returning a confidence about a company's health status. This ability awards the PD-rating with a slightly better ROC, compared to our model, when measured over the validation set. Considering all the performance measures presented in Chapter 6 it is safe to say that our model beats the PD-rating.

The key accomplishment in Part I has been the feature generation. In order to get such results as reported above we must have succeeded in elevating the relevant information from the time series. The Feature Evaluation supports this notion, but suggests removing low performing features in order to save implementation costs. See Table 6.5 on page 49 for a complete list of how the features perform.

The model proposed can easily be used in a system for real time early warning, or flagging, of unhealthy companies. A trained model can be installed in such a system—as is—and will immediately provide good estimates of the financial health of companies.

Due to the success criteria stated in Section 1.4, we reckon that Part I already satisfies the goals for our thesis. In Part II we will try to sanitize the results further by using Gaussian Processes over the raw data. The companies we in Part I classify as unhealthy will form the data set for Part II. By utilizing information we, through our feature generation, have abstracted away, we hope to recover some of the companies that actually are healthy but which are reported sick by our system.

II *Fine-Grained Separation Using Gaussian Processes*

We used features to perform inference on the time series forming deposits and withdrawals in Part I; the features were used as abstractions capturing only relevant information from the time series. No measurements were made on how *complete* the features were with respect to the total amount of relevant information, but empirical results suggests that the features are at least as good as the system currently in place at SpareBank 1: the PD rating system. Measuring how complete the features are is not trivial since we have no notion of the total amount of relevant information that are encoded in the dataset. This part addresses the question of completeness; we perform inference based on the raw data forming the time series.

Artificial Neural Networks and fuzzy logic based models have gained wide popularity in classification and the modelling of non-linear relationships between variables. Industrial applications of such models ranges from fingerprint analysis [Sagar and Alex, 1999] and data mining [Lu et al., 1996] to automated trading of stocks and bonds [Deboeck, 1994], and learning arm trajectory models for robotic arms [Wada and Kawato, 1993]. Artificial Neural Networks have been especially popular in the context of bankruptcy prediction [Atiya, 2001; Odom and Sharda, 1990; Wilson and Sharda, 1994; Zhang et al., 1999]. Both the Artificial Neural Network and fuzzy logic based models are known as "black box"-models which are mainly identified using input/output data[Gregorcic and Lightbody, 2002]. The main difficulty with such models are the lack of transparency, which means that the resulting model does not provide any physical knowledge about the underlying system. As a consequence of the lack of transparency, incorporation of prior knowledge into such models is not possible. Neither Neural Networks nor fuzzy logic models use probabilities; this is not necessarily a weakness, but using probabilistic models we are provided with a notion of uncertainty. In the case of bankruptcy prediction, uncertainty estimates enables us to reason about how likely a bankruptcy is.

Gaussian Processes (GP) is a probabilistic model based on Bayesian inference. We gave a brief introduction to Bayesian inference when discussing the Naive Bayes classifier in Section 5.3, a thorough explanation is provided in the next chapter. Gaussian Processes are based on performing inference directly on the space of functions instead of, e.g. network parameters. This not only drastically reduces the number of parameters to optimize, but the optimized parameters provides information about the underlying function, for instance how smooth the function is assumed to be. The strongest advantage of Gaussian Processes is that it provides an analytic expression of

the model uncertainty. Given a set of models (configurations of different GP's) one is able to argue about how good each of the models are with respect to the observations.

9 Bayesian Inference

Both classifiers presented in Part I and Part II are based on Bayesian inference. We briefly recap the essence of Bayesian Inference in this chapter; the readers who are familiar with the subject may skip to Chapter 10.

9.1 Basics

Let us first define the problem. Given a dataset $\mathcal{D}$ of n observations, $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, n)\}$, where $\mathbf{x}$ denotes an input vector of dimension D and y denotes a scalar output or target. We wish to find a mapping $h : X \to Y$ so that we can associate new input values, $\mathbf{x}_*$, with an output y_*. We write $\mathcal{D}$ as $\mathcal{D} = (X, \mathbf{y})$ where X is the $D \times n$ input matrix and $\mathbf{y}$ is the associated targets.

The starting point when doing Bayesian inference is to define a model M for the problem at hand along with some free parameters π for the model. The free parameters π are given a *prior* distribution $p(\pi)$ which encodes our initial knowledge about the parameters. The quality of the inference depends on the accuracy of the model along with the value of its parameters, before any observations is made.

Specifying information about the relationships and distributions of variables and parameters may seem counterintuitive at first, after all, is not this part of the learning problem? This is partly true, however, if we know *nothing* about the problem at hand it is impossible to do inference since the next unknown state may be *anything*. Consider the advanced case of predicting future stock prices. If we have observed the closing price for a particular stock for the past two days to be 11.43 \$ and 14.58 \$, does this mean we can say anything about the closing price for the stock today? Do we believe that the probability for the price to be 19.0 \$ equals the probability for the price to be 10^{26} \$? Obviously, we can not predict the exact closing price for today, but given previous data we can say something about the probability.

The *likelihood* is the probability of the observed data given the parameters: $p(\mathcal{D}|\pi)$. Inference in the Bayesian framework is based on the *posterior distribution* over the parameters, and can be expressed as a function of the prior and likelihood via Bayes' rule:

$$\begin{aligned} p(\pi|\mathcal{D}) &= \frac{p(\mathcal{D}|\pi)p(\pi)}{p(\mathcal{D})} \\ p(\pi|\mathcal{D}) &\propto p(\mathcal{D}|\pi)p(\pi) \end{aligned} \tag{9.1}$$

where $\propto$ means "proportionate to" and $p(\mathcal{D})$ is the constant *marginal likelihood* and is only used for normalization. $p(\mathcal{D})$ can easily be obtained from the likelihood and prior by marginalization:

$$p(\mathcal{D}) = \int p(\mathcal{D}, \pi)\, d\pi = \int p(\mathcal{D}|\pi)p(\pi)\, d\pi$$

The posterior distribution tells us something about the model that most likely explains the data seen so far. Inference in the Bayesian framework is done by conditioning the unknown target y_* on the new data $\mathbf{x}_*$ and all previously observed data:

$$p(y_*|\mathbf{x}_*, \mathcal{D}) = \int p(y_*, \pi|\mathbf{x}_*, \mathcal{D})\, d\pi = \int p(y_*|\mathbf{x}_*, \pi)p(\pi|\mathcal{D})\, d\pi \tag{9.2}$$

The last part of Equation 9.2 is the likelihood times the posterior. Notice that we average over *all* possible parameter values weighted by the posterior probability in order to find the predictive distribution. This is in contrast to non-Bayesian schemes where one usually choose one value for π based on some criterion to define the predictive distribution.

9.2 Making Decisions

Given the predictive distribution $p(y_*|\mathbf{x}_*, \mathcal{D})$ we can decide for our belief of y_*. If we believe that the *cost* of every decision is equal (i.e. there is no worse to make decision A when we should have made decision B and vice versa), then we simply use the expected value of the predictive distribution of y_*.

On the other hand, if it is indeed worse to make decision A when we should have made decision B than the other way around, we need to incorporate a loss function $\mathcal{L}(y, y')$ and minimize the predictive distribution with respect to the loss:[1]

$$y_* = \int p(y_*|(x_*), \mathcal{D})\mathcal{L}(y_*, y')\, dy_*$$

Where y_* is the decision we make and y' is the correct decision. Note that if the predictive distribution and the loss function both are symmetric then the integral equals the expected value of the predictive distribution and the loss function: $\mathbb{E}[p(y_*|\mathbf{x}_*, \mathcal{D})\mathcal{L}(y_*, y')]$. We induce this notion of cost by wrapping the classifier with MetaCost, discussed in Section 5.4.

9.3 Coin toss example

Let us consider the example of predicting the probability for a *head* when tossing a coin. Given $m + n$ observed cases with m heads, we are concerned with predicting the probability of a head in the next case (with respect to the mapping problem, we can think of $\mathbf{x}$ as the $m + n$ observed cases, and y_* as the probability of a head in the next case). This is a binomial experiment with possible outcomes $\{T, H\}$, let m denote the number of heads and n denote the number of tails. Then the likelihood is the probability of the $m + n$ observations given the probability of a head, $\pi = p(H)$.

$$p(m, n|\pi) = \binom{n+m}{m}\pi^m(1-\pi)^n$$

The posterior becomes

$$p(\pi|m, n) \propto \binom{n+m}{m}\pi^m(1-\pi)^n \cdot p(\pi)$$

The only thing left is to define a prior $p(\pi)$ in order to calculate the posterior and predictive distribution. It can be shown that if the prior $p(\pi)$ takes the form of a Beta distribution, then the posterior also takes the form of a Beta distribution [Rasmussen and Williams, 2006]. We let the prior be uniform, this is a special case of the Beta distribution with parameters $\alpha = \beta = 1$, and is illustrated in Figure 9.1 on the next page.

Let us assume the following observations: *TTHHTTTHTT*, $m = 3$, $n = 7$, the posterior beta distributions have parameters $\alpha = 1 + m$ and $\beta = 1 + n$:

$$p(\pi|m, n) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\pi^m(1-\pi)^n = \frac{\Gamma(12)}{\Gamma(4)\Gamma(8)}\pi^3(1-\pi)^7$$

Figure 9.2 on the facing page illustrates the posterior distribution. The predictive distribution becomes

$$p(y_*|m, n) = \int \pi p(\pi|m, n)\, d\pi = \frac{\Gamma(12)}{\Gamma(4)\Gamma(8)} \cdot \frac{\Gamma(5)\Gamma(8)}{\Gamma(13)} = \frac{1}{3}$$

Thus, we believe that the probability for the next coin flip to be a head is 1/3.

[1] The loss function relates to the utility function in economics; loss is negative utility. In economics, one is concerned with maximizing expected utility while in statistics one is concerned with minimizing expected loss. Therefore, economists are optimists and statisticians are pessimists.

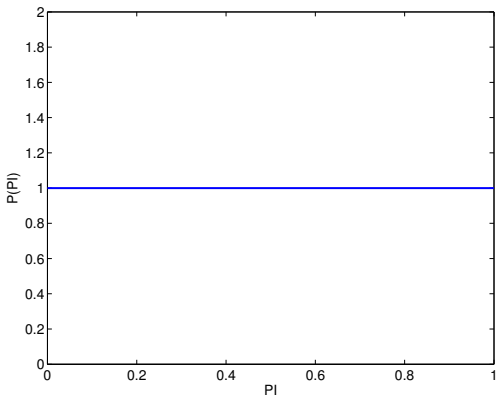


Figure 9.1: Prior distribution for $p(\pi)$. This is the same as stating "every value for π is equally reasonable" as our initial belief.

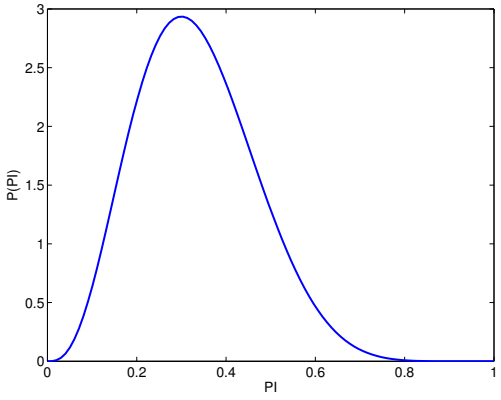


Figure 9.2: Posterior distribution of π. Notice the difference from the prior after accounting for the observations.

9.4 Summary

The goal of Bayesian inference is the posterior distribution which captures everything we know. We can use this distribution to do inference about future values for a parameter or to make predictions for y_* given some test inputs $\mathbf{x}_*$.

The Bayesian framework obeys the likelihood principle: conclusions depend only on the likelihood of the observations (and the explicit model assumptions). The explicit prior distribution is subject to most of the criticism of the Bayesian framework. The prior incorporates prior knowledge about the domain in the model, predictions becomes biased with respect to the prior. The question is whether you should incorporate prior knowledge in the model or not. This has been debated by statisticians for centuries and we will not delve deep into the discussion. We settle for that not knowing anything about the prior probabilities does not equal that we accept the prior probabilities to be anything (as illustrated by the stock example on page 65). In our case we *have* knowledge of the parameters of the model. This will become clear when we discuss Gaussian Processes in Part II.

The major problem when doing Bayesian inference is solving the integral for the predictive distribution. If the likelihood and prior does not have certain properties, then solving this integral analytically becomes intractable and we need to settle for approximations.

10 Gaussian Processes Basics

Remember from Chapter 9 that there was three important distributions when doing Bayesian inference, (1) the prior $p(\pi)$, (2) the likelihood, $p(\mathcal{D}|\pi)$, and (3) the posterior, $p(\pi|\mathcal{D}) \propto p(\mathcal{D}|\pi)p(\pi)$, where π was the parameter of the model and $\mathcal{D}$ was the observed data, $\propto$ means "proportionate to". In the coin toss example in Chapter 9 we used a uniform distribution as the prior distribution for the parameter π. The reason for this choice was deliberate as the posterior distribution then took the form of a Beta distribution which is easy to integrate. Choosing a prior on the basis of mathematical properties and not initial knowledge however, seems to contradict the very principle of the prior distribution. If we were to choose a prior that makes the integrals necessary for inference intractable, then the Bayesian framework is in practice inadequate for the problem at hand. In other words, when deciding to use Bayesian inference one should ask whether or not the prior knowledge of the domain can be expressed by a given distribution. A prior $p(\pi)$ is *conjugate* to the likelihood $p(D|\pi)$ if the posterior takes the same form as the prior. In Chapter 9 we could choose an arbitrary Beta distribution as the prior instead of a uniform distribution to achieve the same results. However, using a uniform distribution seems more appropriate as the uniform distribution is a special case of the Beta distribution and it is easier interpreted in the domain of coin tossing. Gaussian Process models is a family of statistical models where the likelihood is assumed to be Gaussian and the integrals needed to perform predictions turns into simple linear algebra. Further, the prior can take a wide variety of shapes without loosing the analytical tractable expressions for the posterior.

We start our discussion of Gaussian Processes models with a short summary of the Gaussian distribution; an example is provided where the multivariate Gaussian distribution is is used to solve a linear regression problem in a parametric model. We then show how to extend the parametric model into a non-parametric model where a Gaussian Process is used to model a distribution over function. Finally we discuss the covariance function of the Gaussian Process and how learning and model selection is performed.

10.1 The Gaussian Distribution

The Gaussian Distribution (or Normal Distribution) $\mathcal{N}(x; \mu, \sigma)$ is given by

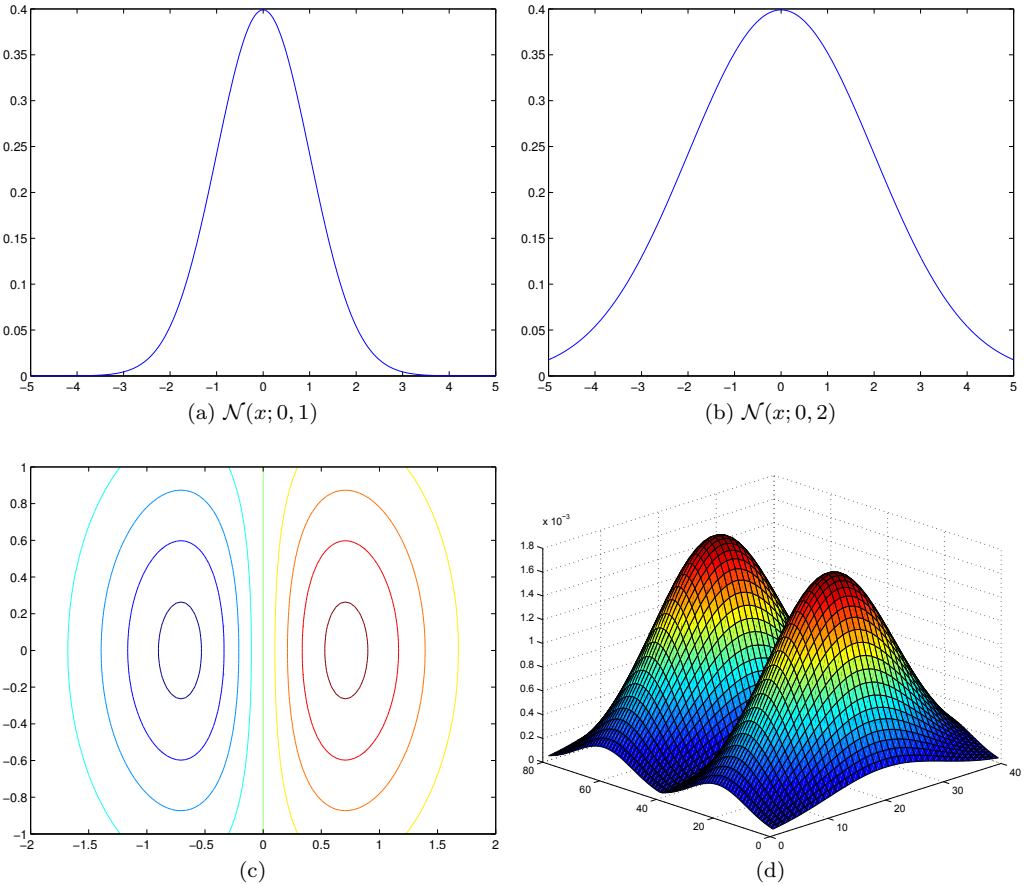
$$\mathcal{N}(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

where x is the random variable and μ and σ^2 are the mean and variance respectively. In the general case, we often allow x to have any number of dimensions giving rise to the multivariate Gaussian distribution $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \Sigma)$:

$$f_X(x_1, \ldots, x_N) = \frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})) \tag{10.1}$$

where $\boldsymbol{\mu}$ is the mean vector and Σ is the covariance matrix for the set of inputs X. We will use upper case letters to denote sets of variables and bold face letters to denote vectors unless otherwise clear from the context. The eigenvalues of Σ are the square of the variances in the direction of the eigenvectors. Figure 10.1 on the following page illustrates a one- and two dimensional Gaussian distribution; note that we usually work with distributions in much higher dimensions, but these distributions are hard to visualize.

The Gaussian distribution has several nice properties which makes it a useful distribution for the prior (in the general case):



(a) $\mathcal{N}(x; 0, 1)$ (b) $\mathcal{N}(x; 0, 2)$

(c) (d)

Figure 10.1: Different plots of Gaussian distributions. (a) and (b) 2D plots of one-dimensional Gaussians with different standard deviation. (c) Contour plot of 2 two-dimensional Gaussians (d) 3D plot of 2 two-dimensional Gaussians. We will use contour plots throughout this thesis as it is easiest to interpret.

Conditional property If $P(A)$ is normally distributed, then $P(B|A)$ is also normally distributed. Here, A and B are stochastic variables, and not sets of variables.

Marginalization property $P(X) = \int P(X, Y)\, dY$. We can calculate $P(X)$ from the joint distribution of X and Y regardless the dimension of that distribution, where X and Y are stochastic variables.

The marginalization property of the Gaussian distribution allows us to reason about $P(X)$ from $P(X, Y)$ regardless of the dimensionality of $P(Y)$. This is useful when $P(Y)$ is of a dimensionality which is impossible or impractical to store on a computer.

10.2 A Parametric Example: Curve Fitting

Before we discuss the Gaussian Process model, let us see how we can use the Gaussian distribution in the original Bayesian framework to do inference. We use as our example problem the problem of finding parameters α and β that makes $f(x)$ fit the observed data:

$$f(x) = \alpha x + \beta \tag{10.2}$$

This equation can be more compactly written as

$$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}, \qquad y = f(\mathbf{x}) + \varepsilon \tag{10.3}$$

where $\mathbf{x} = [x, 1]$ and $\mathbf{w} = [\alpha, \beta]$. Notice that we introduce the variable y, this variable is the actual observed data which is the underlying function $f(x)$ plus some additive noise. We assume that this noise follows an independent identically Gaussian distribution with zero mean and variance σ_n^2:

$$\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$$

We could use any other noise model, but for the sake of example, this model makes computations easier. Given a training set $\mathcal{D} = (X, \mathbf{y})$, where X denotes the set of input vectors $\mathbf{x}_i$ and $\mathbf{y}$ is a vector of target values y_i for each input vector. We define the likelihood $p(\mathbf{y}|X, \mathbf{w})$ and the prior $p(\mathbf{w})$ to be Gaussian, further, we assume the observations y_i to be independent of $\mathbf{x}$ given $\mathbf{w}$ (*cf.* Figure 10.3 on page 73).

$$\begin{aligned} p(\mathbf{y}|X, \mathbf{w}) &= \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{w}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_n} \exp(-\frac{(y_i - \mathbf{x}_i^\top \mathbf{w})^2}{2\sigma_n^2}) \\ &= (2\pi\sigma_n)^{-n/2} \exp(-\frac{1}{2\sigma_n^2}|\mathbf{y} - X^\top \mathbf{w}|^2) = \mathcal{N}(X^\top \mathbf{w}, \sigma_n^2 I) \end{aligned}$$

where $|\mathbf{z}|$ is the Euclidean length of the vector $|\mathbf{z}|$. Notice that we only allow for variance in each observation by multiplying σ_n^2 with the identity matrix. We give the prior a normal distribution with zero mean and covariance matrix Σ_p:

$$\begin{aligned} p(\mathbf{w}) &\sim \mathcal{N}(0, \Sigma_p) \\ &= (2\pi)^{-N/2} |\Sigma_p|^{-1/2} \exp(-\frac{1}{2}\mathbf{w}^\top \Sigma_p^{-1} \mathbf{w}) \end{aligned} \tag{10.4}$$

The posterior distribution is given by Bayes rule $p(a|b) = p(b|a)p(a)/p(b)$ where $p(b)$ is constant. The exact calculation of the posterior is not particularly interesting, but is provided below for completeness.

$$\begin{aligned} p(\mathbf{w}|X, \mathbf{y}) &\propto p(y|X, \mathbf{w})p(\mathbf{w}) \\ &\propto \exp\big(-\frac{1}{2\sigma_n^2}(\mathbf{y} - X^\top \mathbf{w})^\top (\mathbf{y} - X^\top \mathbf{w})\big) \exp\big(-\frac{1}{2}\mathbf{w}^\top \Sigma_p^{-1} \mathbf{w}\big) \\ &\propto \exp\big(-\frac{1}{2}(\mathbf{w} - \bar{\mathbf{w}})^\top (\frac{1}{\sigma_n^2} X X^\top + \Sigma_p^{-1})(\mathbf{w} - \bar{\mathbf{w}})\big) \end{aligned} \tag{10.5}$$

where $\bar{\mathbf{w}} = \sigma_n^{-2}(\sigma^{-2} X X^\top + \Sigma_p^{-1})^{-1} X \mathbf{y}$, and we recognize the form of the posterior as a Gaussian with mean $\bar{\mathbf{w}}$ and covariance matrix A^{-1}:

$$p(\mathbf{w}|X, \mathbf{y}) \sim \mathcal{N}(\bar{\mathbf{w}}, A^{-1})$$

where $A^{-1} = \sigma_n^{-2} X X^\top + \Sigma_p^{-1}$. Given new test data $\mathbf{x}_*$, we calculate the predictive distribution for y_* by calculating $p(y_*, \mathbf{w}|\mathbf{x}_*, X, \mathbf{y})$ and integrate over the weights.

$$\begin{aligned} p(y_*|\mathbf{x}_*, X, \mathbf{y}) &= \int p(y_*|\mathbf{x}_*, \mathbf{w}) p(\mathbf{w}|X, \mathbf{y})\, d\mathbf{w} \\ &= \mathcal{N}\big(\frac{1}{\sigma_n^2}\mathbf{x}_*^\top A^{-1} X \mathbf{y},\ \mathbf{x}_*^\top A^{-1} \mathbf{x}_*\big) \end{aligned} \tag{10.6}$$

The product in the integral above equals the likelihood weighted by the posterior. Since the product of two Gaussians is a non-normalized Gaussian (*cf.* Appendix B.1), the integral in Equation 10.6 equals the expected value or the mode of the distribution which is analytically tractable. Notice that we calculate prediction based on all possible values for $\mathbf{w}$, this is in contrast to other inference methods, for instance maximum likelihood.

10.3 Gaussian Processes Definition

The Gaussian Process model is an extension of the Bayesian framework to a model with multivariate Gaussian distributions of infinite dimensions. This is achieved by replacing the mean and covariance matrix in Equation 10.1 with their infinite counterparts: functions (we can think of a function as an infinitely long vector). So, $\boldsymbol{\mu}$ is replaced by a mean function $m(\mathbf{x})$ and Σ is replaced by a covariance function $k(x, x')$. Just as a given row and column in Σ specifies the covariance between two dimensions, the function k calculates the covariance between dimension x and x'.

Using the normal distribution with finite number of dimensions, we say that $p(\mathbf{x})$ is a distribution over different values of $\mathbf{x}$. In the Gaussian Process model, we say that $p(f(\mathbf{x}))$ is a distribution over *functions*.

Definition 1 (Gaussian Process)**.** A Gaussian Process is a collection of random variables, any finite number of which have a joint Gaussian distribution. A Gaussian Process is completely defined by a mean function m and a covariance function k.

$$p(f(\mathbf{x})) = \mathcal{GP}(m(\mathbf{x}),\ k(x, x'))$$

To get an intuition about what this distribution looks like, we can draw some samples over a finite interval and plot them. We consider the following Gaussian Process

$$p(f(\mathbf{x})) = \mathcal{GP}\big(m(\mathbf{x}) = 0,\ k(x, x') = \exp(-\frac{1}{2}(x - x')^2)\big) \tag{10.7}$$

for which a subset of a sample function values $\mathbf{f}$, $\mathbf{f} = \{f(x_1), f(x_2), \ldots, f(x_n)\}$ is normally distributed, $\mathbf{f} \sim \mathcal{N}(0, \Sigma)$; given two dimensions p and q we have $\Sigma_{pq} = k(x_p, x_q)$. Some sample functions drawn from this distribution is shown in Figure 10.2, details on how to sample a Gaussian Process is described in Appendix B.2.

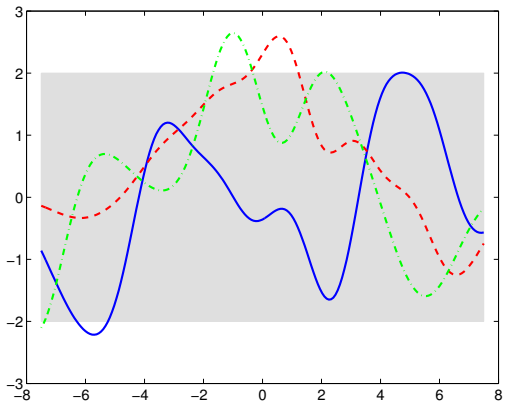


Figure 10.2: Three samples drawn from the GP prior, Equation 10.7. The functions has been drawn as lines by sampling with small interval Δx. The shaded area corresponds to the pointwise mean plus two times the standard deviation, i.e. a 95 % confidence interval.

Observe that the functions in Figure 10.2 have some common traits: they are all smooth and they change at approximately the same time: every unit step on the x-axis. All these properties are embodied in the covariance function $k(x, x')$ which we will discuss further in Section 10.5.

A graphical model of a Gaussian Process evaluated at three points is illustrated in Figure 10.3 on the facing page. We assume that the observations y is a function of inputs $\mathbf{x}$, but has been modified with noise by an underlying process f. Our task in Gaussian Process regression and

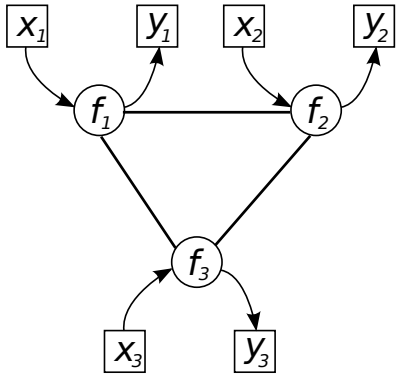


Figure 10.3: A graphical model of a Gaussian Process. Squares represents observed data, f is the underlying unknown process (function). We assume noisy observations y_i; function values f are fully connected. For the parametric model, replace $\mathbf{f}$ with $\mathbf{w}$.

classification is to estimate the most probable values for f given several observations pairs $(\mathbf{x}_i, y_i)$. The more we know about the underlying process generating the outputs, the better will the prediction for a new observation y_* be given a *test input* $\mathbf{x}_*$. Notice the absence of links between two outputs, e.g. y_1 and y_2, this means that the observations y_1 and y_2 are independent if we know the latent function values $\mathbf{f}_1$ or $\mathbf{f}_2$. Also, an output y_i is independent of $\mathbf{x}_i$ given $\mathbf{f}_i$. The independence assumptions means that $p(\mathbf{y}|\mathbf{f})$ can be expressed as a product $\prod_i p(y_i|\mathbf{f}_i)$, this will become clear later.

Unfortunately, we can not say anything about the dependencies between function values before we have learned the hyperparameters. Prior to learning, all values for the underlying function f is assumed to be dependent (hence the undirected bold links).

10.4 A Non-Parametric Model

In Section 10.2 we saw how to use the Bayesian framework with Gaussian distributions to fit a linear function $f(x)$ to some observed data points where $f(x)$ was parameterized by $\mathbf{w}$. In a *non-parametric* model, the "parameters" are the function itself. We do not have a fixed model like the one in Equation 10.2, instead we perform inference directly on the space of all possible functions that maps X to $\mathbf{y}$. With respect to the graphical model in Figure 10.3, we search for the most probable function $\mathbf{f}$ with the characteristics in Definition 1.

To illustrate how this works, we redo the example from Section 10.2, but this time we replace all occurrences of $\mathbf{w}$ with $f(x)$. Now, $f(x)$ has nothing to do with Equation 10.2, it is simply any possible function. In this model, the prior goes from a distribution over values for $\mathbf{w}$, $p(\mathbf{w})$, to a distribution over functions, $p(f(x))$. Since a Gaussian Process is exactly that—a distribution over functions—we can use it to model the prior.

$$p\big(f(x)\big) = \mathcal{GP}\big(m(\mathbf{x}) = 0,\ k(x_p, x_q) + \sigma_n^2 \delta_{pq}\big) \tag{10.8}$$

$$p\big(\mathbf{y}|X, f(x)\big) \sim \mathcal{N}(\mathbf{f}, \sigma_n^2 I) \tag{10.9}$$

where δ_{pq} is the Kronecker delta function.[1] The likelihood is Gaussian shaped centered at the training data X, $\mathbf{f} = \{f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n) \mid \mathbf{x}_i \in X\}$ with mean $m(\mathbf{x})$. The variance equals the noise variance for every observation y_i.

The unnormalized posterior distribution is given as always by multiplying together the prior and the likelihood. The product of two finite dimension Gaussians is another (unnormalized) finite

[1] $\delta_{pq} = \begin{cases} \delta_{pq} = 1, & \text{if } p = q \\ \delta_{pq} = 0, & \text{if } p \neq q \end{cases}$

dimension Gaussian. The same goes for a Gaussian Process: the product of an infinite dimensional Gaussian with at finite dimension Gaussian results in an infinite dimensional Gaussian, or a Gaussian Process. Thus, the posterior can be expressed as:

$$\begin{aligned} p(f(x)|\mathbf{x},\mathbf{y}) = \mathcal{GP}(m_{\text{post}}(x) &= k(x,X)[K(X,X)+\sigma_n^2 I]^{-1}\mathbf{y}, \\ k_{\text{post}}(x,x') &= k(x,x') - k(x,X)[K(X,X)+\sigma_n^2 I]^{-1}k(X,x')) \end{aligned} \tag{10.10}$$

where m_{post} and k_{post} are the results after multiplying together two Gaussians, and $K(\cdot,\cdot)$ is the covariance matrix calculated by k. See Appendix B for details on the multiplication.

Let K be a shorthand for $K(X,X)$, the covariance matrix for training inputs, and $\mathbf{k}_*$ the covariance matrix between a single test input $\mathbf{x}_*$ and the training data X, $K(\mathbf{x}_*,X)$. The joint distribution of observations $\mathbf{y}$ and a new prediction $\mathbf{f}_* = f(\mathbf{x}_*)$ can be written as

$$\begin{bmatrix}\mathbf{y}\\ \mathbf{f}_*\end{bmatrix} \sim \mathcal{N}\Big(\mathbf{0}, \begin{bmatrix}K+\sigma_n^2 I & \mathbf{k}_* \\ \mathbf{k}_*^\top & k(\mathbf{x}_*,\mathbf{x}_*)\end{bmatrix}\Big) \tag{10.11}$$

We can consider Figure 10.2 on page 72 as different functions drawn from the prior. In order to do inference, we may simply draw functions from the prior, rejecting the ones that disagree with the observations. The resulting set of functions which conform the training data is then used to calculate the distribution for $p(\mathbf{f}_*|\mathbf{y},\mathbf{x}_*)$. Of course, there is no reason why a drawn function should conform the training data, so this process is quite tedious. Instead, we do as we would have done in the original Bayesian framework, find the joint distribution $p(\mathbf{f}_*,\mathbf{f}|\mathbf{y},\mathbf{x}_*)$ and marginalize out $\mathbf{f}$ to obtain $p(\mathbf{f}_*|\mathbf{y},\mathbf{x}_*)$. Since the likelihood and posterior is Gaussian, this distribution is also Gaussian. Formally, we obtain the predictive distribution from Equation 10.11 (by utilizing Equation B.3):

$$p(\mathbf{f}_*|X,\mathbf{y},\mathbf{x}_*) \sim \mathcal{N}\big(\bar{\mathbf{f}}_*, \mathbb{V}(\mathbf{f}_*)\big), \quad \text{where} \tag{10.12}$$

$$\bar{\mathbf{f}}_* = \mathbb{E}(\mathbf{f}_*|X,\mathbf{y},\mathbf{x}_*) = \mathbf{k}_*^\top (K+\sigma_n^2 I)^{-1}\mathbf{y} \tag{10.13}$$

$$\mathbb{V}(\mathbf{f}_*) = k(\mathbf{x}_*,\mathbf{x}_*) - \mathbf{k}_*^\top (K+\sigma_n^2 I)^{-1}\mathbf{k}_* \tag{10.14}$$

Notice that we can calculate the predictive distribution solely by linear algebra, in contrast to the Bayesian framework which calls for solving integrals (e.g. Equation 10.6). Figure 10.4 on the facing page illustrates the posterior conditioned on some training data $\mathcal{D}$ marked with crosses. The shaded area is the 95 % confidence interval for the function.

The results from the deduction in this section is the predictive distribution in Equation 10.12. Using this distribution, we can achieve the same results as in the parametric example in Section 10.2; the model is now not explicitly formulated like in Equation 10.3, but through the covariance function of the Gaussian Process, Equation 10.8. If we believe that the observations are linear we can explicitly communicate this through the covariance function; more on this in Section 10.5. First, let us analyze the predictive distribution in Equation 10.12; the mean of the predictive distribution is given by $\bar{\mathbf{f}}_* = \mathbf{k}_*^\top (K+\sigma_n^2 I)^{-1}\mathbf{y}$. The mean can be expressed as a linear combination

$$\begin{aligned} \bar{\mathbf{f}}_* &= \mathbf{k}_*^\top (K+\sigma_n^2 I)^{-1}\mathbf{y} \\ &= \sum_n \boldsymbol{\beta} y_n \end{aligned} \tag{10.15}$$

for $\boldsymbol{\beta} = \mathbf{k}_*^\top (K+\sigma_n^2 I)^{-2}$, the mean is a linear combination of the observations. In statistics, this is called a "linear smoother" (remember our discussion about moving averages, Section 4.4 on page 21). We can also express Equation 10.15 as

$$\begin{aligned} \bar{\mathbf{f}}_* &= \mathbf{k}_*^\top (K+\sigma_n^2 I)^{-1}\mathbf{y} \\ &= \sum_n \boldsymbol{\alpha}_n \mathbf{k}_*^\top \\ &= \sum_n \boldsymbol{\alpha}_n k(\mathbf{x}_*, X) \end{aligned} \tag{10.16}$$

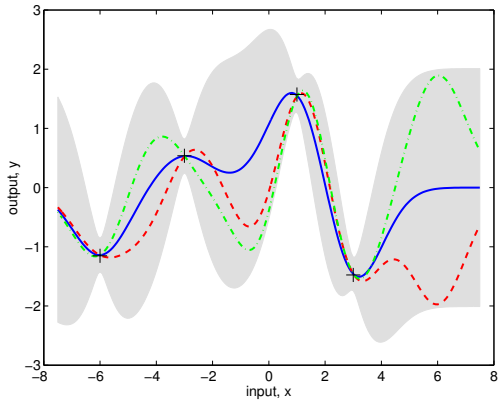


Figure 10.4: Three samples drawn from the posterior distribution (prior conditioned on noisy observations), Equation 10.10. The covariance function used is the squared exponential: $k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{2}|\mathbf{x} - \mathbf{x}'|^2)$. The shaded area represents the pointwise mean plus minus two times the standard deviation for each input value (corresponding to a 95 % confidence interval). The mean function is drawn with a solid blue line, two other realizations of the data are drawn with dotted and dash-dotted red and green lines.

for $\boldsymbol{\alpha} = (K + \sigma_n^2)^{-1}\mathbf{y}$. We can view the mean as a linear combination of the covariance function evaluated at every training input for a particular test input $\mathbf{x}_*$. The notation in Equation 10.16 is commonly used in the kernel literature. For the readers who are familiar with Support Vector Machines, this is the exact same form as the kernel used in the SVM. Just as the SVM, Gaussian Processes are linear in the feature space defined by the kernel.

The variance is the difference between two terms, $\mathbb{V}(\mathbf{f}_*) = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1}\mathbf{k}_*$. The first term is the prior variance for the test inputs and the second term tells us how much the data X has explained. If the correlation between the test inputs and the training data is low, then the covariance $\mathbf{k}_*$ should be close to zero. The result is that the second term is close zero and that the posterior variance is close to the prior variance. Notice that if the second term had been negative, we would end up with a higher variance (uncertainty) about $\mathbf{f}_*$ than we had before any observations was made. This is clearly a contradiction; in fact, it is a *requirement* that the covariance function is *positive semi-definite*. We will elaborate this a bit in Section 11.2, for now it means simply that the second term above can not be negative. Note that the variance of $\mathbf{f}_*$ does not depend on any observations $\mathbf{y}$, but only on the observed inputs X (embodied in $\mathbf{k}_*$).

10.5 The Covariance Function

So far we have seen how to escape an explicit formulation of the model as in Section 10.2 by using a non-parametric model. A non-parametric model calls for a distribution over functions and we used a Gaussian Process to model this distribution. The space of functions which are considered are restricted by the mean and covariance functions of the Gaussian Process. We usually choose the mean function to be zero, this is done in order to simplify computations; however, choosing another function for the mean function has surprisingly little effect on the overall fit [Rasmussen and Williams, 2006, Chapter 2.7]. In this section we discuss the covariance function and how we can use it to communicate prior knowledge of the model. The covariance function defines properties of the functions which are drawn from the Gaussian Process.

10.5.1 A Linear Covariance Function

Assume that we expect the relationship between X and $\mathbf{y}$ to be linear. For the sake of example, let us assume zero noise, so $f(x) = y$. We choose a Gaussian prior for the parameters a and b with variance α and β respectively.

$$f(x|a,b) = ax + b, \qquad A \sim \mathcal{N}(0,\alpha), \quad B \sim \mathcal{N}(0,\beta) \tag{10.17}$$

where $f(x)$ is considered a distribution over functions where x are fixed and the parameters a and b are drawn from the prior distributions. The mean and covariance functions for the Gaussian Process are calculated directly from Equation 10.17:

$$\begin{aligned} m(x) &= \mathbb{E}[f(x)] \\ &= \iint f(x)p(a)p(b)\,da\,db = x\int ap(a)\,da + \int bp(b)\,db \\ &= 0 \end{aligned}$$

$$\begin{aligned} \mathrm{Cov}[f(x), f(x')] = k(x,x') &= \mathbb{E}[(f(x)-0)(f(x')-0)] \\ &= \iint (ax+b)(ax'+b)p(a)p(b)\,da\,db \\ &= xx'\int a^2p(a)\,da + \int b^2p(b)\,db + (x+x')\int abp(a)p(b)\,da\,db \\ &= \alpha xx' + \beta \end{aligned} \tag{10.18}$$

The mean function is not surprisingly zero. This is intuitive as we can think of the distribution over functions $f(x)$ as having an equal amount of functions with mean above zero as mean below zero (the normal distribution is symmetric).

If we define a Gaussian Process $\mathcal{GP}(0,\ \alpha xx' + \beta)$ we end up with a distribution over linear functions. α and β are the parameters of the covariance function, often referred to as the *hyperparameters*.[1] Estimating values for the hyperparameters are part of the learning process.

10.5.2 A Non-Linear Covariance Function

The example in the previous section illustrates how to use a Gaussian Process to model linear regression. Of course, if you ever where to do linear regression, there are algorithms which are far more efficient. Let us consider a more interesting regression problem. This time we let $f(x)$ be the weighted sum of infinitely many Gaussian functions centered at every point on the x-axis

$$f(x) = \int_{-\infty}^{\infty} \gamma(u)\exp(-(x-u)^2)\,du, \quad \gamma(u) \sim \mathcal{N}(0,1),\ u \in \mathbb{R}$$

where $f(x)$ is considered a distribution over functions with x fixed and $\gamma(u)$ and u as parameters. Using this function as our regression model, we are able to estimate *any* differentiable function. The mean and covariance are calculated as before.

$$\begin{aligned} m(x) &= \mathbb{E}[f(x)] \\ &= \int \exp\big(-(x-u)^2\big)\int \gamma(u)p(\gamma)\,d\gamma\,du \\ &= 0 \end{aligned}$$

[1] We refer to the parameters of the covariance function as "hyperparameters" to emphasize that that they are parameters of a non-parametric model.

$$\begin{aligned}\mathrm{Cov}[f(x), f(x')] &= \mathbb{E}[(f(x)-0)(f(x')-0)] \\ &= \int \exp\big(-(x-u)^2 - (x'-u)^2\big)\, du \\ &= \int \exp\big(-2(u-\frac{1}{2}(x+x'))^2 + \frac{1}{2}(x+x')^2 - x^2 - x'^2\big)\, du \\ &\propto \exp(-\frac{(x-x')^2}{2}) \end{aligned} \tag{10.19}$$

If we use a Gaussian Process with the covariance function in Equation 10.19, it is the same as doing regression in a model in which there are infinitely many Gaussian "bumps" on the x-axis. This particular covariance function is called the *Squared Exponential* (SE) function—or in the kernel community, the Gaussian kernel.

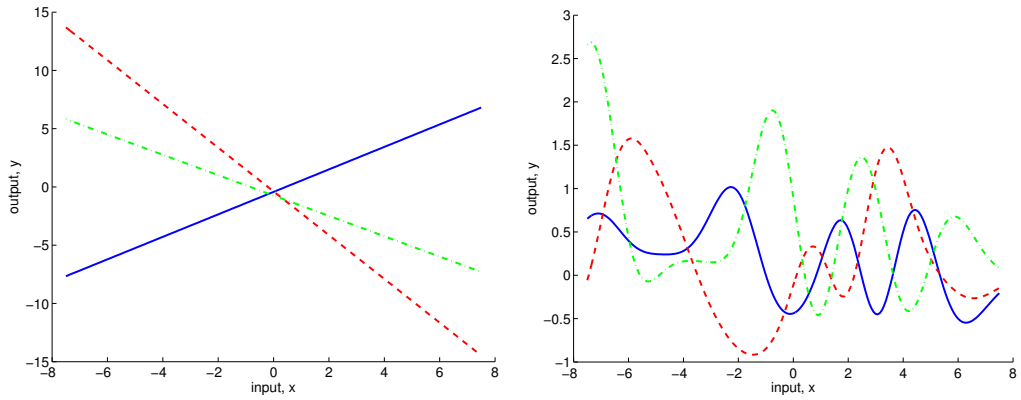
We make an important observation for both the covariance functions considered so far. The covariance between two *outputs* are calculated as a function of the *inputs* (this is true in the general case, *cf.* Equation 10.14). For the squared exponential, we see that if the squared distance between x and x' is high, then the covariance between $f(x)$ and $f(x')$ is low, and vice versa. Thus, by using the squared exponential, we state that "points close in input space are correlated". The squared exponential is also infinitely differentiable, so we expect smooth curves between the observed outputs.

We most often encounter the squared exponential function parameterized by ℓ, σ_f^2 and σ_n^2:

$$k(x, x') = \sigma_f^2 \exp(-\frac{(x-x')^2)}{2\ell} + \sigma_n^2 \delta_{xx'} \tag{10.20}$$

ℓ is called the *length scale* parameter of the squared exponential, it allows us to define the semantics of points lying "near to each other". A long length scale means that we are willing to accept points with a large Euclidean distance as lying near to each other. A short length scale works in the opposite way. σ_f^2 adjusts the amplitude of the approximated function, and the noise variance is controlled by the Kronecker delta function as before (we assume independent noise).

In the Gaussian Process model we perform inference over the hyperparameters of the covariance function in addition to the covariance function itself. This is in contrast to other kernel machines, e.g. Support Vector Machines, where one consider the kernel (covariance function) as fixed. Some samples drawn from a Gaussian Process with linear and the squared exponential covariance functions are shown in Figure 10.5



(a) $\alpha = 1,\ \beta = 1$ (b) $\sigma_f^2 = 1,\ \ell = 1,\ \sigma_n^2 = 0.1$

Figure 10.5: Random samples drawn from a zero mean Gaussian Process with covariance functions in (a) and (b) as Equation 10.18 and Equation 10.20 respectively.

11 Advanced Gaussian Processes

We saw in the previous chapter how the solution for the predictive distribution had a simple analytical form by using a non-parametric model with a Gaussian Process prior; this in contrast to the parametric model with a Gaussian Distribution prior. Common for the two models were that they are both models for the regression problem: given a set of training data $\{\mathbf{x}_i, y_i \,|\, \mathbf{x}_i \in \mathbb{R}^D, y_i \in \mathbb{R}\}$, find y_* given a test input $\mathbf{x}_*$. In classification, the problem is stated slightly differently: given a set of training data $\{(\mathbf{x}_i, y_i) \,|\, \mathbf{x}_i \in X, y_i \in \mathbf{y}\}$, find a mapping $h : X \to \mathbf{y}$ that maps any input $\mathbf{x}_i$ to its true output label y_i for a *finite* set of class labels $\mathbf{y}$. The solution to the classification problem does *not* have an analytical form and needs to be approximated. We find several approaches for approximations in the literature: Laplace Approximation [Williams and Barber, 1998a], Expectation Propagation [Minka, 2002], and Variational methods [Bernardo et al., 2003; Opper and Archambeau, 2009], each with their own strengths and weaknesses.

The space of functions defined by a Gaussian Process is fully specified by the covariance function (assuming the mean function to be zero). Thus, the quality of the regression and classification depends on the choice of covariance function and the hyperparameters for that function; this is referred to as the *model selection problem*. We introduce this chapter with a discussion of Gaussian Process classification and its approximations. Then, we present some covariance functions along with their characteristics, and finally we show how to do learning over the space of covariance functions and the corresponding hyperparameters.

11.1 Classification

A natural starting point when discussing classification is the joint probability function $p(y, \mathbf{x})$, where $\mathbf{x}$ is the inputs and y is the class label, $y \in \mathcal{C}$, $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$. Using Bayes' theorem, we can express the joint distribution as either $p(\mathbf{x}|y)p(y)$ or $p(y|\mathbf{x})p(\mathbf{x})$. If we know the class conditional distribution $p(\mathbf{x}|y)$ we can calculate the posterior distribution by Bayes' formula: $p(y|\mathbf{x}) = p(\mathbf{x}|y)p(y)/p(\mathbf{x})$. Alternatively we can model the posterior explicitly. These two approaches are referred to in the literature as the *generative* and *discriminative* approach respectively.

In order to do inference we have to model either $p(\mathbf{x}|y)$ or $p(y|\mathbf{x})$. Using the generative approach in the Bayesian framework, we usually model $p(\mathbf{x}|y)$ (the likelihood) with a known distribution, and then place a conjugate prior over $p(y)$ in order to calculate $p(y|\mathbf{x})$. If we additionally assume independent inputs, we arrive at the Naive Bayes model discussed in Chapter 5.

For the discriminative case we could use a regression model to estimate $p(y|\mathbf{x})$ and then pass the result through a *response function* g, $g : \hat{p} \to [0, 1]$ to obtain a valid probability estimate. The response function must have domain in $\mathbb{R}$ and range $[0, 1]$. A common choice is to combine a linear model $h(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}$ with a sigmoid function,[1] for instance the logistic function.

$$p(y = c_i|\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{x}^\top \mathbf{w})}$$

By replacing $\mathbf{x}^\top \mathbf{w}$ with a function $f(\mathbf{x})$ drawn from some space over functions, we can use Gaussian Process regression in order to do classification (demonstrated later). Figure 11.1 on the following page illustrates the logistic function.

The question of which approach to use remains unsolved and should be influenced by prior knowledge of the problem at hand. The generative approach is attractive because we have access to $p(\mathbf{x})$ (from marginalization of y from $p(\mathbf{x}|y)p(y)$), thus we are able to account for outliers and

[1] A sigmoid function is any monotonically increasing bounded s-shaped function.

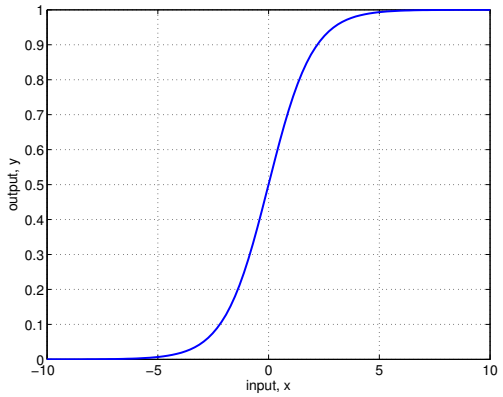


Figure 11.1: A Sigmoid function

missing data in our predictions. This comes at the cost of defining a model for $p(\mathbf{x}|y)$, which is usually very hard, especially if $\mathbf{x}$ is high dimensional; using the generative approach may mean we are trying to solve a harder problem than we need to (Naive Bayes solves the problem by making preposterous assumptions about the data). The discriminative approach is appealing in that we are modelling exactly what we want and is the method of choice for this thesis. For a more comprehensive discussion of the generative and discriminative approach, we redirect the interested reader to Ripley [1996, Chapter 2.1].

We introduce Gaussian Process classification in a similar manner as we did with Gaussian Process regression. We start by giving a parametric example using only Gaussian distributions, and then replace the parameters in the parametric model with a distribution over functions resulting in a non-parametric solution.

11.1.1 A Parametric Example

Let us consider binary classification. We follow the Support Vector Machine (SVM) literature and let $y \in \{-1, +1\}$. The likelihood $p(y|\mathbf{x}, \mathbf{w})$ is a linear combination of inputs $\mathbf{x}$ and weights $\mathbf{w}$ passed through a sigmoid function. The deduction in this section parallels the deduction given Section 10.2 on page 70.

$$p(y = +1|\mathbf{x}, \mathbf{w}) = \Phi(\mathbf{x}^\top \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{x}^\top \mathbf{w})} \tag{11.1}$$

Since the sigmoid function is symmetric we have $\Phi(-z) = 1 - \Phi(z)$ and thus $p(y = -1|\mathbf{x}, \mathbf{w}) = 1 - p(y = +1|\mathbf{x}, \mathbf{w})$. Equation 11.1 can be written more compactly as

$$p(y_i|\mathbf{x}, \mathbf{w}) = \Phi(y_i \mathbf{x}^\top \mathbf{w}) \tag{11.2}$$

Note that in contrast to the regression case, the likelihood is now not normally distributed. We assume as before a given dataset $\mathcal{D} = (X, \mathbf{y})$ and we define a prior distribution over the weights $\mathbf{w}$, $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma_p)$. The posterior distribution is given by Bayes' rule; recall the independence assumptions from Figure 10.3 on page 73: we assume independent observations y_i given f, (f corresponds to $\mathbf{w}$ in this case), and y_i independent from $\mathbf{x}_i$ given $\mathbf{w}$.

$$\begin{aligned} p(\mathbf{w}|X, \mathbf{y}) &= \frac{p(y|\mathbf{w})p(\mathbf{w}|X)}{p(X, \mathbf{y})} \\ &\propto \prod_{i=1}^{n} \Phi(y_i \mathbf{x}^\top \mathbf{w}) \exp(-\frac{\mathbf{w}^\top \Sigma_p^{-1} \mathbf{w}}{2}) \\ \log p(\mathbf{w}|\mathcal{D}) &\stackrel{c}{=} \sum_{i=1}^{n} \log \Phi(y_i \mathbf{x}^\top \mathbf{w}) - \frac{1}{2} \mathbf{w}^\top \Sigma_p \mathbf{w} \end{aligned} \tag{11.3}$$

where $\stackrel{c}{=}$ means "equality up to a constant". Since the likelihood is not Gaussian, the posterior is also not Gaussian. To make predictions for new inputs $\mathbf{x}_*$ we calculate the predictive distribution as the likelihood $p(y_*, \mathbf{w}|\mathcal{D}, \mathbf{x}_*)$ and integrate over $\mathbf{w}$.

$$p(y_*|\mathcal{D}, \mathbf{x}_*) = \int p(y_*|\mathbf{w}, \mathbf{x}_*)p(\mathbf{w}|\mathcal{D}, \mathbf{x}_*)\, d\mathbf{w} \tag{11.4}$$

which is also analytically intractable. Rasmussen and Williams [2006] note that even though the posterior is analytically intractable, it is concave and unimodal for some response functions (one of which is the logistic function that we are using here). Thus, we can approximate the posterior by maximizing Equation 11.3 with respect to $\mathbf{w}$; this is also known as the MAP-estimate. The predictive integral in Equation 11.4 turns into calculating Equation 11.2 for $\mathbf{x}_*$ and the MAP estimate of $\mathbf{w}$. The maximum of the posterior can be calculated numerically with Newton's method, or more sophisticated algorithms like conjugate gradient descend [Minka, 2003].

11.1.2 A Non-Parametric Example

We now turn to a non-parametric model of the classification problem. The idea of Gaussian Process classification is similar to that of regression: we relate the outputs (class labels) to a function $f(\mathbf{x})$, $p(y_i|f(\mathbf{x}))$, and use a Gaussian Process to draw different functions; then, we perform inference directly on the space of functions. In classification, we "squash" $f(\mathbf{x})$ through a sigmoid function to obtain a valid probability estimate. Assuming some observed training data $\mathcal{D} = (X, \mathbf{y})$, the likelihood is given by

$$p(\mathbf{y}|\mathbf{f}) = \prod_{i=1}^{n} p(y_i|f(\mathbf{x}_i)) = \prod_{i=1}^{n} \Phi(y_i f(\mathbf{x}_i))$$

where $\mathbf{f}$ is the function f evaluated at the training inputs. The function $f(\mathbf{x})$ plays the role of a *nuisance* function. We are not particularly interested in the values of $f(\mathbf{x})$, but rather in the values of $\Phi(f(\mathbf{x}))$, particular in the case of test inputs, $\Phi(f(\mathbf{x}_*))$. The purpose of f is simply to allow for a convenient formulation of the model. As in the regression case, we integrate over f to obtain the predictive likelihood of $p(y_*|\mathbf{x}_*, \mathcal{D})$.

We define a prior on $f(\mathbf{x})$ to be a Gaussian Process with a mean and covariance function as before

$$\begin{aligned} p(f(\mathbf{x})) &= \mathcal{GP}\big(m(\mathbf{x}),\ k(x, x')\big) \\ p(\mathbf{f}|X) &\sim \mathcal{N}\big(\boldsymbol{\mu}, K\big) \end{aligned}$$

then, we obtain the unnormalized posterior distribution by multiplying the likelihood with the prior

$$\begin{aligned} p(\mathbf{f}|\mathcal{D}) &\propto p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|X) \\ &\propto \prod_{i=1}^{n} \Phi(y_i f(\mathbf{x}_i))\mathcal{N}\big(\mathbf{f}|0, K\big) \end{aligned}$$

which is not Gaussian because of the non-Gaussian likelihood. Inference is performed in two steps: first we calculate the value for the latent function at test point $\mathbf{x}_*$, $f_* = f(\mathbf{x}_*)$, then, we integrate out f_* to produce a probabilistic prediction

$$\begin{aligned} p(f_*|\mathcal{D}, \mathbf{x}_*) &= \int p(f_*|X, \mathbf{x}_*, \mathbf{f})p(\mathbf{f}|\mathcal{D})\, d\mathbf{f} \\ p(y_*|\mathcal{D}, \mathbf{x}_*) &= \int p(y_*|f_*)p(f_*|\mathcal{D}, \mathbf{x}_*)\, df_* \end{aligned} \tag{11.5}$$

In the regression case, the predictive integral was analytically tractable since both the likelihood and posterior were Gaussian. In the classification case, none of this is true and we need to resort to analytical approximations of the integrals on the previous page or solutions based on Monte Carlo sampling. The latter approach combined with Markov Chains was explored by [Neal, 1999]. A variety of methods have been used to analytically approximate the non-Gaussian joint posterior of which the predictive integrals is a direct consequence of, *cf.* the introduction of this chapter.

Nickisch and Rasmussen [2008] performs an empirical study of analytical approximations to the integrals on the preceding page and concludes that the Expectation Propagation algorithm is almost always the method of choice unless the computation budget is very tight; we refer the interested reader to Minka [2002] for details of the Expectation Propagation algorithm. Both Markov Chain Monte Carlo sampling, Expectation Propagation and the other analytical approximations runs in $\mathcal{O}(n^3)$ [Rasmussen and Williams, 2006] where n is the number of observations (i.e. companies).

11.2 Covariance Functions

The covariance function is the vital ingredient to our Gaussian process as it specifies the space of functions considered for regression and classification. We already saw examples of two covariance functions, a linear covariance function and the Squared Exponential covariance function (*cf.* Section 10.5). As there are infinitely many functions, there are also infinitely many covariance functions. In this section we define the properties of a covariance function more formally, and inspect other popular covariance functions.

11.2.1 Definition and Properties

From integral operators theory, a function k of two arguments mapping into $\mathbb{R}$ is called a *kernel*. The operator T_k is defined as

$$(T_k f)(\mathbf{x}) = \int_X k(\mathbf{x}, \mathbf{x}') f(\mathbf{x}')\, d\mathbf{x}'$$

A kernel is said to be symmetric if $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$. Given a set of inputs $X = \{\mathbf{x}_i | i = 1, \ldots, n\}$, we can compute the Gram matrix K whose entries are $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.[1] If k is a covariance function, we call K a *covariance matrix*. From the definition, both K and k are symmetric if k is a covariance function. We use the terms "kernel" and "covariance function" interchangeably in this chapter unless specified otherwise.

We noted in the discussion of the non-parametric model, Section 10.4 on page 73, that the covariance matrix is positive semidefinite; we defined this informally as k being non-negative for all arguments $\mathbf{x}_i$ and $\mathbf{x}_j$. Formally, K is positive semidefinite if and only if $\mathbf{v}^\top K \mathbf{v} \geq 0$ for all vectors $\mathbf{v} \in \mathbb{R}^D$.

A covariance function k is said to be *stationary* if $K(x_i, x_j)$ is only a function of the difference $\mathbf{x}_i - \mathbf{x}_j$; stationary covariance functions are invariant to translations. If k is a function of $|\mathbf{x}_i - \mathbf{x}_j|$ it is *isotropic* and also invariant to rotations. The squared exponential is an example of a stationary isotropic covariance function.

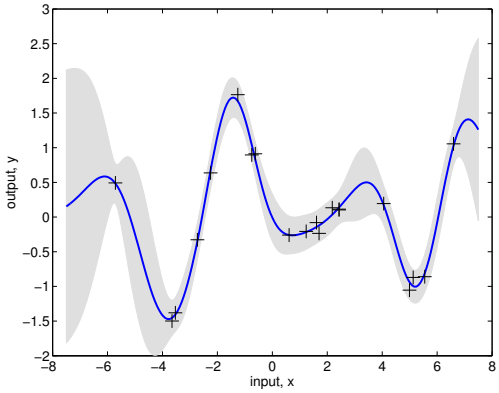
If the covariance function is not a function of the Euclidean distance between the arguments it is called a *non-stationary* covariance function. The dot-product kernel used to define a function over linear functions in Figure 10.5 (a) on page 77 is non-stationary. Non-stationary covariance functions are able to adapt to variable smoothness of functions which is not possible when only considering the stationary functions [Gibbs, 1997; MacKay, 1997]; whether variable smoothness is expected depends on the domain in question. Note that we can model any degree of smoothness

[1] A Gram matrix is a symmetric matrix of inner products. Our covariance matrix is an example of such a matrix.

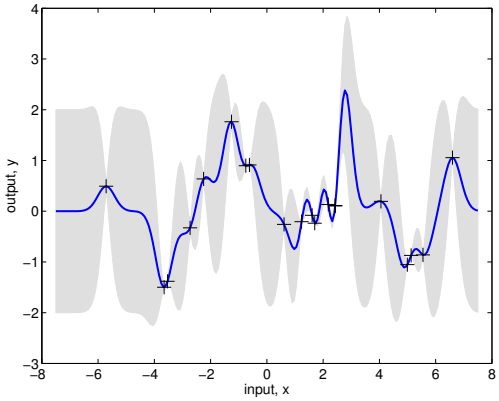
(including not smooth) with ordinary stationary kernels. Before we discuss stationary and non-stationary covariance functions, we take a look at some common hyperparameters of stationary covariance functions and how they affect the distribution over functions.

11.2.2 Varying Hyperparameters

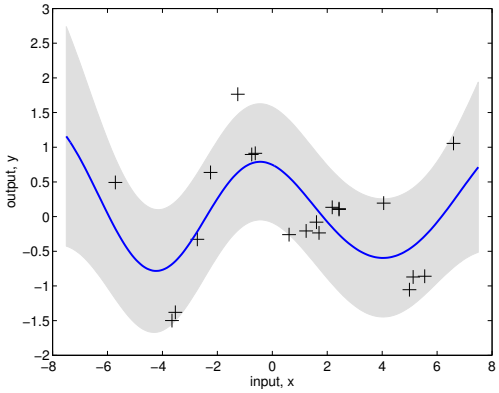
Stationary covariance functions are often characterized by a *length-scale* attribute ℓ. Informally, the length-scale specifies how far—in input space—you need to move along a particular axis for the function values to become uncorrelated. We use the Squared Exponential to illustrate the role of the length-scale parameter. Figure 11.2 illustrates how the hyperparameters affects the



(a) $\ell = 1$



(b) $\ell = 0.3$



(c) $\ell = 3$

Figure 11.2: Illustrations of different values for ℓ for the same data points. The grey area is the mean function plus minus two times the standard deviations, corresponding to a 95 % confidence interval for the underlying function f. The hyperparameters are $\theta = (\ell, \sigma_f, \sigma_n)$. (a) Data points generated by a Gaussian process is shown with crosses; the mean function is drawn with a solid line, $\theta = (1.0, 1.0, 0.1)$ (b) $\theta = (0.3, 1.0, 0.0001)$. (c) $\theta = (3.0, 1.3, 0.4)$.

distribution over functions conditioned on some data points. Figure 11.2 (a) shows the data points drawn from a GP with Squared Exponential covariance function with hyperparameters $\theta = (1.0, 1.0, 0.1)$. As the distance between observations increases, so does the uncertainty of the function values. Because of the noise component, we do not expect the observations to be functional, i.e. $\mathbf{x}_i$ and $\mathbf{x}_j$ may yield different y even though $\mathbf{x}_i = \mathbf{x}_j$. Observe that the data points are fitted fairly well in (a), the function explains the discrepancies from the observations near $x = 2$ as noise, the same goes for the observations near $x = 5$.

If we lower the length scale, we allow the function to vary more rapidly, as seen in Figure 11.2 (b). Notice that the error bars are now much higher in the areas with no observations. Since the function is allowed to move more freely, the noise component is much lower than in (a). Notice that the observations near $x = 2$ is now explained by the function and not by the noise. In a sense, the function in (b) is "better" than the function in (a) as it fits all the observations perfectly, however the uncertainties in (b) makes the function less attractive; not surprisingly, the function in (b) is not favoured by the marginal likelihood. The marginal likelihood is a measure of well a model fits the data when complexity is taken into account, more on this in Section 11.3.

Figure (c) illustrates what happens when we set the length scale too high and optimize the other parameters. Now, almost all the data points are explained by noise and the error bars are quite high over the entire interval. The amplitude of the function, σ_f, has also been raised a bit. We can of course take this to the extremes. If we let $\ell \to \infty$ then the function would be constant with added white noise. Conversely, if $\ell \to 0$, the function would model white noise.

Given only the data points in Figure 11.2 (a), we are able to deduce that white noise is not a suitable model for the data as the observations seems correlated. However, if the inputs is of higher dimensions, such correlation is no longer obvious. We inspect how we can use the marginal likelihood to score various models of the hyperparameters in Section 11.3, now we take a look at some commonly used covariance functions.

11.2.3 Stationary Covariance Functions

The Squared Exponential

We have already seen several examples of the Squared Exponential (SE) covariance function. Using the Squared Exponential covariance function is the same as performing regression with $f(x) = \int_{-\infty}^{\infty} \gamma(u) \exp(-(x-u)^2)\, du$. In other words, we place a squared exponential at every point on on the x-axis (equals at every point in the input-space in higher dimensional spaces), and sum all the components, *cf.* page 76. This is the same as using a Gaussian Process to model a space of functions with Equation 11.6 as the covariance function.

$$k_{SE}(r) = \sigma_f \exp\big(-\frac{|r|}{2\ell}\big) \tag{11.6}$$

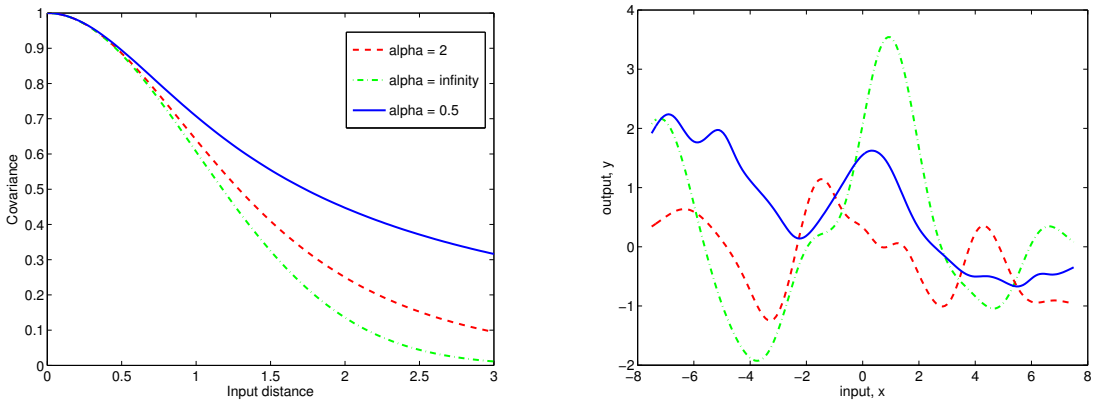
where $r = \mathbf{x} - \mathbf{x}'$ and $|\mathbf{z}|$ is Euclidean length of the vector $\mathbf{z}$. The Squared Exponential is infinitely differentiable and is thus very smooth. Also, the length scale is constant for the entire function; we do not accept the functions drawn from this distribution to have varying length scales. The Squared Exponential is usually critiqued for the smoothness assumptions, Stein [1999] argues that such strong smoothness is unrealistic for many physical problems. Nevertheless, the Squared Exponential remains the most popular covariance function for solving practical problems.

The Rational Quadratic

The Rational Quadratic covariance function is a generalization of the Squared Exponential with variable length-scale ℓ. We let $\tau = \ell^{-2}$ and use a Gamma distribution to model τ, $p(\tau|\alpha,\beta) \propto \tau^{\alpha-1} \exp\big(\alpha\tau/\beta\big)$, and get

$$\begin{aligned} k_{RQ}(r) &= \int p(\tau|\alpha,\beta) k_{SE}(r|\tau)\, d\tau \\ &= \int \tau^{\alpha-1} \exp\big(-\frac{\alpha\tau}{\beta}\big) \exp\big(\frac{\tau r^2}{2}\big)\, d\tau \\ &\propto \Big(1 + \frac{r^2}{2\alpha\ell^2}\Big)^{-\alpha} \end{aligned} \tag{11.7}$$

Figure 11.3 on the facing page illustrates the behaviour of the Rational Quadratic kernel for various α. Just as the Squared Exponential, the Ration Quadratic is also infinitely differentiable.



(a) (b)

Figure 11.3: The Rational Quadratic function (a) The covariance as a function of Euclidean distance for various α. (b) Random samples drawn from Gaussian Processes with Rational Quadratic covariance functions. $\ell = 1$ for all cases.

In the limit of the RQ covariance, $\alpha \to \infty$, the gamma distribution converges to one and the integral in Equation 11.7 becomes the SE covariance function with length scale ℓ. The Rational Quadratic is the most general representation for an isotropic kernel which defines a valid covariance function in any dimension [Stein, 1999].

The Matérn Class of Covariance Functions

The Matérn class of covariance functions is—in addition to the length scale—parameterized by a parameter that controls how many times sample functions are mean square differentiable; this allows the user to perform inference on how smooth functions are. The Matérn class of covariance functions is given by

$$k_{Matern}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)}\Big(\frac{\sqrt{2\nu}r}{\ell}\Big)^{\nu} K_{\nu}\Big(\frac{\sqrt{2\nu}r}{\ell}\Big)$$

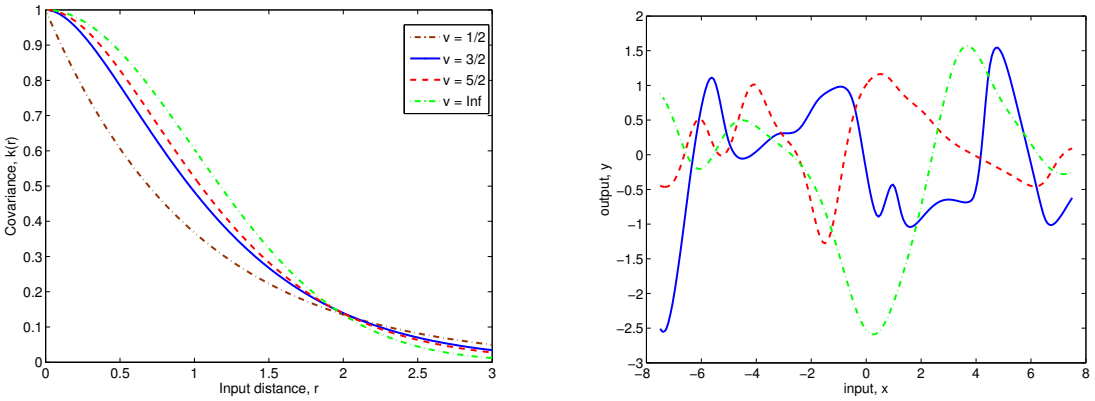
where ν denotes how many times function samples are mean square differentiable and ℓ denotes the length-scale; Γ is the Gamma function and K_{ν} is a modified Bessel function [Abramowitz and Stegun, 1965]. If ν is is half-integer, $\nu = p + 1/2$, $p \in \mathbb{Z}$, then the covariance function becomes a product of an exponential and a polynomial of order p. Rasmussen and Williams [2006] argues that the most interesting values for p in the context of machine learning is 1 and 2, resulting in $\nu = 3/2$ and $\nu = 5/2$ respectively. For $p = 0$ the sample functions become very rough since they are not differentiable; $p \geq 3$ requires prior knowledge about higher order derivatives which is probably hard to determine from noisy training examples. The Matérn covariance function for $\nu = 3/2$ and $\nu = 5/2$ is given below.

$$\begin{aligned} k_{\nu=3/2}(r) &= \Big(1 + \frac{\sqrt{3}r}{\ell}\Big)\exp\Big(-\frac{\sqrt{3}r}{\ell}\Big) \\ k_{\nu=5/2}(r) &= \Big(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\Big)\exp\Big(-\frac{\sqrt{5}r}{\ell}\Big) \end{aligned} \tag{11.8}$$

Figure 11.4 on the following page illustrates the covariance and samples from the Matérn class covariance functions. When $\nu \to \infty$, the Matérn covariance function becomes the Squared Exponential with length-scale ℓ^2.

11.2.4 Non-Stationary Covariance Functions

All the covariance functions in the previous section was a function of the distance between two input vectors $r = \mathbf{x}_i - \mathbf{x}_j$. We have already seen an example of one covariance function



(a) (b)

Figure 11.4: (a) The covariance as a function of input distance for Matérn covariance functions with different ν. (b) Random samples drawn from a Gaussian Process with Matérn covariance functions, $\ell = 1$. The solid blue line only has one derivative which is not easily seen in the figure.

which is not stationary: the linear dot product kernel in Section 10.5 on page 75. This kernel is not particularly useful as it can only represent linear functions. However, there are other non-stationary kernels which are more interesting, one of them is the Neural Network kernel.

Neural Network Kernel

It has been shown that neural networks with one hidden layer are universal approximators—meaning they can approximate any function—as the number of hidden units tends to infinity for a wide class of transfer functions [Hornik, 1993]. Consider a network which takes as input a vector $\mathbf{x}_i$, has one hidden layer with N_H neurons weighting the inputs with weights $\mathbf{u}$, and a single output neuron weighting the hidden layer neurons and a bias with weights $\mathbf{v}$. The construction is illustrated in Figure 11.5 with four hidden units; the bias neural is always one. $h(\mathbf{x}; \mathbf{u}_j)$ is the hidden unit transfer function which is usually chosen to be a sigmoid function, e.g. the logistic function. However, strictly speaking, the only requirement is that transfer function is bounded.

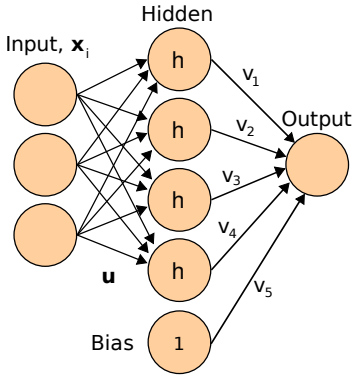


Figure 11.5: An Artificial Neural Network. Inputs are weighted by weights $\mathbf{u}$, $h(\mathbf{x}; u_j)$ is the transfer function for each hidden neuron. The output weights the hidden layer neurons and a bias neuron with weights $\mathbf{v}$.

The mapping from inputs $\mathbf{x}_i$ to outputs $f(\mathbf{x}_i)$ in the neural network can be written as

$$f(\mathbf{x}_i) = b + \sum_{j=1}^{N_H} v_j h(\mathbf{x}_i; \mathbf{u}_j)$$

Neal [1996] demonstrated how to construct a kernel approximating a neural network with unlimited hidden neurons. Following Neal [1999], we calculate the mean and covariance functions in the same fashion as we did with the linear and squared exponential on page 76. Assuming independent zero mean Normal distributions on b, $\mathbf{u}$ and $\mathbf{v}$ with variance σ_b^2, σ_u^2 and σ_v^2 respectively, we get (denoting $\mathbf{w}$ as all the weights):

$$m(x) = \mathbb{E}_{\mathbf{w}}[f(x)] = 0$$

$$\begin{aligned} \mathrm{Cov}[f(\mathbf{x}), f(\mathbf{x}')] &= \mathbb{E}[\big(f(\mathbf{x}) - 0\big)\big(f(\mathbf{x}') - 0\big)] \\ &= \sigma_b^2 + \sum_j \sigma_v^2 \mathbb{E}_{\mathbf{u}}[h(\mathbf{x}; \mathbf{u}_j) h(\mathbf{x}'; \mathbf{u}_j)] \\ &= \sigma_b^2 + N_H \sigma_v^2 \mathbb{E}_{\mathbf{u}}[h(\mathbf{x}; \mathbf{u}) h(\mathbf{x}'; \mathbf{u})] \end{aligned} \tag{11.9}$$

the sum in Equation 11.9 turns into N_H since h is identically distributed for all the hidden neurons. The covariance function is found by evaluating $\mathbb{E}_{\mathbf{u}}[h(\mathbf{x}; \mathbf{u}) h(\mathbf{x}'; \mathbf{u})]$ which depends on the choice of transfer function. Williams and Barber [1998b] use the error function $\mathrm{erf}(z) = 2/\sqrt{\pi} \int_0^z \exp(-t^2)\, dt$ as the transfer function with $h(\mathbf{x}; \mathbf{u}) = \mathrm{erf}\big(u_0 + \sum_{j=1}^{D} u_j x_j\big) = \mathrm{erf}(u_0 + \mathbf{u}\mathbf{x})$, and chose $\mathbf{u} \sim \mathcal{N}(0, \Sigma)$. This transfer function is a sigmoid function, just as the logistic function; the covariance function for this transfer function is

$$k_{NN}(\mathbf{x}, \mathbf{x}') = \frac{2}{\pi} \sin^{-1}\left(\frac{2\tilde{\mathbf{x}}^\top \Sigma \tilde{\mathbf{x}}'}{\sqrt{(1 + 2\tilde{\mathbf{x}}^\top \Sigma \tilde{\mathbf{x}})(1 + 2\tilde{\mathbf{x}}'^\top \Sigma \tilde{\mathbf{x}}')}} \right) \tag{11.10}$$

where $\tilde{\mathbf{x}} = (1, x_1, \ldots, x_n)^\top$ is the input vector augmented with the bias node; $\Sigma = \mathrm{diag}(\sigma_0^2, \sigma^2)$ and controls the variance for u_0 and $\mathbf{u}$ respectively. Samples from Gaussian Processes with this covariance functions can be viewed as regression in a model with superpositions of h [Rasmussen and Williams, 2006], just as the Squared Exponential covariance function can be viewed as regressions with superpositions of squared exponentials (*cf.* Section 10.5 on page 75). Figure 11.6 illustrates some samples drawn from a Gaussian Process with this covariance function. Notice that samples displays the non-stationarity of the covariance function in that the samples tend to constant for large values of $-x$ and x: consistent with the model of superpositioned sigmoid functions. For more non-stationary covariance functions, see [Abrahamsen et al., 1997; Rasmussen and Williams, 2006; Schölkopf and Smola, 2002].

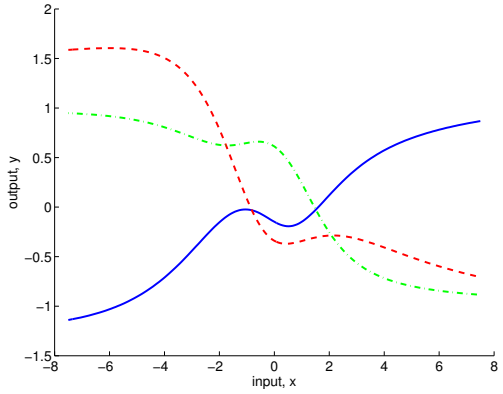


Figure 11.6: Samples drawn from a Gaussian Process with a Neural Network Kernel.

11.3 Model Selection

Having defined the Gaussian Process formalism and the use of covariance functions, the only questions left are what covariance function should be used, and what is the configuration of the hyperparameters of that function (if any). These two questions forms the learning aspect of Gaussian Processes and we use the term "model selection" to cover both determination of the covariance function and selection of the corresponding hyperparameters.

We saw several examples of covariance functions in the previous section and the variety of functions which can be represented by a single covariance function by varying the free hyperparameters. The selection of covariance function and hyperparameters not only helps us refine the predictions of the model, but also gives valuable insight in the properties of the data. For example, the characteristic length scale parameter ℓ in stationary kernels tells us the correlation assumption made by the model: if the length scale has a very large value, then the covariance becomes almost independent of the inputs.

Our task is, based on a set of training data, to make inference about the form and parameters of the covariance function, or equivalently, about the relationships in the data. We need to be able to compare different models and select the optimal model. Also we need to compare Gaussian Process models with any other model such as neural networks or fuzzy logic models. Although there are endless variations in suggestions for model selection in the literature, Rasmussen and Williams [2006] argue that three general principles cover most: (1) compute the probability of the model given the data, (2) estimate the *generalization error* and (3) bound the generalization error. The generalization error is the error calculated from the classification of the validation set, assumed to be from the same distribution as the training set. The training error is usually a bad estimate of the error since there is a possibility of overfitting the training data (model the noise in the data). We already discussed how to estimate the generalization error with the validation set and cross validation, Section 3.1 and 6.4.

The *marginal likelihood* in the Bayesian framework provides a probability estimate of the model given the data, and we use this to find the optimal model for our classification problem. We start our discussion of model selection by introducing a parametric example in the plain Bayesian framework, and then extend the parametric example to non-parametric model with Gaussian Processes.

11.3.1 Model Selection: A Parametric Example

We usually view model selection as a hierarchical specification of parameters. At the bottom level we have the parameters $\mathbf{w}$ for the model; be it a linear model as in Section 9.3 or the weights in a neural network model. At the second level are hyperparameters θ which controls the distribution of $\mathbf{w}$, then, at the top level, we may have set of possible models structures M_i under consideration; inference takes place one level at a time. At the bottom level, the *posterior* for the parameters $\mathbf{w}$ is given by Bayes' rule

$$p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \theta, M_i) = \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w}, M_i)p(\mathbf{w}|\theta, M_i)}{p(\mathbf{y}|\mathbf{X}, \theta, M_i)} \tag{11.11}$$

where $p(\mathbf{y}|\mathbf{X}, \mathbf{w}, M_i)$ is the *likelihood* and $p(\mathbf{w}|\theta, M_i)$ is the parameter prior. The prior distribution reflects our initial beliefs about the data and will affect the inference. Care should be taken not to employ too restrictive priors as this may rule out reasonable explanations of the data.[1] If we have vague prior knowledge, then we usually choose a broad prior distribution to reflect this. The denominator in Equation 11.11 is called the *marginal likelihood*, or evidence; it is the probability of the observations given the model and is independent of the parameters $\mathbf{w}$. The

[1] This is known as Cromwell's dictum after Oliver Cromwell who on August 5th, 1650 wrote to the synod of the Church of Scotland: "I beseech you, in the bowels of Christ, consider it possible that you are mistaken."

marginal likelihood is calculated by

$$p(\mathbf{y}|\mathbf{X}, \theta, M_i) = \int p(\mathbf{y}|\mathbf{X}, \mathbf{w}, M_i)p(\mathbf{w}|\theta, M_i)\, d\mathbf{w}$$

Moving up the ladder we can express the posterior for the hyperparameters θ in the same fashion:

$$p(\theta|\mathbf{y}, \mathbf{X}, M_i) = \frac{p(\mathbf{y}|\mathbf{X}, \theta, M_i)p(\theta|M_i)}{p(\mathbf{y}|\mathbf{X}, M_i)} \tag{11.12}$$

$p(\theta|M_i)$ is the prior for the hyperparameters and the marginal likelihood is given by

$$p(\mathbf{y}|\mathbf{X}, M_i) = \int p(\mathbf{y}|\mathbf{X}, \theta, M_i)p(\theta|M_i)\, d\theta$$

at the top level we compute the posterior for the model as

$$p(M_i|\mathbf{y}, \mathbf{X}) = \frac{p(\mathbf{y}|\mathbf{X}, M_i)p(M_i)}{p(\mathbf{y}|\mathbf{X})}$$

and the marginal likelihood for the model as

$$p(\mathbf{y}|\mathbf{X}) = \sum_i p(\mathbf{y}|\mathbf{X}, M_i)p(M_i)$$

assuming a discrete set of models M_i. In practice, the prior for the models $p(M_i)$ is usually chosen to be flat: all models are deemed equally probable. We note that implementation of Bayesian inference requires that we solve integrals which may be intractable, calling for analytical approximations or solutions based on Monte Carlo sampling. Rasmussen and Williams [2006] notes that the posterior distribution for the hyperparameters θ, Equation 11.12, are usually hard to calculate and that maximum likelihood approximations works well in practice as the distribution of the hyperparameters—in contrast to the model parameters $\mathbf{w}$—is usually unimodal [see MacKay, 1999, for a discussion].

Given a set of models and a set of datasets, there is a trade-off between complexity and how well the model explains the data. A simple model—say, a linear model—will explain few datasets well and the marginal likelihood will be low for most datasets; the few datasets it *does* explain will however receive very high marginal likelihood since the probability distribution needs to integrate to one. On the other hand, we have complex models—say, neural networks—that explains a lot of datasets, but will not be rewarded well for any particular because of the exact same reason. The *optimal* model is something in-between that describes enough datasets to be useful, and few enough to explain those datasets well (in other words, receives high marginal likelihood). This scenario is illustrated in Figure 11.7 on the following page and is in the machine learning literature referred to as "Occam's Razor" after William of Occam, 1285–1349, whose principle "of equally complex models, choose the simplest one".[1] Notice that Occam's Razor is automatic in a Bayesian framework, there is no parameter controlling the trade-off between complexity and data-fit.

11.3.2 A Non-Parametric Extension

In the non-parametric model we have no parameters $\mathbf{w}$ and it may not be immediately obvious what the parameters for the model actually are. Generally, we consider the latent noise-free function $f(x)$ evaluated at the training inputs, $\mathbf{f}$, as the parameters for the non-parametric model. Thus, the more training inputs, the more parameters. As in the parametric example, we integrate out $\mathbf{f}$ to obtain the marginal likelihood

$$p(\mathbf{y}|\mathbf{X}, \theta) = \int p(\mathbf{y}|\mathbf{X}, \theta, \mathbf{f})p(\mathbf{f}|\theta)\, d\mathbf{f}$$

[1] Latin: Pluralitas non est ponenda sine necessitate, or directly translated, "Plurality should not be assumed without necessity".

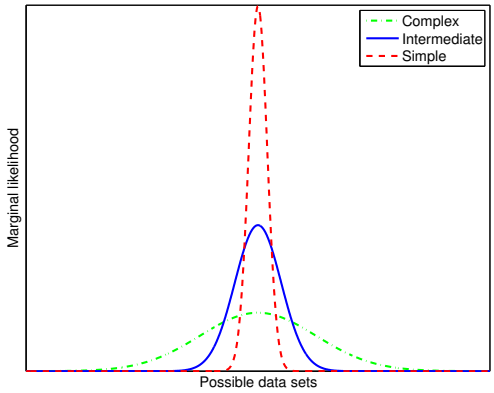


Figure 11.7: Occam's Razor. The solid blue line denotes the optimal model according to the marginal likelihood.

In the Gaussian Processes model the prior and likelihood are Gaussian, thus the marginal likelihood is also Gaussian. Written in the log domain, we can interpret the distribution a bit easier.

$$\log p(\mathbf{y}|X,\theta) = -\frac{1}{2}\mathbf{y}^\top K^{-1}\mathbf{y} - \frac{1}{2}\log|K| - \frac{n}{2}\log(2\pi) \tag{11.13}$$

the model along with the inputs are incorporated in K. The first term is the only term that depends on the data—the *data fit term*—it describes how well the model fits the data. The second term is the complexity penalty depending only on the covariance function and the inputs; the third term is a normalization constant ensuring that the range of the marginal likelihood is in $[0, 1]$. Figure 11.8 illustrates three different explanations for the observations, marked with crosses. The dotted red line illustrates a complex model overfitting the data; the dash-dotted green line illustrates a simple model not explaining the data very well. The solid blue line is the model favored by the marginal likelihood and balances well between complexity and data-fit.

The Gaussian Process model provides a framework for choosing the optimal model M_i (consisting of a covariance function and hyperparameters) rather than a way to perform inference on the space of all possible covariance functions. When using Gaussian Processes in practice, one usually use the marginal likelihood as an indicator to how good the current covariance function is (a covariance function may be composed of several other covariance functions) and then tune the covariance function accordingly.

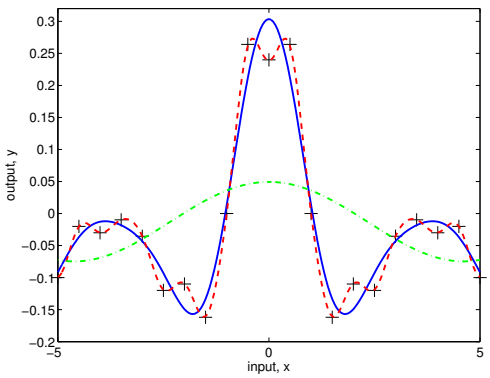


Figure 11.8: Gaussian Process regression with three different length-scales. Blue solid lines, $\ell = 1.0$. Red dotted line, $\ell = 0.3$. Green dash-dotted line, $\ell = 5$. σ_f and σ_n where set to 1.0 and 0.1 respectively in all cases. The optimal setting is close to $\ell = 1.0$.

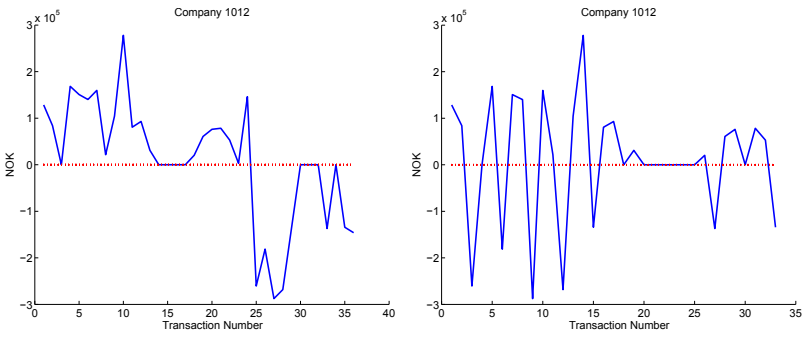
12 Results

The purpose of Part II of this thesis is to perform classification over the raw data: i.e. the withdrawal and deposit time series associated with each company. How to define the learning problem, particularly how to combine the time series into one series, is not obvious. Our suggested solution for this design issue is discussed in Section 12.1. We perform several experiments using binary GP classification utilizing the covariance functions covered in Section 11.2. The results from the experiments are presented in Section 12.2, where we additionally use the best performing covariance function to classify the companies in the validation set. We will use the term "Gaussian Process Classification" to mean Bayesian classification using a non-parametric model with a Gaussian Process prior.

12.1 Test Setup

In addition to the target values, y, our data set $\mathcal{D}$ consists of n companies, each described by two time series: withdrawals and deposits. Hence, we have a $n \times D \times 2$ large input data set, where D is the dimension of the time series. As with all common learning algorithms, classical Gaussian Process classification only handles data sets $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, n)\}$, where $\mathbf{x}$ denotes a *single* input vector of dimension D and y denotes a target. A way to combine the two time series into one vector $\mathbf{x}$ is needed.

The figures below illustrates two distinct options for how to combine the time series: (1) Concatenating the withdrawals with the deposits, (2) Interleave withdrawal transactions into the deposit time series: one withdrawal transaction is interleaved for every two deposits since a company only transfers money to the authorities every second month (*cf.* Section 2.1). Additionally, the options of using only the deposits, or only the withdrawals, were also considered. The drawback by only analyzing one of the time series is obvious: we neglect half of the information we have access to. Recall from Section 11.1, the dimensionality of the time series does not have an impact on the time complexity of the algorithm, which is $\mathcal{O}(n^3)$, where n is the number of companies. Hence, there is no obvious reason for using only half of the time series data.



(a) Concatenated (b) Interleaved

Figure 12.1: Two ways of representing the input. In (a) the withdrawals and deposits have been concatenated, in (b) the raw data input is created by interleaving the withdrawals and the deposits. Experiments showed that a binary Gaussian Process classifier performed better over the concatenated representation.

The best way to represent the raw data is chosen by running 10-fold cross-validation over the two different representations using a binary Gaussian Processes classification algorithm, as described in Section 11.1. According to such experiments, the concatenated representation resulted in a slightly higher log marginal likelihood. Hence, the concatenated representation is used in the resulting part of this thesis.

For clarification, we note that the same safety buffers used in Part I is also used in this part when deciding what segment of a time series to analyze. To recap, for financially sick companies the transactions occurring in the preceding months before a loss reckoning, or a loan defaulting, are analyzed. For financially healthy companies we analyze an equally long interval as analyzed for sick companies, ending six months before the last data point. We know that the segment analyzed is followed by six months where the company is still healthy, hence the name "safety buffer".

The results presented in the following section are acquired over a data set consisting of transactions, both withdrawals and deposits, occurring over an interval length of 24 months. If a company has not existed in all of the 24 months before it defaults on a loan or incur a loss, we pad the time series with zeros. This way we ensure consistency over the length of the time series.

First, we perform model selection. That is, we run experiments aimed at identifying the best performing covariance function. All of the covariance functions described in Section 11.2 are tested, and their corresponding hyperparameters are learned from the training data. The model that results in the highest log marginal likelihood is chosen as the one to use for further classification tasks.

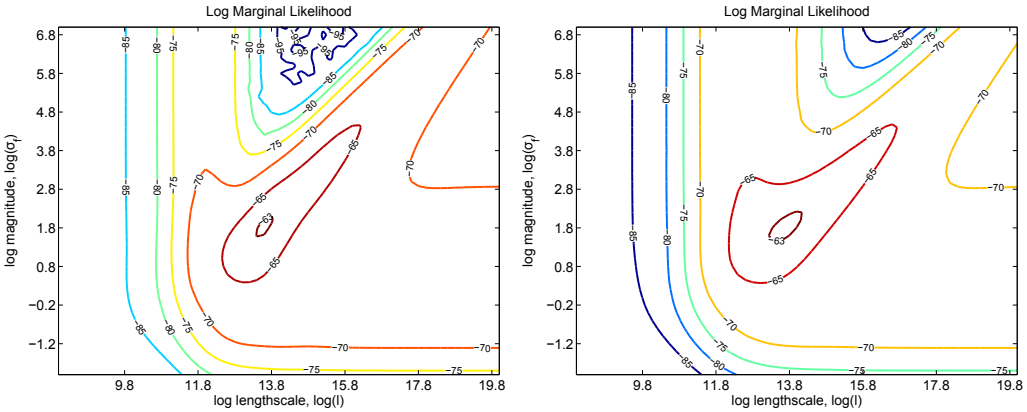
Secondly, we test the best performing model by classifying the companies in the validation set. As in Part I the classification algorithm is wrapped by MetaCost, described in Section 5.4 on page 40. This way we induce the notion of cost into the classification, and additionally, we handle the imbalanced data set. We use the same cost matrix as used in Part I, see Table 12.1.

	Predicted, i	
Actual, j	0	1
0	0	1
1	10	0

Table 12.1: Cost Matrix

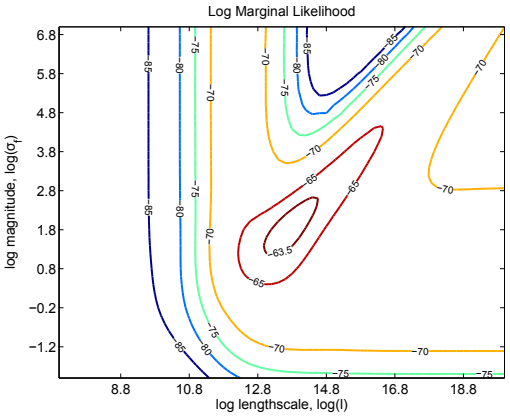
12.2 Empirical Results

First, we present the results from performing model selection. For each covariance function described in Section 11.2, we find the optimal hyperparameters. Then, the covariance function that best describes the data, indicated by the highest log marginal likelihood, is chosen as the model. In Figure 12.2 on the next page the log marginal likelihood is plotted as a function of the hyperparameters for various covariance functions.

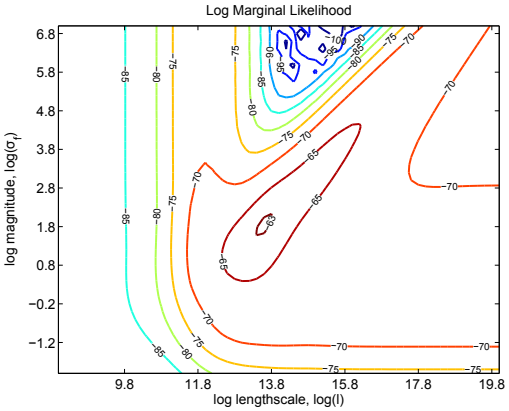


(a) Squared Exponential

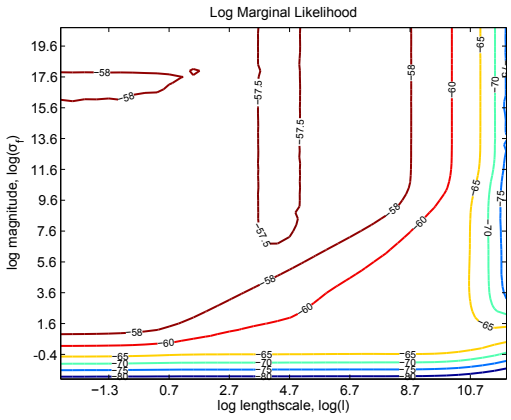
(b) Matérn, $\nu = 3/2$



(c) Matérn, $\nu = 5/2$



(d) Rational Quadratic, $\alpha = 4.6050$



(e) Neural Network

Figure 12.2: Model selection over the training data. The figure shows contour plots of the log marginal likelihood as a function of $\log(\ell)$ and $\log(\sigma_f)$, using a binary Gaussian Process classifier with various covariance functions. In Table 12.2 on the next page we list the optimal hyperparameters together with the optimal log marginal likelihood.

Covariance Function	*Max Log Marginal Likelihood*	*Hyperparameters*
Rational Quadratic	-62.8790	$\log(\ell) = 13.60$, $\log(\sigma_f) = 1.85$, $\alpha = 4.6050$
Squared Exponential	-62.8768	$\log(\ell) = 13.60$, $\log(\sigma_f) = 1.85$
Matérn, $\nu = \frac{5}{2}$	-62.9616	$\log(\ell) = 13.71$, $\log(\sigma_f) = 1.83$
Matérn, $\nu = \frac{3}{2}$	-62.8198	$\log(\ell) = 13.67$, $\log(\sigma_f) = 1.80$
Neural Network	-57.4885	$\log(\ell) = 4.30$, $\log(\sigma_f) = 10.0$

Table 12.2: A summary of the covariance functions along with the optimal hyperparameters and marginal likelihood. The optimal hyperparameters are learned by maximizing the log marginal likelihood as a function over the hyperparameters.

Table 12.2 shows a detailed summary of Figure 12.2 on the preceding page. The Neural Network covariance function excels the other covariance function with a log marginal likelihood of -57.4885. The rest of the covariance functions fit the training data similarly well, with a log marginal likelihood just higher than -63. Thus, the Neural Network covariance function with the corresponding hyperparameters listed in Table 12.2 will be used as a model for further classification over the companies in the validation set.

Next, the selected model, consisting of Neural Network covariance function with hyperparameters $\log(\ell) = 4.30$ and $\log(\sigma_f) = 10.0$, is tested on the validation set. Given a company from the validation set, the classifier predicts a probability that the company is financially sick, see Equation 11.5 on page 81. A company is classified as financially sick if the predicted probability is higher than 0.5. Below follows tables and figures describing the results from the validation set and training set, using cross validation on the latter. The final results after combining the approaches in Part I and Part II is presented in the next section. The results are commented in Chapter 13.

		Predicted Value	
		N	P
Actual Value	N	17	9
	P	0	5

(a) Validation Set

		Predicted Value	
		N	P
Actual Value	N	67	37
	P	6	20

(b) Cross validation

Table 12.3: Confusion matrices for the validation set and training set (using cross validation). A neural network covariance function was used with logarithmic hyperparameters: $\log(\ell) = 4.30$ and $\log(\sigma_f) = 10.0$

Results	
E(cost)	0.290
ROC	0.884
Accuracy	0.709

(a) Validation set

Results	
E(cost)	0.746
ROC	0.733
Accuracy	0.669

(b) Cross validation

Table 12.4: Performance measures for the GP-classifier on the validation- and training set.

Detailed Performance By Class

Class	*TP Rate*	*FP Rate*	*FN Rate*	*Precision*	*Recall*
0	0.6538	0.0	0.3462	1.0	0.6538
1	1.0	0.3462	0.0	0.3571	1.0

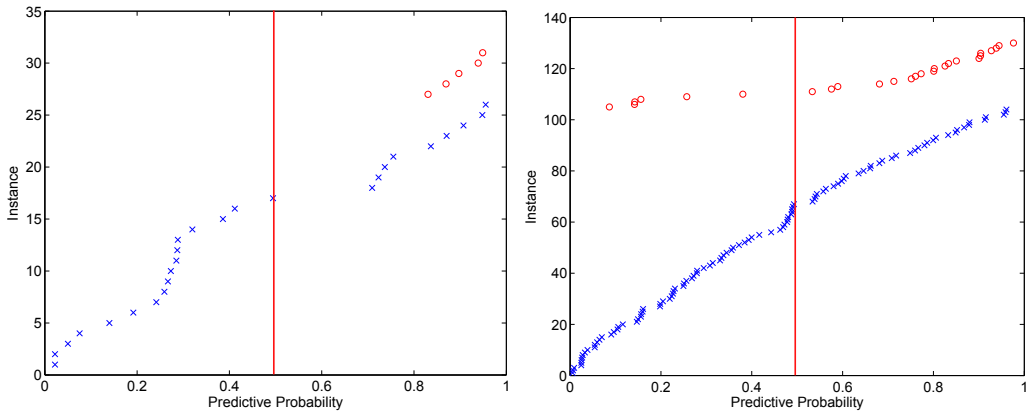
(a) Validation set

Detailed Performance By Class

Class	*TP Rate*	*FP Rate*	*FN Rate*	*Precision*	*Recall*
0	0.6442	0.2308	0.3558	0.9178	0.6442
1	0.7692	0.3558	0.2308	0.3509	0.7692

(b) Cross validation

Table 12.5: Detailed performance measures for the GP-classifier.



(a) Validation set (b) Cross validation

Figure 12.3: Confidence plots for the binary GP-classifier for the validation set and training set using cross validation. Confidences assigned to unhealthy companies are marked with red circles, confidences assigned to healthy companies are marked with blue crosses.

12.3 Combining the Results with the Naive Bayes Results

In Part I of this thesis, we performed a coarse-grained separation of the companies by a cost-sensitive Naive Bayes classifier using a set of features. The result from the coarse-grained separation were a grey area consisting of companies believed to be financially unhealthy. In this part, a cost-sensitive binary Gaussian Processes classifier takes a second look at the—believed to be—financially sick companies, analyzing the raw time series. The objective is to decrease the number of false positives (healthy companies classified as sick), tentatively without increasing the number of false negatives (sick companies classified as healthy).

In Table 12.6 (a) on the following page, we include an updated version of the confusion matrix presented in Table 6.1 on page 46, where we combine the result from Naive Bayes with the Gaussian Processes result. The combined confusion matrix is derived the following way:

$$
\begin{aligned}
TN_{comb} &= TN_{nb} + TN_{gp} \\
FP_{comb} &= FP_{nb} - TN_{gp} \\
TP_{comb} &= TP_{nb} - FN_{gp} \\
FN_{comb} &= FN_{nb} + FN_{gp}
\end{aligned}
$$

		Predicted Value	
		N	P
Actual Value	N	264	9
	P	2	5

(a) Combined, Validation

		Predicted Value	
		N	P
Actual Value	N	836	37
	P	11	20

(b) Combined, Cross-Val.

		Predicted Value	
		N	P
Actual Value	N	242	31
	P	2	5

(c) PD-Rating,Validation

		Predicted Value	
		N	P
Actual Value	N	752	121
	P	9	22

(d) PD-Rating, Cross-Val.

		Predicted Value	
		N	P
Actual Value	N	247	26
	P	2	5

(e) Naive Bayes, Validation

		Predicted Value	
		N	P
Actual Value	N	769	104
	P	5	26

(f) Naive Bayes, Cross-Val.

Table 12.6: (a)–(b) shows confusion matrices derived from integrating the results from the cost-sensitive Naive Bayes classifier with the results from the cost-sensitive classifier based on Gaussian Processes. (a) shows the result from classifying companies in the validation set, while (b) shows the result from performing cross-validation over the training set. In (c)–(d), we include the PD-rating confusion matrices; (e)–(f) shows the Naive-Bayes results.

In Table 12.6 (b), we include for completeness a combined confusion matrix summarizing the results from performing cross-validation over the training set. The corresponding PD-rating confusion matrices have been excerpted from Chapter 6 to ease the efforts of comparing our results with the PD-rating's performance, see Table 12.6(c)–(d). At last, the Naive Bayes results have been excerpted from Part I to ease the comparison between Part I and Part II results, see Table 12.6(e)–(f).

To calculate the ROC for the combined result, the confidences of the Naive Bayes classifier in Part I have been updated with the confidences derived from the Gaussian Processes classifier in this Part II. In Table 12.7 the ROC, Expected Cost and Accuracy is listed for every confusion matrix in Table 12.6.

Experiment	*E(cost)*	*ROC*	*Accuracy*
Combined	0.104	0.888	0.961
PD-Rating	0.182	0.921	0.882
Naive Bayes	0.164	0.877	0.900

(a) Validation set

Experiment	*E(cost)*	*ROC*	*Accuracy*
Combined	0.163	0.917	0.947
PD-Rating	0.233	0.856	0.856
Naive Bayes	0.170	0.909	0.879

(b) Cross validation

Table 12.7: Table enumerating the performance for the different experiments listed in Table 12.6.

13 Discussion

13.1 Model Selection

Of the covariance function tested, the neural network covariance function excelled the others. Table 12.2 on page 94 shows that the optimal log marginal likelihood using the neural network covariance function is -57.4885. The second best performing covariance function, the Matérn with $\nu = \frac{3}{2}$, resulted in a log marginal likelihood at -62.8198. This means that the marginal likelihood has increased by a factor of $\exp(62.8198 - 57.4885) = 206.7$ by choosing the neural network covariance function, compared to the Matérn covariance function.

Interestingly, the neural network covariance function is the only non-stationary covariance function tested in this thesis. Non-stationary covariance function allow the model to adapt to functions whose smoothness varies with the inputs [Paciorek and Schervish, 2004]. Hence, more flexibility is allowed for the latent function $\mathbf{f}$.

The optimal length-scale hyperparameter for the stationary covariance functions are relatively long: all are of magnitudes about 13.6. A consequence of the long length-scales is that the covariance between two outputs $f(\mathbf{x}_i)$ and $f(\mathbf{x}_j)$ will be close to one (or $\sigma_f^2 + \delta_{ij}\sigma_n^2$, *cf.* Equation 10.20 on page 77) and not provide any new information about the underlying process f. Put differently: a latent function value $f(\mathbf{x})$ is affected by a large set of other function values and the new observations provide limited new information to the whole picture. Short length-scales, on the other hand, cause new test observations to be dominated by the class-labels of the nearest neighbors in input space. If the distance to the nearest observation is large, the function may take a wide set of different values resulting in a large confidence interval. This effect is illustrated for regression in Figure 11.2 (b) on page 83. Notice that the prediction of a point y_1 given $x_1 = -4$ is not impacted by the fact that the prediction of point $y_2 = 0$ given $x_2 = 4$. This is due to the highly fluctuating regression function, which fits each training point exactly. Contrary to the effect observed with short length-scales, long length scales implies that observations are deemed close to each other. A prediction of a unseen test company is dominated by the class-label of several neighbors, limiting the flexibility. Of course, the length-scale needs to be "just right" and is part of the learning problem. The length scale for the non-stationary covariance function is much smaller than that for the stationary ones. The non-stationary covariance function explains the observations with high variance σ_f^2 and relatively short length scale ℓ.

Analogously to a Gaussian Process classifier with a long length-scale is a k-nearest neighbor classifier with a high k. In this analogy the covariance function corresponds to the distance function, representing a measure of similarity between two observations. A high k means that the prediction for a new test example is impacted by many neighbors; thus limiting the flexibility, since the prediction is an average over many class-labels (or targets).

The neural network covariance function stands out from the other covariance functions due to two observations, see Table 12.2. First, the length-scale is much shorter. In fact, the optimal length-scale for the neural network covariance function is $\exp(13.6 - 4.3) = 10{,}938$ times shorter than the second shortest optimal length-scale. Secondly, the neural network covariance function also has a relatively high optimal magnitude of the latent function $\mathbf{f}$, $\log(\sigma_f) = 10.0$. By itself, an increased $\log(\sigma_f)$ leads to harder predictions (i.e. confidences closer to 0 and 1), but the associated variances will also increase. This increased uncertainty tends to soften the confidences, i.e. move them closer to 0.5.[Rasmussen and Williams, 2006, p.67].

In summary, the most flexible covariance function fitted the data set best, with a relatively short length-scale ℓ (compared to the other covariance functions) and a high σ_f. This may indicate that the data set is hard to separate, containing high entropy. The less flexible covariance

functions were not able to distinguish the companies from each other, thus treating them all as similar. Composite covariance functions was tested without interesting results.

13.2 Evaluation of Gaussian Processes Results

First we present the results from classifying the validation set. Recall from Chapter 3 on page 9, validation set results indicates our models ability to generalize, i.e. perform well over data not used for tweaking the model. Validation set results are usually presented as the main result, but in our case it ended up being too small to distinguish the performance between two models. Therefore, the cross-validation results will be discussed as well.

Table 12.3 on page 94 presents the confusion matrix derived from classifying the validation set companies. The result is in general pleasing. Without misclassifying any of the sick companies we manage to correctly classify 17 healthy companies. Recall, all companies classified in this part were classified as financially sick by the Naive Bayes classifier. The data set consists of companies that are not easily recognized as financially healthy; hence, the task of separating the healthy companies from the sick is not considered an easy task. In Figure 13.1, we present two companies that Naive Bayes misclassified as sick. Both companies have time series containing several zero-transactions, which is probably partly the reason why Naive Bayes classified them as sick. The Gaussian Processes classifier manage to correctly classify the company presented in Figure 13.1(a) and (b), with confidence $1 - p(y = 1|\mathcal{D}, \mathbf{x}_*) = 0.81$.

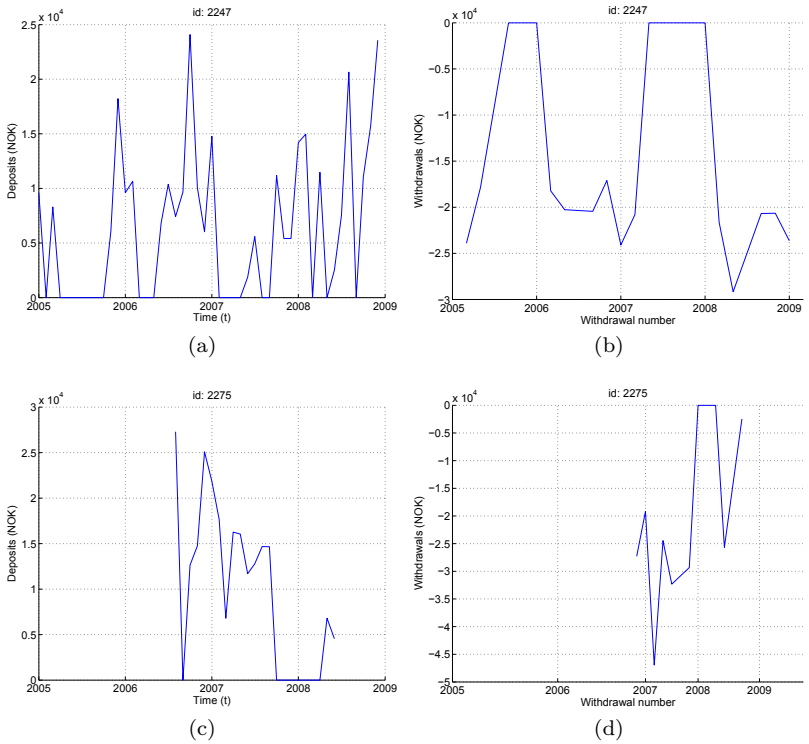


Figure 13.1: Examples of two companies; one correctly classified and one misclassified by the Gaussian Processes classifier. Company 2247 is correctly classified as healthy, with a confidence: $1 - p(y = 1|\mathcal{D}, \mathbf{x}_*) = 1 - 0.19 = 0.81$. (a) and (b) presents the deposits and the withdrawals, respectively. Company 2275 is misclassified as financially sick, with a confidence: $p(y = 1|\mathcal{D}, \mathbf{x}_*) = 0.96$. (c) and (d) presents the deposits and the withdrawals, respectively.

Company 2275, whose time series are presented in Figure 13.1(c) and (d), is unsurprisingly wrongly classified by the Gaussian Processes classifier. Considering what we learned from the study we performed in Section 5.1, we realize by visual inspection of the time series that the economical situation in this company is hardly healthy as the company several times does not transfer money to the authorities, and there is a strong negative trend in the withdrawals. Hence, even though it is a misclassification the confidence of $p(y = 1|\mathcal{D}, \mathbf{x}_*) = 0.96$ does not appear that far-fetched. In addition to the zero transactions, notice the steep negative trends.

The cross-validation results are not as good as the validation set results, but still decent. In Table 12.3 on page 94 we see that by misclassifying six actually unhealthy companies, we gain 67 more correctly classified healthy companies. The high number of false negatives makes the expected cost higher for the cross-validation results, compared to the validation results. Specifically, $E(cost) = 0.290$ for the validation set, compared to $E(cost) = 0.746$ when doing cross-validation. The same tendency is observed for the ROC and Accuracy metrics.

Another interesting tendency is observed in the confidence plots, see Figure 12.3. Notice the high number of companies that are classified with a confidence close to 0.5. This tendency is especially present in the cross-validation confidence plot, and indicates uncertainty in the classifier. A possible explanation may be the relatively high σ_f, as discussed in Section 13.1. Increasing the σ_f means increasing the magnitude of the values obtained for the latent function $\mathbf{f}$. Recall, a high σ_f increases the associated variances, hence soften the predictions (moving the confidences closer to 0.5). MetaCost will also impact the confidences, increasing them to higher probabilities.

The data set used in this part is different from the one used in Part I, being only a subset of the original data set. Hence, the results discussed above is not directly comparable to the results presented in Chapter 6. In next section, we discuss the combined results from the Gaussian Processes classifier and the Naive Bayes classifier from Part I. These results are directly comparable to both the PD-rating performance and the results in Part I.

In an effort to justify the use of a heavy statistical model such as Gaussian Processes, we tested the model from Part I (Naive Bayes over features) over the training data set used in this section. The result indicates a very low performance. The Naive Bayes classifier only managed to gain 26 correctly classified healthy companies, while at the same time misclassifying 6 unhealthy. This emphasizes the contribution done by the Gaussian Processes classifier. Further affirmation of this contribution is achieved when combining the contribution from the Gaussian Processes classifier with the Naive Bayes results, see next section.

13.3 Evaluation of Combined Results

Section 12.3 on page 95 presents the combined results from the Gaussian Processes classifier and the Naive Bayes classifier. The improvement done by the Gaussian Processes classifier is readily apparent by comparing the confusion matrices in Table 12.6, summarized in Table 12.7.

Prior to the Gaussian Process classification, all the companies considered were classified as unhealthy, this corresponds to an expected cost of 0.83 (26 false positives, 0 false negatives and a total of 31 companies, *cf.* Equation 6.1). For the validation set, the Gaussian Process classifier recognized 17 of these companies as healthy resulting in an expected cost of 0.29: a reduction of 0.54. For the training set, 67 companies were recognized as healthy, but this came at the cost of classifying 6 unhealthy companies also as healthy. Since we consider it worse to misclassify unhealthy companies as healthy, the resulting expected cost is 0.746: a reduction of only 0.08.

The ROC Area has increased when comparing the Naive Bayes results with the combined results. The explanation is evident, recall from the confidence plots in Chapter 6 that the Naive Bayes classifier tends to be overly confident (confidences close to 0 and 1), even when misclassifying healthy companies. When the Gaussian Processes classifier identifies several misclassified healthy companies, it updates the confidence to a probability lower than 0.5; hence the ROC area is increased. Similarly, the expected cost is decreased since the Gaussian Processes classifier generally decreases the number of misclassifications done by the Naive Bayes classifier. By comparing the

combined results with the PD-rating performance we notice that the narrow performance gap claimed in Part I has increased. The PD-rating still achieves a slightly higher ROC over the validation set, but performs less well on the other metrics.

At last, we will explore the fact that the Gaussian Processes classifier enhances the results from Part I. This suggest that the features, derived in Section 5.1, fails at catching all the relevant information from the time series. By analyzing the raw time series, the Gaussian Processes classifier learns relevant patterns that is lost in the abstractions made by the features. Of course, such patterns may only be apparent in an high dimensional space; thus not easy to discover through the kind of study we performed in Section 5.1.

14 Conclusion

We concluded Part I by claiming that our goal to beat the PD-rating had been accomplished. That is, we perform the task of recognizing financially unhealthy companies with a higher performance, compared to the PD-rating measured a month before a company defaults on a loan, or induces a loss-reckoning.

The results from Part I are satisfying, and are further enhanced by a binary Gaussian Processes classifier analyzing the raw data. Hence, the claim that we have accomplished our goal to beat the PD-rating has been further affirmed through Part II. A neural network covariance function with hyperparameters $\log(\ell) = 4.30$ and $\log(\sigma_f) = 10.0$ constitutes the selected model used for classifying the grey-area companies from Part I. For the validation set, this model results in 17 more companies correctly classified as healthy, without losing any correctly classified sick companies. By integrating the results from Part I and Part II we manage to correctly classify five out of seven financially sick companies, and 264 out of 273 financially healthy companies.

We have through Part I and II successfully derived and tested a two-fold model for classification of financially unhealthy companies. The model can be integrated into an early-warning system by training the cost-sensitive Naive Bayes classifier over the whole training set and the cost-sensitive Gaussian Processes classifier over the grey-area companies. After training, the model will immediately be able to estimate the probability that a given company is financially unhealthy. If the Naive Bayes classifier deems a company as sick, the Gaussian Processes will take a second look and return its confidence. On the other hand, if Naive Bayes classifies a company as healthy, a second opinion from the Gaussian Processes classifier is not needed.

The features from Section 5.1 does not catch all the relevant information in the time series. If they had, there would be no gain in running Gaussian Processes classification over the raw data. Hence, the main contribution of Part II is the recognition of new patterns in the raw data, indicating economical sickness.

As of future work, it would be interesting to identify the patterns only recognized by the Gaussian Processes classifier and incorporate them as Part I features. There are several ways to gain more knowledge about the underlying function: A more complete study of the hyperparameters will probably reveal more information. Additionally, there are several non-stationary covariance functions which we did not test, for instance, Gibbs [1997] proposed a covariance function where the characteristic length scale is controlled by a function, this covariance may perform better than the neural network covariance function. Rasmussen and Williams [2006] suggest testing covariance function implementing automatic relevance determination (ARD). ARD has been used successfully for removing irrelevant input, hence may be useful for acquire more knowledge about the data set. At last, we have not performed a test of how early in the time series a default, bankruptcy or other actions leading to loss is evident. There is no need to change the existing model to perform such test, however, this is left for future work.

A PD-Rating

This appendix contains specifics about the PD-rating.

A.1 Risk Class Mapping

In practice, SpareBank 1 operates with *risk classes* rather than the explicit PD-rating number. The mapping from the PD-rating to risk classes is illustrated in Table A.1. Risk class `J` is assigned automatically to companies that have defaulted on a loan; risk class `K` is assigned automatically to companies where a loss has been reckoned.

From	To	Risk Class
0.00	0.0010	A
0.0010	0.0025	B
0.0025	0.0050	C
0.0050	0.0075	D
0.0075	0.0125	E
0.0125	0.0250	F
0.0250	0.0500	G
0.050	0.100	H
0.1000	0.99	I
1	1	J
1	1	K

Table A.1: Numerical PD-rating to Risk Class mapping

B Mathematical Prerequisites

B.1 Gaussian Identities

The multivariate Gaussian distribution, $\mathcal{N}(\mathbf{x};\, \mathbf{m}, \Sigma)$, is given by

$$p(\mathbf{x}|\mathbf{m}, \Sigma) = \frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(x-\mathbf{m})^\top \Sigma^{-1}(x-\mathbf{m})) \tag{B.1}$$

where $\mathbf{m}$ is the mean vector and Σ is the (symmetric, semi-positive definite) covariance matrix. As a shorthand, we write $\mathbf{x} \sim \mathcal{N}(\mathbf{m}, \Sigma)$.

Joint Distribution

Let $\mathbf{x}$ and $\mathbf{y}$ be two Gaussian random vectors, then the joint distribution is also Gaussian and is given by

$$\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix} \sim \mathcal{N}\Big(\begin{bmatrix}\mathbf{m_x}\\ \mathbf{m_y}\end{bmatrix}, \begin{bmatrix}A & C\\ C^\top & B\end{bmatrix}\Big) \tag{B.2}$$

Marginal Distribution

The *marginal distribution* of $\mathbf{x}$ is $\mathbf{x} \sim \mathcal{N}(\mathbf{m_x}, A)$ and the marginal distribution of $\mathbf{y}$ is $\mathbf{y} \sim \mathcal{N}(\mathbf{m_y}, B)$. The covariance between $\mathbf{x}$ and $\mathbf{y}$ is given by C.

Conditional Distribution

The *conditional distribution* $p(\mathbf{x}|\mathbf{y})$ is

$$p(\mathbf{x}|\mathbf{y}) \sim \mathcal{N}\big(\mathbf{m_x} + CB^{-1}(\mathbf{y}-\mathbf{m_y}),\ A - CB^{-1}C^\top\big) \tag{B.3}$$

Products

The product of two Gaussians results in another (unnormalized) Gaussian

$$\begin{aligned}\mathcal{N}(\mathbf{x}|\mathbf{m_x}, A)\cdot\mathcal{N}(\mathbf{x}|\mathbf{m_y}, B) &= Z^{-1}\mathcal{N}(\mathbf{x}|\mathbf{m_c}, C)\\ \text{where } \mathbf{m_c} &= C(A^{-1}\mathbf{m_x} + B^{-1}\mathbf{m_y})\\ \text{and } C &= (A^{-1}+B^{-1})^{-1}\end{aligned} \tag{B.4}$$

The normalizing constant Z^{-1} looks itself like a Gaussian

$$Z^{-1} = (2\pi)^{-D/2}|A+B|^{-1/2}\exp\big(-\frac{1}{2}(\mathbf{m_x}-\mathbf{m_y})^\top(A+B)^{-1}(\mathbf{m_x}-\mathbf{m_y})\big) \tag{B.5}$$

B.2 Generating Samples from a Multivariate Gaussian Distribution

To generate samples $\mathbf{x} \sim \mathcal{N}\big(\mathbf{m_x}, K\big)$ with arbitrary mean and covariance matrix K, we proceed as follows

1. Compute the Cholesky decomposition, L, of K so that $LL^\top = K$, where L is a lower triangular matrix.

2. Generate a vector $\mathbf{u} \sim \mathcal{N}(0, I)$. Using a Gaussian random number generator. If you only have access to a uniform random number generator, then use the Box-Muller transform to obtain $\mathbf{u}$ from a set of uniformly distributed random numbers.

3. Compute $\mathbf{x} = \mathbf{m}_\mathbf{x} + L\mathbf{u}$ which has the desired distribution with mean $\mathbf{m}_\mathbf{x}$ and covariance matrix K.

In practice it may be necessary to add a small amount of noise to identity matrix I. This is because the eigenvalues of K may decay very rapidly. Without this stabilization, the Cholesky decomposition may fail.

Bibliography

Abrahamsen, P., Aji, S., McEliece, R., Altun, Y., Tsochantaridis, I., Hoffman, T., Amari, S., Anthony, M., Shawe-Taylor, J., Aronszajn, N., et al. (1997). A review of Gaussian Random Fields and Correlation Functions. *ANNS*, 26:2743–2760.

Abramowitz, M. and Stegun, I. (1965). *Handbook of mathematical functions with formulas, graphs, and mathematical table*. Courier Dover Publications.

Altman, E. I. (1973). Predicting Railroad Bankruptcies in America. *Bell Journal of Economics*, 4(1):184–211.

Atiya, A. (2001). Bankruptcy prediction for credit risk using neural networks: Asurvey and new results. *IEEE Transactions on neural networks*, 12(4):929–935.

Bernardo, J., Bayarri, M., Berger, J., Dawid, A., Heckerman, D., Smith, A., and West, M. (2003). The variational Bayesian EM algorithm for incomplete data: with application to scoring graphical model structures. In *Bayesian Statistics 7: Proceedings of the Seventh Valencia International Meeting*, page 453. Oxford University Press, USA.

Chatfield, C. (1996). *The Analysis of Time Series, 5th ed.* Chapman & Hall, New York, NY.

Deboeck, G. (1994). *Trading on the edge: neural, genetic, and fuzzy systems for chaotic financial markets*. Wiley.

Domingos, P. (1999). MetaCost: A General Method for Making Classifiers Cost-Sensitive. In *In Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 155–164. ACM Press.

Domingos, P. and Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine learning*, 29(2):103–130.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern recognition letters*, 27(8):861–874.

Fayyad, U. and Irani, K. (1993). Multi-interval discretization of continuous-valued attributes for classification learning.

G.E.P. Box, G. J. and Reinsel, G. (1994). *Time Series Analysis: Forecasting and Control (3rd Ed.)*. Prentice Hall, Englewood Cliffs, NJ.

Gibbs, M. (1997). Bayesian Gaussian Processes for Regression and Classification. *Unpublished doctoral dissertation, University of Cambridge*.

Gordon, M. and Rosenthal, J. (2003). Capitalism's growth imperative. *Cambridge Journal of Economics*, 27(1):25–48.

Gregorcic, G. and Lightbody, G. (2002). Gaussian processes for modelling of dynamic non-linear systems. In *Proceedings of the Irish Signals and Systems Conference, Cork*, pages 141–147.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182.

Hand, D. and Till, R. (2001). A simple generalisation of the area under the ROC curve for multiple class classification problems. *Machine Learning*, 45(2):171–186.

Hand, D. J. and Henley, W. E. (1997). Statistical Classification Methods in Consumer Credit Scoring: a Review. *Journal Of The Royal Statistical Society Series A*, 160(3):523–541.

Hodrick, R. and Prescott, E. (1997). Postwar US Business Cycles: An Empirical Investigation. *Journal of Money, Credit & Banking*, 29(1).

Hornik, K. (1993). Some new results on neural network approximation. *Neural Networks*, 6(9):1069–1072.

Japkowicz, N. (2002). The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429–449.

Japkowicz, N. et al. (2000). Learning from imbalanced data sets: a comparison of various strategies. In *AAAI workshop on learning from imbalanced data sets*, pages 00–05.

Karagiannopoulos, M., Anyfantis, D., Kotsiantis, S., and Pintelas, P. (2007). A Wrapper for Reweighting Training Instances for Handling Imbalanced Datasets. page 29.

Kotsiantis, S., Kanellopoulos, D., and Pintelas, P. (2006). Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, 30(1):25–36.

Kubat, M., Holte, R., and Matwin, S. (1998). Machine learning for the detection of oil spills in satellite radar images. *Machine Learning*, 30(2):195–215.

Ling, C., Huang, J., and Zhang, H. (2003). AUC: a statistically consistent and more discriminating measure than accuracy. In *International Joint Conference on Artificial Intelligence*, pages 519–526.

Lu, H., Setiono, R., and Liu, H. (1996). Effective data mining using neural networks.

MacKay, D. (1997). Introduction to Gaussian processes. *NATO ASI series. Series F: computer and system sciences*, pages 133–165.

MacKay, D. (1999). Comparison of approximate methods for handling hyperparameters. *Neural Computation*, 11(5):1035–1068.

Makhoul, J., Kubala, F., Schwartz, R., and Weischedel, R. (1999). Performance measures for information extraction. In *Broadcast News Workshop'99 Proceedings*, page 249. Morgan Kaufmann.

Maloof, M. (2003). Learning when data sets are imbalanced and when costs are unequal and unknown. In *ICML-2003 workshop on learning from imbalanced data sets II*.

Minka, T. (2002). Expectation propagation for approximate Bayesian inference. *update*, 500(2):5z.

Minka, T. (2003). A comparison of numerical optimizers for logistic regression. *Unpublished draft*.

Neal, R. (1996). *Bayesian learning for neural networks*. Springer.

Neal, R. (1999). Regression and classification using Gaussian process priors. In *Bayesian Statistics 6: Proceedings of the Sixth Valencia International Meeting, June 6-10, 1998*, page 475. Oxford University Press.

Neyman, J. and Pearson, E. (1928). On the use and interpretation of certain test criteria for purposes of statistical inference: Part II. *Biometrika*, pages 263–294.

Nickisch, H. and Rasmussen, C. (2008). Approximations for Binary Gaussian Process Classification. *Journal of Machine Learning Research*, 9:2035–2078.

Odom, M. and Sharda, R. (1990). A neural network model for bankruptcy prediction. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 163–168.

Opper, M. and Archambeau, C. (2009). The variational gaussian approximation revisited. *Neural Computation*, 21(3):786–792.

Paciorek, C. J. and Schervish, M. J. (2004). Nonstationary Covariance Functions for Gaussian Process Regression. In *In Proc. of the Conf. on Neural Information Processing Systems (NIPS)*. MIT Press.

Platt, H. D. and Platt, M. B. (1991). A note on the use of industry-relative ratios in bankruptcy prediction. *Journal of Banking & Finance*, 15(6):1183 – 1194.

Rasmussen, C. and Williams, C. (2006). *Gaussian processes for machine learning*. Springer.

Ravn, M. and Uhlig, H. (2002). On adjusting the Hodrick-Prescott filter for the frequency of observations. *Review of Economics and Statistics*, 84(2):371–376.

Ripley, B. (1996). *Pattern recognition and neural networks*. Cambridge university press.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14(5):465–471.

Russell, S. J. and Norvig, P. (1995). *Artificial intelligence: a modern approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Sagar, V. and Alex, K. (1999). Hybrid fuzzy logic and neural network model for fingerprintminutiae extraction. In *Neural Networks, 1999. IJCNN'99. International Joint Conference on*, volume 5.

Schölkopf, B. and Smola, A. (2002). *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press.

Serrano-Cinca, C. (1996). Self organizing neural networks for financial diagnosis. *Decis. Support Syst.*, 17(3):227–238.

Stein, M. (1999). *Interpolation of spatial data: some theory for kriging*. Springer.

Treacy, W. F. and Carey, M. S. (1998). Credit risk rating at large U.S. banks. *Federal Reserve Bulletin*, (Nov):897–921.

Turner, T. (2007). *Beginner's Guide to Day Trading Online*. Adams Media Corporation.

Šušteršič, M., Mramor, D., and Zupan, J. (2009). Consumer credit scoring models with limited data. *Expert Syst. Appl.*, 36(3):4736–4744.

Wada, Y. and Kawato, M. (1993). Neural network model for arm trajectory formation using forward and inverse dynamics models. *Neural Networks*, 6(7):919–932.

Williams, C. and Barber, D. (1998a). Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351.

Williams, C. and Barber, D. (1998b). Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351.

Wilson, R. and Sharda, R. (1994). Bankruptcy prediction using neural networks. *Decision Support Systems*, 11(5):545–557.

Witten, I. and Frank, E. (2005). Data mining-practical machine learning tools and techniques. Second Edition.

Zhang, G., Y. Hu, M., Eddy Patuwo, B., and C. Indro, D. (1999). Artificial neural networks in bankruptcy prediction: General framework and cross-validation analysis. *European Journal of Operational Research*, 116(1):16–32.