ANOMALY DETECTION FOR INTERNET BANKING USING SUPERVISED LEARNING ON HIGH DIMENSIONAL DATA

ADAGRAD WIH DIAGONAL MATRICES ON THE LOG-LIKELIHOOD COMPUTED OVER LABELLED TRANSACTIONS

by

H. S. Selman BSc

in partial fulfilment of the requirements for the degree of

Master of Science

in Applied Mathematics

with specialisation Statistics

at the faculty EEMCS of Delft University of Technology,

to be defended publicly on Monday January 26, 2015 at 11:00 AM.

Student number:	1509551	
Project duration:	March 17, 2014 – December 16, 2014	
Supervisor:	Prof. dr. ir. G. Jongbloed,	TU Delft
Thesis committee:	Dr. ir. F. H. van der Meulen,	TU Delft
	Dr. L. J. P. van der Maaten,	TU Delft
	W. Ipenburg MSc,	Rabobank

This thesis is confidential and cannot be made public until January 26, 2015.

TU Delft Delft University of Technology

PREFACE

For the department FEC (Financial Economic Crime) of Rabobank, I have done research into anomaly detection in internet banking as my graduation project. We look at anomalies because Rabobank wants to prevent abuse. I am part of the research team CCR (CyberCrime ContainmentControl Research) which works together with the teams of VCM (Virtual Channel Monitoring) and TM (Transaction Monitoring).

During my project, Werner Ipenburg was my supervisor from the CCR team of Rabobank. The first time we met he introduced me to the assignment and was very clear on the fact that the project would be challenging, given the complexity of the problem. He warned me for the many struggles and disappointments that I would encounter during research. And he was right indeed. It cost a lot of time and effort to get to the results that are presented here. Werner especially helped me with my presentations to other colleagues of Rabobank. He provided useful feedback on the content and appearance of the presentations. I am very grateful for the assistance and guidance that got me to where I am.

One or two days a week I worked on my project at Rabobank. On these days I would meet with the research team CCR. In this way I could tell about my progression in the project and also stayed informed on the research that my colleagues were doing. Thank you, Raymond, Rebecca and Frank, for making my feel part of the team. Telling you about my project helped me to structure my story and your questions helped a lot to see my project from a broader perspective.

I would like to thank Frank van der Meulen and Laurens van der Maaten for their intellectual discussions about the problem. As my two supervisors from Statistics (Mathematics) and Pattern Recognition (Computer Science) respectively, they helped me during our weekly meetings to come up with ideas for the right model to use on the data Rabobank delivered. The problem came with many challenges: large volume of data, a lot of parameters and anonymous data. Frank and Laurens, thank you for your guidance and expertise.

During my project I got an office on the 6^{th} floor of EWI to do my research. Along with other fellow students we shared this office. I would like to thank Niels and Said for their fruitful discussions that enlightened my day. Also I would like to thank the employees that work on 6^{th} floor of EWI, Statistics and Probability. Especially the technician, Carl, and the secretary, Leonie. Thank to all of you for making me feel at home and also thanks for making me feel part of the department during this period.

H. S. Selman BSc
Delft, December 2014

Contents

Preface iii

List of Figures vii

List of Abbreviations ix

List of Symbols xi

Introduction 1

1 Experimental setup 3
- 1.1 Extracting features from the data set . . . 3
  - 1.1.1 Overview . . . 4
  - 1.1.2 How features are extracted from `.ccr` files . . . 4
  - 1.1.3 Split into numerical and categorical features . . . 5
  - 1.1.4 Binary setting for categorical variables . . . 5
  - 1.1.5 Feature scaling for numerical variables . . . 5
  - 1.1.6 Feature selection . . . 7

2 Predicting cases 9
- 2.1 Dependencies between $\mathbf{x}$, s and y . . . 9
- 2.2 Assumptions . . . 9
  - 2.2.1 Logistic regression . . . 11
- 2.3 Estimating $p(s|\mathbf{x})$ . . . 11
- 2.4 Estimating $\mathbb{P}(y = 1|\mathbf{x})$ . . . 12
  - 2.4.1 Likelihood function . . . 12
  - 2.4.2 Determining the model parameters . . . 13
- 2.5 Incorporating priors . . . 16
  - 2.5.1 Maximum a posteriori estimation . . . 16
  - 2.5.2 Prior on γ . . . 16
  - 2.5.3 Prior on $\boldsymbol{\theta}$ . . . 17
  - 2.5.4 Regularization . . . 18
- 2.6 Objective function . . . 19

3 AdaGrad algorithm 21
- 3.1 AdaGrad with diagonal matrices . . . 21
  - 3.1.1 Diagonal matrices . . . 22
- 3.2 Regularization of $\boldsymbol{\theta}$ . . . 22

4 Testing the final model 27
- 4.1 Procedure . . . 27
  - 4.1.1 Create data set . . . 28
  - 4.1.2 Validate λ . . . 28
  - 4.1.3 Train model . . . 29
  - 4.1.4 Test model . . . 29
- 4.2 Testing the model on data generated from the model . . . 32
  - 4.2.1 Examples . . . 32
- 4.3 Testing the model on Rabobank data . . . 38
  - 4.3.1 Test results . . . 38
  - 4.3.2 Taking a constant value for c . . . 41

5 Conclusions 43

6 Discussion 45

A Previous work **47**
A.1 Machine learning . . . 47
A.2 Supervised learning . . . 47
A.3 Online learning . . . 47
A.4 PU-learning . . . 48

B Background information **49**
B.1 Probit regression . . . 49
B.2 Convex optimization . . . 50
B.2.1 Penalized convex optimization . . . 50
B.2.2 Using ℓ_2-projection with Adagrad with diagonal matrices . . . 51

C Missing data problem **53**
C.1 Likelihood function . . . 53
C.1.1 The complete data likelihood . . . 53
C.2 Expectation–Maximization algorithm . . . 54
C.2.1 E - step . . . 54
C.2.2 M - step . . . 55
C.3 Data Augmentation method . . . 55
C.3.1 Step 0: choose a start value for $\boldsymbol{\theta}$ . . . 56
C.3.2 Step 1: data augmentation . . . 56
C.3.3 Step 2: updating parameter $\boldsymbol{\theta}$ . . . 56

D MATLAB code **59**
D.1 AdaGrad algorithm . . . 59
D.1.1 Run model . . . 59
D.1.2 Projection algorithm . . . 75
D.1.3 Objective function for AdaGrad . . . 77

Bibliography **79**

LIST OF FIGURES

1.1 Available information included for every transaction 3

2.1 Situation 9
2.2 Beta prior, shape parameters: $\alpha = 15$ and $\beta = 1$. 17

4.1 How the model works. 27
4.2 How the data set was created. 28
4.3 How λ was validated. 29
4.4 How the model was trained. 30
4.5 How the model was tested. 31
4.6 ROC curves for Example 1, $AUC_{train} = 0.9920$ (solid red), $AUC_{test} = 0.9914$ (solid blue), $AUC_{train} = 0.9524$ (dotted red), $AUC_{test} = 0.9544$ (dotted blue). 33
4.7 ROC curves for Example 2, $AUC_{train} = 0.9899$, $AUC_{test} = 0.9871$. 34
4.8 ROC curves for Example 3, $AUC_{train} = 0.9933$, $AUC_{test} = 0.9872$. 35
4.9 ROC curves for Example 4, $AUC_{train} = 0.9282$, $AUC_{test} = 0.9467$. 36
4.10 AUC values for Example 5. 37
4.11 ROC curves, $AUC_{train} = 0.5279$, $AUC_{test} = 0.4792$. 39
4.12 ROC curves, $AUC_{train} = 0.7234$, $AUC_{test} = 0.7510$. 39
4.13 All the scores on the cases, when the top-75 features are used. 40
4.14 All the scores on the non-cases, when the top-75 features are used. 40
4.15 Red line and blue line indicate AUC values when c is estimated by the model. The stars indicate the AUC values when c is fixed. 41

List of Abbreviations

ACC	Accuracy
AdaGrad	Adaptive Gradient method
AUC	Area Under the ROC Curve
CCR	CyberCrime ContainmentControl Research
CF	Confusion Matrix
F	Feature (prefix of a feature)
FEC	Financial Economic Crime (Department of Rabobank)
FN	False Negative
FP	False Positive
FPR	False Positive Rate
GD	Gradient Descent
GLM	Generalized Linear Models
KKT	Karush–Kuhn–Tucker
logit function	logistic function
MAP	Maximum A Posteriori
ML estimate	Maximum Likelihood estimate
MLE	Maximum Likelihood Estimation
PCA	Principal Component Analysis
PU learning	Positive–Unlabelled learning
ROC	Receiver Operating Characteristics
SGD	Stochastic Gradient Descent
SPC	Specificity
T	Top–values (prefix of a numerical feature)
TM	Transaction Monitoring (Part of FEC Rabobank)
TN	True Negative
TP	True Positive
TPR	True Positive Rate
V	Values (prefix of a categorical feature)
VCM	Virtual Channel Monitoring (Part of FEC Rabobank)

List of Symbols

$\boldsymbol{\theta}$	One model parameter (intercept) and parameters corresponding to the features
γ	Reparametrized version of c
λ	Lagrange multiplier or constant used for regulating $\boldsymbol{\theta}$
ℓ	Log-likelihood function: $\log(\mathscr{L})$
$\mathscr{L}$	Likelihood function
$\mathbb{P}$	Probability
∇	Gradient
Θ	Optimizing restricted to the space Θ: $\boldsymbol{\theta} \in \Theta$
Γ	Gamma function
τ	Threshold
$\mathbb{E}$	Expectation
$\|\cdots\|_A$	Mahalanobis norm with respect to A
$\Pi_\Theta^A(\kappa)$	Projection of κ on Θ according to A
$\mathbf{x}_i$	Features of transaction i
η_t	Step size (or learning rate) at iteration t
$\boldsymbol{\theta}_t$	Parameter $\boldsymbol{\theta}$ after t iteration
Beta	Beta function
$\hat{s}_i(\tau)$	Predicted label of transaction i, given threshold τ
c	Probability of labelling a case
f	Logistic function or probit function
G_t	Outer product matrix at iteration t
H_t	Correction factor on η_t
n	Number of transactions used for training
Q	Objective function
Q_i	Contribution of the i^{th} transaction to Q
R	Regret
s	labelled ($s = 1$) or unlabelled ($s = 0$)
y	case ($y = 1$) or non-case ($y = 0$)

INTRODUCTION

Nowadays, a high number of transactions are performed via internet banking. Rabobank processes more than 10 million transactions per day. Most of these transactions are (part of) normal behaviour. On the other hand, some transactions are considered to be out of the ordinary. These anomalous events occur relatively infrequently (less than 10 per day). Employees, that try to find these anomalous events, combine the transactions data, historical knowledge of the anomalous events and their expertise to detect and quantify them. Several types of anomalies are considered to be interesting and so they are labelled. These anomalies need to be detected, so they can be prevented in the future. The employees try to find events similar to known anomalies. Characteristics of anomalies change over time and employees also need to detect this slightly changed, but similar, behaviour. It is not our goal to detect completely new types of anomalies. In this thesis, the focus lies on finding events similar to the known anomalies. In order to assist these employees, a model that uses the transaction data and incorporates known anomalous events is built. Our model is able to score new incoming transactions and use these to update the model parameters. The scores can be returned to the employees to assist them in finding transactions that are similar to a particular type of anomaly.

A transaction that Rabobank finds interesting is called a *case*, from here on.

The transactions that are labelled by Rabobank are considered to be cases. For the unlabelled transactions, it is unknown whether it is a case. Most of the unlabelled transactions will be *non-cases*, but the assumption is that there could be cases that are unlabelled. Hence the data set contains positive and unlabelled transactions. That is why this problem can be solved using Positive–Unlabelled learning (PU learning).

For every transaction i, we have a vector $\mathbf{x}_i$ (the characteristics of transaction i), a label $s_i \in \{0,1\}$, a random ID and a time stamp. Vector $\mathbf{x}_i$ contains the *features* of transaction i.

PU learning falls into the category of supervised anomaly detection, a subfield of machine learning. In Appendix B, more details are given about machine learning and especially about previous work in PU–learning. In this field, statistics and optimization are used. Supervised learning is a machine learning task where labelled data is used. A supervised learning algorithm uses the training data to produce a probability density function that can be used to map new examples to a probability, a score between 0 and 1. Given a threshold, this score can be translated in a label (0 or 1). In the optimal scenario, the supervised learning algorithm produces a function that maps all the new examples to the correct label.

The data set is split into two parts, a training set and a test set. Then traditionally, the complete training set is used at once to train the model and the test set is used to test the performance of the model. But training with the complete training set is rather expensive considering the amount of transactions available for training. Our model will be an online learning model, where only one training example is used per iteration. The advantage is that we only need one example from the training set and so we don't need a lot of memory to store all the data. Also we could use this property in practice, namely scoring a new transaction given the current model parameters and using that transaction to update the model parameters.

Using the training set, we have tried to find the model parameters that fit the training data the best. Assume that all the events, in both the training set and test set, are independent and identically distributed. Every event $\mathbf{x}$ that is an anomaly is labelled $s = 1$, while the rest is labelled $s = 0$. To fit the model parameters, we look at the the likelihood function, which is the product of the probability density function $p(s|\mathbf{x})$. This approach basically fits the conditional distribution of the labels, given the data and the model parameters, for every type of anomaly. This approach also gives us for every event (from the test set) the probability that it should be labelled as a similar event.

Outline of this thesis:

- Chapter 1 discusses the data set from Rabobank, and how the features are extracted from the data set.
- Chapter 2 describes the goal we want to achieve and explains how we want to reach our goal.
- Chapter 3 describes the final model, AdaGrad with diagonal matrices.
- Chapter 4 describes the testing procedure and discusses the results.
- Chapter 5 summarizes the main results and the conclusions.
- Chapter 6 sums up the improvements and remarks that can be made.

The appendices describe the following subjects:

- Appendix A discusses previous work done in the area of machine learning, online learning, and PU-learning.
- Appendix B discusses probit regression, convex optimization and ℓ_2-regularization on AdaGrad with diagonal matrices.
- Appendix C describes how our model would look like if the problem is treated as a missing data problem. Every iteration of the model will consist of two steps: imputing the missing data and updating the parameters to fit the data better.
- In Appendix D, the most important MATLAB files are included.

1

EXPERIMENTAL SETUP

In this chapter, the properties of the data set and the adjustments to the data set are discussed. First, the structure of the transactions and feature extraction are discussed. Then, the adjustments, that are made on the extracted features, are discussed.

1.1. EXTRACTING FEATURES FROM THE DATA SET

The characteristics of internet banking transactions contain information about the client and their computer. This information is private, and needs to be treated with care. That is why the data set, used for this thesis, only contains anonymous data. The features are binned into 256 bins.

For this thesis, Rabobank provided a data set which consists approximately one 1000^{th} of the transactions from November 2013 up till October 2014. The data set is divided over 1521 files (each containing approximately 1600 transactions), which gives us about 2.5 million transactions in total. All 806 anomalies of this period, discovered by Rabobank, are delivered in separate files. Every transaction consists of the following information, as listed Figure 1.1. The vector $\mathbf{x}_i$ contains the characteristics of transaction i.

Transaction i:
- Random ID
- Time stamp
- Case type
- (Anonymous) characteristics of the transaction $\mathbf{x}_i$:
  - categorical features
  - numerical features
- $s_i = \begin{cases} 1, & \text{if transaction } i \text{ is labelled (as a case)} \\ 0, & \text{if transaction } i \text{ is unlabelled} \end{cases}$

Figure 1.1: Available information included for every transaction

From here on, an anomaly is called a case and every case belongs to a class of a certain case type. Every case type gets a name (e.g. N9), consisting of a capital letter (A – Z) and a number (0 – 9). In this way, we are able to test the model on a specific case type or on a combination of case types. Rabobank used the name O0 for the non-cases, to group them as well. Every transaction has an (unique) random ID to identify which transaction it is. To prevent problems with uniqueness and also to make it easier to find this transaction, also the time stamp of the transaction is included. Every transaction has a case type, categorical features and numerical features. For this research, the data was prepared such that everything is anonymous. The meaning of the features and the names of the features are unknown to us. The features are derivatives of the

actual characteristics of the transactions. This forced us to use only statistics and optimization to optimize the model and the results. It also makes it possible to publish the model and the results.

First, a list of case types is chosen. The cases of these case types are labelled $s = 1$. The rest of the transactions are unlabelled, hence $s = 0$. We are interested in whether a transaction is a case. The assumption is that the cases provided by Rabobank are correctly labelled, but maybe there are also cases that are unlabelled. That is why y is introduced.

$$y_i = \begin{cases} 1, & \text{if transaction } i \text{ is a case} \\ 0, & \text{if transaction } i \text{ is not a case} \end{cases}$$

For a case, we have $y = 1$. We assume that for the rest of the transactions y is unknown.

1.1.1. OVERVIEW

For a transaction i:

- $\mathbf{x}_i$ are the features.
- s_i is the label that is given to a transaction.

$$s_i = \begin{cases} 1, & \text{if the transaction is labelled,} \\ 0, & \text{if the transaction is unlabelled.} \end{cases}$$

- y_i is the unobserved variable, in which we are interested.

$$y_i = \begin{cases} 1, & \text{if the transaction is a case,} \\ 0, & \text{if the transaction is a non-case.} \end{cases}$$

Furthermore, $\mathbf{x}$ and s are called the observed variables and y is called unobserved variable.

Vectors are written bold.

1.1.2. HOW FEATURES ARE EXTRACTED FROM .CCR FILES

The data files are provided as `.ccr` files. Because of the huge amount of transactions that have to be stored, these files are built with sequences of 0's and 1's. Eight bits (a 0 or 1) make a byte. Every byte corresponds to one of the 256 categories ($2^8 = 256$), 0–255. Later on, more on how these categories are used to extract the features, is discussed.

The first part of each `.ccr` file contains a receipt header (which indicates which rules are used to make the file). After that, a couple of transactions are listed after each other. For each transaction, first the random ID and the time stamp are spelled, using the ASCII value of the bytes. After that, the word DATA is spelled, again using the ASCII values of the bytes. This is to assure that the next 10.000 numbers between 0 and 255 are the data that belongs to that specific random ID and time stamp. It is possible that only a part of the 10.000 reserved spots are used. In our data set, only 1523 features are used.

The first feature will be named F0001. The second feature is F0002. And so on.

Now the meaning of the 256 categories, chosen to describe the features, is looked at. The categories 0 to 255 correspond to following meanings:

0 : Empty.
1 – 200 : First of all, each feature is divided over a maximum of 200 bins, which corresponds to its underlying value. Bins are chosen in such a way that the bins are evenly filled.
201 – 210 : Top-values. If a single value comes along often, more than 5% of all the events, then the feature gets another category and the values are excluded from the first 200 bins. Most of the time this corresponds to a special situation, for example a lot of empty values, a default or a boolean.
211 – 250 : Spare. Saved for later use.
251 : Virtual class. When feature falls into 1 - 200, then also virtual in 251.
252 : Technical error.
253 : Logical error.
254 : Below the minimum value. So below bin 1.
255 : Above the maximum value. So above bin 200.

1.1.3. SPLIT INTO NUMERICAL AND CATEGORICAL FEATURES

The 1523 features, extracted from the `.ccr` files, are a combination of numerical and categorical features. The values 0 – 200 are numerical. The values 201 – 255 are categorical. That is why, every feature is separated into two features, one numerical and one categorical. Below is shown, what we do to split them:

Feature (from `.ccr` file)	Numerical feature	Categorical feature
0 :	0	V
1 – 200 :	1 – 200	V
201 – 210 :	0	A – J
251 – 255 :	0	V – Z

Numerical features will be named using the V (Values). Categorical features will be named using the T (Top-values). For example, V0001 and T0001 correspond to F0001.

This means that 3046 features are available per transaction.

1.1.4. BINARY SETTING FOR CATEGORICAL VARIABLES

A total of 1523 categorical variables are presented by the letters A – Z. In order to use them, each variable is split into 26 binary variables, containing only one 1 and twenty-five times a 0. The 1 is on the position of the letter in the alphabet. For example, “B” corresponds to a 1 on the second position:

$$0,\ 1,\ 0,\ \ldots 0.$$

1.1.5. FEATURE SCALING FOR NUMERICAL VARIABLES

A total of 1523 numerical variables that can take integer values from 0 to 200. In order to obtain better results the numerical variables can better be normalized. For this the sample mean and sample variance are needed. Keep in mind that all 1523 variables have to be normalized by the accompanying sample mean and sample variance. This seems an easy task, but this can be quite tricky.

In Chan, Golub, and LeVeque [1], all sorts of algorithms for computing the sample variance are discussed. First, simple algorithms and their disadvantages are discussed. Afterwards, an online algorithm is proposed to compute the sample variance online.

TWO-PASS ALGORITHM

Fundamentally, one would compute the sample mean:

$$\bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i, \tag{1.1}$$

and use the sample mean to compute the sample variance, $\frac{S}{n}$ (biased) or $\frac{S}{n-1}$ (unbiased), where S is the sum of squares of the deviations from the mean and computed as follows:

$$S = \sum_{i=1}^{n} (\mathbf{x}_i - \bar{\mathbf{x}})^2. \tag{1.2}$$

The formulas (1.1) and (1.2) define a straightforward algorithm for computing S. This will be called the *standard two-pass algorithm*, since it requires passing through the data twice: once to compute $\bar{\mathbf{x}}$ and then again to compute S. For our model, this method is impossible to use, since every record is only observed once. Also, the large size of the data set prevents this. All transactions cannot be placed in memory to compute the sample mean and sample variance.

ONE-PASS ALGORITHM

The *textbook one-pass algorithm* that only needs one sweep through the data. By keeping track of three values, for every numerical variable, the sample mean and sample variance can be determined: n_{seen} (number of records seen so far), "total" = $\sum_{i=1}^{n_{\text{seen}}} \mathbf{x}_i$ and "squared total" = $\sum_{i=1}^{n_{\text{seen}}} \mathbf{x}_i^2$.

This is because formula (1.2) for S can be rewritten as:

$$S = \sum_{i=1}^{n} \mathbf{x}_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} \mathbf{x}_i\right)^2.$$

The sample mean is defined as $\hat{\mu} = \frac{1}{n_{\text{seen}}} \sum_{i=1}^{n_{\text{seen}}} \mathbf{x}_i$ and the biased sample variance as $\hat{\sigma}^2 = \frac{1}{n_{\text{seen}}} \sum_{i=1}^{n_{\text{seen}}} (\hat{\mu} - \mathbf{x}_i)^2$. Note that the unbiased sample variance can be obtained from the biased sample variance by $\bar{\sigma}^2 = \frac{n_{\text{seen}}}{n_{\text{seen}} - 1} \cdot \hat{\sigma}^2$. It is easy to see that the sample mean is obtained by dividing "total" by n_{seen}. To see how to calculate the biased sample variance one can write out the squared expression to get $\hat{\sigma}^2 = -\hat{\mu}^2 + \frac{1}{n_{\text{seen}}} \sum_{i=1}^{n_{\text{seen}}} \mathbf{x}_i^2$.

Indeed, only three values have to be updated every record to obtain the sample mean and the unbiased sample variance. For now, a constant data set is used. In only one sweep through the data, the sample mean and unbiased sample variance are obtained. But still, it is very inefficient considering that in practice the quantities "total" and "squared total" will become very large and eventually will be computed with some rounding error.

ONLINE ALGORITHM

For this thesis, the textbook one-pass algorithm would suffice. The complete data set can be used to determine the sample mean and sample variance of each feature. When the model is used in the online setting, where every transaction is loaded, used to update the parameters and deleted afterwards, the sample means and sample variances are needed to normalize the features. Then the data set on which the model needs to be trained, is changing constantly. Maybe, using a method that can update the sample mean and sample variance, when a new transaction is added to the data set.

There is a method that is quite accurate and that can update the sample mean and sample variance in an online setting. Furthermore, the method computes a running variance. That is, the method updates the variance after each transaction. The features do not need to be kept in memory, since no second pass through the data is needed. This method was first purposed in Welford [2]. In the *Art of Computer Programming* by Knuth [3], this method is also mentioned.

Initialize $M_1 = \mathbf{x}_1$ and $S_1 = 0$. For subsequent $\mathbf{x}_i$'s, use the following recurrence formulas

$$M_k = M_{k-1} + \frac{\mathbf{x}_k - M_{k-1}}{k},$$

$$S_k = S_{k-1} + (\mathbf{x}_k - M_{k-1}) \cdot (\mathbf{x}_k - M_k),$$

for $2 \leq k \leq n$. The k^{th} online estimate of the sample variance is $\sigma_k^2 = \frac{S_k}{k-1}$.

Updating the sample mean and sample variance in an online setting, also means that every transaction is normalized in a different way. That is why in this case, the sample mean and sample variance are determined over the complete data set, before the model is run. In practice, the sample mean and sample variance can be computed over a large proportion of the data to determine the sample mean and sample variance and keep them while running the model.

1.1.6. FEATURE SELECTION

In the data set Rabobank presented, there was no feature selection done on the features. Features could be correlated.

A colleague at Rabobank studied the data set and discovered that about 1000 features are constant, over all transactions. These features were filtered out. Also, by comparing all features, it was discovered that about 500 features were exact copies of other features. These variables were left out as well. In practice, if one assumes that these excluded features are in fact important, we recommend regularly checking whether these features stay constant, or stay an exact copy of other features.

EXPERIMENTAL ORDERING OF FEATURES

R was used to order the remaining 1435 features. Using the function `chisq.test` (Pearson's Chi-Squared test of independence) was used to compare every feature to the 'label' (0 for non-cases and 1 for cases). Only a subset of 4618 transactions was used. This ordering is therefore called experimental and the result does not per definition represent the complete data set, but we are eager to know whether feature selection techniques, like Principal Component Analysis (PCA), have potential.

This R function (`chisq.test`) was used to compute p-values, from the asymptotic chi-squared distribution of the test statistic χ^2:

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}},$$

where $O_{i,j}$ is an observed frequency, $E_{i,j}$ the expected frequency, r the number or rows and c the number of columns in the table (consisting a feature and the label). The expected frequency is computed as follows:

$$E_{i,j} = \frac{\left(\sum_{n_c=1}^{c} O_{i,n_c}\right) \cdot \left(\sum_{n_r=1}^{r} O_{n_r,j}\right)}{N},$$

where N is sum of all entries of the table described above. The degrees of freedom is equal to $d = r + c - 1$.

This test of independence is used to determine whether the null hypothesis, that a feature and the label are independent, is true. If the chi-squared probability (the p-value) is less than or equal to 0.05 (common choice, but could choose another critical point), than the null hypothesis will be rejected.

These p-values can be used to make an ordering in the remaining features. The largest p-values indicate some dependence between the label and these features. This information could be used to only take, for example the top-100 features, and see whether the performance of the model improves. First the performance of the model, using all of the 1435 remaining features, is studied. We believe that our model should be able to work without this experimental ordering of features.

2

PREDICTING CASES

The goal is to predict the probability $\mathbb{P}(y = 1|\mathbf{x})$, since it tells us whether a transaction could be a case. This probability cannot be computed directly, since the value of y is unknown for the unlabelled transactions. Otherwise, logistic regression could be used directly to find the probability $\mathbb{P}(y = 1|\mathbf{x})$.

To reach our goal, the dependencies which are used to form the model, and the needed assumptions in order to define the parameters of the model, are discussed.

2.1. DEPENDENCIES BETWEEN $\mathbf{x}$, s AND y

In Figure 2.1 is shown how the situation looks like.

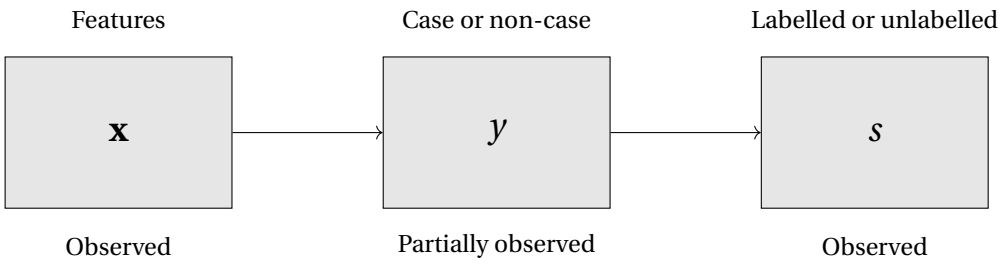

Figure 2.1: Situation

2.2. ASSUMPTIONS

Assumptions are needed to estimate $\mathbb{P}(y = 1|\mathbf{x})$.

Assumption 1. *Labelled transactions are cases, which is written as:*

$$s = 1 \Rightarrow y = 1. \tag{2.1}$$

Corollary 2.1. *The first assumption (2.1) is equivalent to:*

$$\mathbb{P}(y = 1|s = 1, \mathbf{x}) = 1, \tag{2.2}$$

The contraposition from traditional logic is used to find that Assumption 2.1 is also equivalent to:

$$y = 0 \Rightarrow s = 0, \tag{2.3}$$

and can be also written as:

$$\mathbb{P}(s=1|y=0,\mathbf{x})=0, \tag{2.4}$$

which translates to: Non-cases are never labelled.

From Assumption 1 (2.1), it appears that the difficulty lies in finding cases amongst the non-labelled transactions. Since for every transaction the pair $(\mathbf{x}, s)$ is observed, the probability function $p(s|\mathbf{x})$ is looked at. It can be expressed in terms of the observed label s, the true label y and the features $\mathbf{x}$:

$$\begin{aligned} p(s|\mathbf{x}) &= \sum_y p(s,y|\mathbf{x}) \\ &= \sum_y p(s|y,\mathbf{x})\,p(y|\mathbf{x}) \end{aligned} \tag{2.5}$$

To obtain $p(s|y,\mathbf{x})$, a second assumption is needed. This assumption is called 'Selected completely at random' in Elkan and Noto [4].

Assumption 2. *Cases are labelled with probability c.*

$$\mathbb{P}(s=1|\mathbf{x},y=1)=\mathbb{P}(s=1|y=1)=c\in[0,1]. \tag{2.6}$$

The second assumption (2.6) can be formulated as: $\mathbf{x}$ and s are independent given y. Rabobank likes to think that the second assumption is reasonable. The following result is found in Elkan and Noto [4] (as Lemma 1). Only, the proof is different.

Corollary 2.2. *When Assumption 1 (2.1) and Assumption 2 (2.6) hold, the following holds:*

$$\mathbb{P}(y=1|\mathbf{x})=\frac{\mathbb{P}(s=1|\mathbf{x})}{c}, \tag{2.7}$$

where $c\in[0,1]$ is the expected fraction of cases that is being labelled.

Proof. The corollary above follows from:

$$\begin{aligned} p(s=1|\mathbf{x}) &\overset{(2.5)}{=} p(s=1|y=1,\mathbf{x})\,p(y=1|\mathbf{x})+p(s=1|y=0,\mathbf{x})\,p(y=0|\mathbf{x}), \\ &\overset{(2.4)}{=} p(s=1|y=1,\mathbf{x})\,p(y=1|\mathbf{x}), \\ &\overset{(2.6)}{=} c\cdot p(y=1|\mathbf{x}). \end{aligned}$$

□

In equation (2.5), $p(y|\mathbf{x})$ is needed. Define $p(y|\mathbf{x})$ for transaction i, as logistic regression with parameter $\boldsymbol{\theta}$:

$$p(y|\mathbf{x})=\begin{cases} f(\boldsymbol{\theta}^T\mathbf{x}), & \text{if } y=1, \\ 1-f(\boldsymbol{\theta}^T\mathbf{x}), & \text{if } y=0. \end{cases} \tag{2.8}$$

where $f(x)=\dfrac{1}{1+e^{-x}}$ is the logistic function. This can also be written as:

$$p(y|\mathbf{x})=y\cdot f(\boldsymbol{\theta}^T\mathbf{x})+(1-y)\cdot 1-f(\boldsymbol{\theta}^T\mathbf{x}). \tag{2.9}$$

Another choice for f would be the probit function, if probit regression is used instead of logistic regression. Both choices for f transform a linear combination of the variables, using parameter θ, into a probability. In Appendix B.1, probit regression is discussed and the associated function f and its derivative are given.

2.2.1. LOGISTIC REGRESSION

Logistic regression is a type of statistical classification model. It is used to predict a categorical response variable, in particular a binary response variable. Logistic regression is used to predict the odds of being a case based on the values of the variables. The odds are defined as the probability that a particular outcome is a case, say p, divided by the probability that it is a non-case, $1-p$. The natural logarithm of the odds, called the *logit*, is then fitted using linear regression:

$$\begin{aligned}
\log\left(\frac{p}{1-p}\right) &= \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n \Leftrightarrow \\
\frac{p}{1-p} &= e^{\beta_0+\beta_1 x_1+\ldots+\beta_n x_n} \Leftrightarrow \\
\frac{1}{p} - 1 &= e^{-\beta_0-\beta_1 x_1-\ldots-\beta_n x_n} \Leftrightarrow \\
p &= \frac{1}{1+e^{-\beta_0-\beta_1 x_1-\ldots-\beta_n x_n}}.
\end{aligned} \tag{2.10}$$

The combination of the variables and the parameters, $\beta_0+\beta_1 x_1+\ldots+\beta_n x_n$, is usually denoted as $\beta^T\mathbf{x}$, where $\mathbf{x} = [1, x_1, \ldots, x_n]^T$.

The logistic function, also referred to as logit function, is the link function for logistic regression. The link function links the variables and the response variable in a continuous way.

THE DERIVATIVE OF THE LOGIT FUNCTION

The derivative of the function f is needed, further on. If the logistic function

$$f(x) = \frac{1}{1+\exp(-x)} \tag{2.11}$$

is used, then

$$f'(x) = \frac{1}{2+\exp(-x)+\exp(x)}. \tag{2.12}$$

2.3. ESTIMATING $p(s|\mathbf{x})$

The corollary of Assumption 1 (2.4) and Assumption 2 (2.6) make it possible to determine the conditional distribution $\mathbb{P}(s|y,\mathbf{x})$:

		s	
		0	1
y	0	1	0
	1	$1-c$	c

This can be translated as:

$$p(s|y,\mathbf{x}) = \begin{cases} c^s(1-c)^{1-s}, & \text{if } y=1 \\ 1-s, & \text{if } y=0, \end{cases} \tag{2.13}$$

where c is the fraction of fraud events ($y=1$) to be labelled as fraud ($s=1$). The expression (2.13) is rewritten as follows:

$$p(s|y,\mathbf{x}) = y\left(c^s(1-c)^{1-s}\right) + (1-y)(1-s). \tag{2.14}$$

Implementing (2.9) and (2.14) in expression (2.5) gives us:

$$p(s|\mathbf{x}) = c^s(1-c)^{1-s} \cdot f\left(\boldsymbol{\theta}^T\mathbf{x}\right) + (1-s)\cdot\left(1-f\left(\boldsymbol{\theta}^T\mathbf{x}\right)\right). \tag{2.15}$$

Note that (2.15) is equivalent to:

$$p(s|\mathbf{x}) = \begin{cases} c\cdot f(\boldsymbol{\theta}^T\mathbf{x}), & \text{if } s=1, \\ 1-c\cdot f(\boldsymbol{\theta}^T\mathbf{x}), & \text{if } s=0. \end{cases} \tag{2.16}$$

2.4. ESTIMATING $\mathbb{P}(y=1|\mathbf{x})$

Since s and $\mathbf{x}$ are known for each transaction, expression (2.16) can be used to estimate the parameters $\boldsymbol{\theta}$ and c. Once $\boldsymbol{\theta}$ is known, expression (2.9) is used to compute $\mathbb{P}(y=1|\mathbf{x})$.

Also, when c and $\mathbb{P}(y=1|\mathbf{x})$ is known, we can compute:

$$\begin{aligned}\mathbb{P}(y=1|\mathbf{x},s=0) &= \frac{\mathbb{P}(y=1,s=0|\mathbf{x})}{\mathbb{P}(s=0|\mathbf{x})},\\ &= \frac{\mathbb{P}(s=0|\mathbf{x},y=1)\cdot\mathbb{P}(y=1|\mathbf{x})}{\mathbb{P}(s=0|\mathbf{x})},\\ &\overset{(2.6)\&\ (2.7)}{=} \frac{(1-c)\cdot\mathbb{P}(y=1|x)}{1-c\cdot\mathbb{P}(y=1|\mathbf{x})}.\end{aligned}$$

This probability $\mathbb{P}(y=1|\mathbf{x},s=0)$ (also derived in [4]) could be used if $0\leq c\leq 1$ is estimated. When $c=0$, then $\mathbb{P}(y=1|\mathbf{x},s=0)=\mathbb{P}(y=1|\mathbf{x})$. When $c=1$, $\mathbb{P}(y=1|\mathbf{x},s=0)=0$ for all transactions, which is reasonable since $c=1$ implies that all cases are already labelled and the probability of finding new cases amongst the unlabelled examples is 0. If $0<c<1$, then this probability could be used instead of $\mathbb{P}(y=1|\mathbf{x})$ to improve the scores and the performance.

Well, how do we estimate the parameters $\boldsymbol{\theta}$ and c? Maximum Likelihood Estimation (MLE) is often used in statistics to estimate the parameters of a statistical model. This method selects the parameters that maximizes the likelihood function.

Definition 2.1. *The maximum likelihood estimators are defined by:*

$$(\boldsymbol{\theta}^*,c^*)=\arg\max_{\boldsymbol{\theta},c}\mathscr{L}(\boldsymbol{\theta},c|(\mathbf{x}_1,s_1),\ldots,(\mathbf{x}_n,s_n)).$$

Note that often the log-likelihood is used to find the estimated parameters $\boldsymbol{\theta}^*$ and c^*, since the logarithm changes products into sums of logarithms.

Corollary 2.1.1.

$$(\boldsymbol{\theta}^*,c^*)=\arg\max_{\boldsymbol{\theta},c}\ell(\boldsymbol{\theta},c|(\mathbf{x}_1,s_1),\ldots,(\mathbf{x}_n,s_n)),$$

where

$$\ell(\boldsymbol{\theta},c|(\mathbf{x}_1,s_1),\ldots,(\mathbf{x}_n,s_n))=\log\mathscr{L}(\boldsymbol{\theta},c|(\mathbf{x}_1,s_1),\ldots,(\mathbf{x}_n,s_n)).$$

2.4.1. LIKELIHOOD FUNCTION

Assumption 3. *The labels of transactions are independent and identically distributed (i.i.d.), given that the characteristics are known, i.e. s_i and s_j are i.i.d., given $\mathbf{x}_i$ and $\mathbf{x}_j$.*

The likelihood function is equal to

$$\begin{aligned}\mathscr{L}(\boldsymbol{\theta},c|(\mathbf{x}_1,s_1),\ldots,(\mathbf{x}_n,s_n)) &\overset{(i.i.d.)}{=} \prod_{i=1}^{n} p(s_i|\mathbf{x}_i)\\ &\overset{(2.15)}{=} \prod_{i=1}^{n}\left[c^{s_i}(1-c)^{1-s_i}f(\boldsymbol{\theta}^T\mathbf{x}_i)+(1-s_i)(1-f(\boldsymbol{\theta}^T\mathbf{x}_i))\right].\end{aligned}$$

To simplify this expression, the set W is defined:

$$W=\{i:s_i=1\},$$

which contains the labelled transactions. Note that $W^C=\{i:s_i=0\}$. In this way, expression (2.16) could be used instead to give:

$$\mathscr{L}(\boldsymbol{\theta},c|(\mathbf{x}_1,s_1),\ldots,(\mathbf{x}_n,s_n))=\prod_{i\in W}c\cdot f(\boldsymbol{\theta}^T\mathbf{x}_i)\cdot\prod_{i\in W^C}\left[1-c\cdot f(\boldsymbol{\theta}^T\mathbf{x}_i)\right]. \tag{2.17}$$

Hence, the log-likelihood function looks like

$$\begin{aligned}\ell(\boldsymbol{\theta}, c|(\mathbf{x}_1, s_1), \ldots, (\mathbf{x}_n, s_n)) &= \log \mathscr{L}(\boldsymbol{\theta}, c|(\mathbf{x}_1, s_1), \ldots, (\mathbf{x}_n, s_n)) \\ &= \sum_{i \in W} \log\big(c \cdot f(\boldsymbol{\theta}^T \mathbf{x}_i)\big) + \sum_{i \in W^C} \log\big(1 - c \cdot f(\boldsymbol{\theta}^T \mathbf{x}_i)\big), \end{aligned} \tag{2.18}$$

where f is the logistic function.

2.4.2. DETERMINING THE MODEL PARAMETERS

To determine the maximum likelihood estimators, the gradient is needed. This is the vector of derivatives with respect to every parameter $\boldsymbol{\theta}_j$ and c. Each component of the gradient could be set equal to 0 and try to find direct expressions for $\boldsymbol{\theta}^*$ and c^*. First we compute the gradient.

GRADIENT

The derivatives of the log-likelihood function are equal to:

$$\begin{aligned}\frac{\partial \ell}{\partial \theta_j} &= \sum_{i \in W} \frac{c \cdot x_{ij} \cdot f'(\boldsymbol{\theta}^T \mathbf{x}_i)}{c \cdot f(\boldsymbol{\theta}^T \mathbf{x}_i)} + \sum_{i \in W^C} \frac{-c \cdot x_{ij} \cdot f'(\boldsymbol{\theta}^T \mathbf{x}_i)}{1 - c \cdot f(\boldsymbol{\theta}^T \mathbf{x}_i)} \\ &= \sum_{i \in W} \frac{x_{ij} \cdot f'(\boldsymbol{\theta}^T \mathbf{x}_i)}{f(\boldsymbol{\theta}^T \mathbf{x}_i)} - \sum_{i \in W^C} \frac{c \cdot x_{ij} \cdot f'(\boldsymbol{\theta}^T \mathbf{x}_i)}{1 - c \cdot f(\boldsymbol{\theta}^T \mathbf{x}_i)}, \end{aligned}$$

and

$$\begin{aligned}\frac{\partial \ell}{\partial c} &= \sum_{i \in W} \frac{f(\boldsymbol{\theta}^T \mathbf{x}_i)}{c \cdot f(\boldsymbol{\theta}^T \mathbf{x}_i)} + \sum_{i \in W^C} \frac{-f(\boldsymbol{\theta}^T \mathbf{x}_i)}{1 - c \cdot f(\boldsymbol{\theta}^T \mathbf{x}_i)} \\ &= \frac{|W|}{c} - \sum_{i \in W^C} \frac{f(\boldsymbol{\theta}^T \mathbf{x}_i)}{1 - c \cdot f(\boldsymbol{\theta}^T \mathbf{x}_i)}, \end{aligned}$$

where the expressions (2.11) and (2.12) are used for functions f and f'.

Next section, we discuss reparameterizing c, which is needed to assure that $0 \leq c \leq 1$.

Direct expressions for the maximum likelihood estimators cannot be found from the expressions above. That is why, Adaptive Gradient Descent will be used to approximate the maximum likelihood estimators.

Then, Gradient Descend (see Section 2.4.2) and Stochastic Gradient Descent (see Section 2.4.2) are introduced. In Chapter 3, our final algorithm (the AdaGrad algorithm with diagonal matrices) is discussed.

REPARAMETERIZING c

Recall the definition of c:

$$\mathbb{P}(s = 1 | y = 1, \mathbf{x}) = c \in [0, 1].$$

This property has to be guaranteed during the maximization step. This maximization problem is therefore a constrained optimization problem, since $0 \leq c \leq 1$. But this constraint can be omitted by reparametrising c.

Reparametrise c to γ, via:

$$c = \frac{1}{1 + e^{-\gamma}}, \tag{2.19}$$

so that $\gamma \in \mathbb{R}$.

Hence, (2.19) is equivalent to:

$$\gamma = \log(c) - \log(1 - c). \tag{2.20}$$

When optimizing using γ instead of c, then the problem becomes an unconstrained optimization problem. Before optimizing, c is converted to γ using (2.20). After optimizing, c is obtained using (2.19).

The derivative with respect to γ, instead of c, needs to be computed. This is rather easy to do, using the chain rule:

$$\frac{\partial}{\partial \gamma}\ell(\boldsymbol{\theta}, c(\gamma)) = \frac{\partial}{\partial c}\ell(\boldsymbol{\theta}, c(\gamma)) \cdot \frac{\partial}{\partial \gamma}c(\gamma),$$

where

$$\frac{\partial}{\partial \gamma}c(\gamma) = \frac{1}{2 + e^{-\gamma} + e^{\gamma}}.$$

Hence, the log-likelihood function becomes:

$$\ell(\boldsymbol{\theta}, \gamma | (\mathbf{x}_1, s_1), \ldots, (\mathbf{x}_n, s_n)) = \sum_{i \in W} \log\left(\frac{f(\boldsymbol{\theta}^T \mathbf{x}_i)}{1 + e^{-\gamma}}\right) + \sum_{i \in W^C} \log\left(1 - \frac{f(\boldsymbol{\theta}^T \mathbf{x}_i)}{1 + e^{-\gamma}}\right), \tag{2.21}$$

and the derivatives become:

$$\frac{\partial \ell}{\partial \theta_j} = \sum_{i \in W} \frac{x_{ij} \cdot f'(\boldsymbol{\theta}^T \mathbf{x}_i)}{f(\boldsymbol{\theta}^T \mathbf{x}_i)} - \sum_{i \in W^C} \frac{x_{ij} \cdot f'(\boldsymbol{\theta}^T \mathbf{x}_i)}{1 + e^{-\gamma} - f(\boldsymbol{\theta}^T \mathbf{x}_i)},$$

and

$$\frac{\partial \ell}{\partial \gamma} = \left(|W| \cdot \left(1 + e^{-\gamma}\right) - \sum_{i \in W^C} \frac{f(\boldsymbol{\theta}^T \mathbf{x}_i)}{1 - \frac{f(\boldsymbol{\theta}^T \mathbf{x}_i)}{(1 + e^{-\gamma})}} \right) \cdot \frac{1}{2 + e^{-\gamma} + e^{\gamma}}.$$

GRADIENT DESCENT

In Gradient Descent (GD) methods, minimizes an objective function that can be written as a sum of functions over all observations in the training set. In our case, the training set is filled with n transactions. Also, the likelihood function needs to be maximized instead of minimized. By multiplying the likelihood function by -1, this method can be used.

The objective is: to maximize expression (2.21) over $\boldsymbol{\theta}$ and γ. Hence, our objective function:

$$Q(\boldsymbol{\theta}, \gamma) = -\sum_{i \in W} \log\left(\frac{f(\boldsymbol{\theta}^T \mathbf{x}_i)}{1 + e^{-\gamma}}\right) - \sum_{i \in W^C} \log\left(1 - \frac{f(\boldsymbol{\theta}^T \mathbf{x}_i)}{1 + e^{-\gamma}}\right), \tag{2.22}$$

needs to be minimized, where n is the number of transactions used for training and $\boldsymbol{\theta}$ and γ are the parameters which we would like to estimate. Define:

$$Q_i(\boldsymbol{\theta}, \gamma) = \begin{cases} -\log\left(\frac{f(\boldsymbol{\theta}^T \mathbf{x}_i)}{1 + e^{-\gamma}}\right), & \text{if } i \in W, \\ -\log\left(1 - \frac{f(\boldsymbol{\theta}^T \mathbf{x}_i)}{1 + e^{-\gamma}}\right), & \text{if } i \in W^C, \end{cases}$$

as the contribution associated with the i^{th} transaction.

Using GD, the following iteration would be repeated to obtain the maximum likelihood estimators:

$$\begin{aligned} \begin{bmatrix} \boldsymbol{\theta}_{t+1} \\ \gamma_{t+1} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{\theta}_t \\ \gamma_t \end{bmatrix} - \eta_t \nabla Q(\boldsymbol{\theta}_t, \gamma) \\ &= \begin{bmatrix} \boldsymbol{\theta}_t \\ \gamma_t \end{bmatrix} - \eta_t \sum_{i=1}^{n} \nabla Q_i(\boldsymbol{\theta}_t, \gamma), \end{aligned}$$

where η_t is a step size (often called learning rate in machine learning) in iteration t, ∇Q is the gradient vector associated with Q, $\boldsymbol{\theta}_t$ and γ_t are the parameters in iteration t and $\boldsymbol{\theta}_{t+1}$ and γ_{t+1} are the new parameters after iteration t. The choice for the step size η_t is discussed in the next section.

Gradient Descent uses all transactions in the training set to perform an update in the parameters. In our case, only a subset of the number of transactions is taken for the data set, and still the number of transactions is huge (approximately 2.5 million transactions). Also, the amount of features is quite high (1435 variables).

STOCHASTIC GRADIENT DESCENT

In this section, Stochastic Gradient Descent (SGD) is discussed, to maximize the likelihood function. Robbins and Monro [5] were the first to propose stochastic approximation.

In Stochastic Gradient Descent methods, the true gradient $\nabla Q(\boldsymbol{\theta}, \gamma)$ is approximated by the gradient of a single transaction. Every iteration, one transaction is picked at random. For example, the k^{th} transaction:

$$\begin{bmatrix} \boldsymbol{\theta}_{t+1} \\ \gamma_{t+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_t \\ \gamma_t \end{bmatrix} - \eta_t \nabla Q_k(\boldsymbol{\theta}_t, \gamma),$$

where η_t is the step size at iteration t. When the Hessian matrix of Q, which is the matrix with all second partial derivatives of Q, is strictly positive definite ($\mathbf{z}^T Q \mathbf{z} > 0$ for all $\mathbf{z}$), then the best convergence speed is achieved using step sizes $\eta_t \sim t^{-1}$ (as purposed in [5]), e.g. $\eta_t = \dfrac{\eta_0}{1+\xi t}$ where hyperparameters η_0 and ξ need to be chosen. The choice for these hyperparameters is rather sensitive, with respect to the result. We will use an adaptive step size, eventually, and then this problem becomes much less sensitive.

The goal is to find a Stochastic Gradient Decent method, where the regret:

$$R(T) = \sum_{i=1}^{T} Q_i(\boldsymbol{\theta}_i, \gamma) - \inf_{\boldsymbol{\theta} \in \Theta} \sum_{i=1}^{T} Q_i(\boldsymbol{\theta}, \gamma),$$

is small, which means we want to have an upper bound for the regret. The first sum represents the actual loss and the second sum is the loss when using the best static predictors $\boldsymbol{\theta}^*$ and γ^*, after T iterations. Note that, Θ is the set of allowed parameters, hence $\boldsymbol{\theta} \in \Theta$. Small regret means small differences between the result of the SGD method and the result obtained using all the data up to iteration T.

At every time step t, the gradient g_t is obtained. The predictor $\boldsymbol{\theta}_t$ is moved in the opposite direction of g_t (the vector with the derivatives with respect to $\boldsymbol{\theta}$), while assuring that $\boldsymbol{\theta}_{t+1} \in \Theta$ via a projection update. This is called the Greedy Projection, or Lazy projection:

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \Pi_\Theta(\boldsymbol{\theta}_t - \eta_t g_t), \\ &= \arg\min_{\boldsymbol{\theta} \in \Theta} \|\boldsymbol{\theta} - (\boldsymbol{\theta}_t - \eta_t g_t)\|_2^2, \\ &= \arg\min_{\boldsymbol{\theta} \in \Theta} \langle \boldsymbol{\theta} - (\boldsymbol{\theta}_t - \eta_t g_t), \boldsymbol{\theta} - (\boldsymbol{\theta}_t - \eta_t g_t) \rangle. \end{aligned}$$

Since $\gamma \in \mathbb{R}$ has no restriction, we would use:

$$\gamma_{t+1} = \gamma_t - \eta_t \frac{\partial Q}{\partial \gamma}(\boldsymbol{\theta}_t, \gamma_t),$$

to update parameter γ.

Standard stochastic gradient descent methods are oblivious to the characteristics of the observed data. Because of the imbalance of cases to non-cases, we want to consider slightly different stochastic gradient methods. Also, choosing the step size is a sensitive task. By adapting the step size, this problem will decrease drastically.

ADAPTIVE (SUB)GRADIENT METHODS

Standard stochastic gradient descent methods are oblivious to the characteristics of the observed data. So, the algorithm is altered in order to get an adaptive (sub)gradient method that gives frequently occurring features very low learning rates and infrequent features high learning rates. The intuition is that each time an infrequent feature is seen, the learner should “notice" this and use an increased step size. This makes it easier to find and identify predictive but rare features. In our problem, we expect that cases can be detected by noticing predictive but rare features. This is why adaptive (sub)gradient methods are considered.

Subgradients would be needed in case gradients does not exist, due to the use non-differentiable functions. In case of subgradient descent, the choice of the step size needs to be altered. In our objective function, we do not need subgradients.

The AdaGrad algorithm, an adaptive (sub)gradient method, is chosen. In Chapter 3, this algorithm is described. AdaGrad uses an adaptive step size, which makes the problem of choosing the step size η much less sensitive.

2.5. INCORPORATING PRIORS

We would like to incorporate a prior distribution to regulate the search for the parameters. In Section 2.5.1, the Maximum a Posteriori estimate is discussed. It is closely related to the MLE method, but incorporates the prior distribution into the estimation.

The prior probability function describes the uncertainty in the model parameter. Prior knowledge about the model parameter can be incorporated in the estimation. Parameters of the prior probability function are called *hyperparameters*. If no knowledge about the model parameter is available, an uniform distribution can be chosen as prior distribution.

2.5.1. MAXIMUM A POSTERIORI ESTIMATION

The maximum a posteriori (MAP) probability estimate is the mode of the posterior distribution. Based on the observed data, the MAP estimate can be obtained using a point estimate of an unobserved quantity. It is closely related to determining the Maximum Likelihood (ML) estimators.

The Maximum Likelihood estimators are determined, as follows:

$$\begin{aligned}(\boldsymbol{\theta}^*_{ML}, \gamma^*_{ML}) &= \arg\max_{\boldsymbol{\theta},\gamma} \mathscr{L}(\boldsymbol{\theta}, \gamma | (\mathbf{x}_1, s_1), \ldots, (\mathbf{x}_n, s_n)) \\ &= \arg\max_{\boldsymbol{\theta},\gamma} \ell(\boldsymbol{\theta} | (\mathbf{x}_1, s_1), \ldots, (\mathbf{x}_n, s_n)).\end{aligned}$$

The posterior distribution is an updated version of the prior distribution, when the data is considered as well. The prior distributions, written as $f(\boldsymbol{\theta})$ and $g(\gamma)$, reflect the belief about the parameter, before the model is calibrated. The posterior distribution h is equal to:

$$h((\mathbf{x}_1, s_1), \ldots, (\mathbf{x}_n, s_n) | \boldsymbol{\theta}, \gamma) \propto \mathscr{L}(\boldsymbol{\theta}, \gamma | (\mathbf{x}_1, s_1), \ldots, (\mathbf{x}_n, s_n)) f(\boldsymbol{\theta}) g(\gamma). \tag{2.23}$$

In Equation 2.23, the denominator is left out. This constant is not important when the maximum is sought. The MAP estimate is found using the posterior distribution:

$$\begin{aligned}(\boldsymbol{\theta}^*_{MAP}, \gamma^*_{MAP}) &= \arg\max_{\boldsymbol{\theta},\gamma} \mathscr{L}(\boldsymbol{\theta}, \gamma | (\mathbf{x}_1, s_1), \ldots, (\mathbf{x}_n, s_n)) f(\boldsymbol{\theta}) g(\gamma) \\ &= \arg\max_{\boldsymbol{\theta},\gamma} \log\big(\mathscr{L}(\boldsymbol{\theta}, \gamma | (\mathbf{x}_1, s_1), \ldots, (\mathbf{x}_n, s_n)) f(\boldsymbol{\theta}) g(\gamma)\big), \\ &= \arg\max_{\boldsymbol{\theta},\gamma} \ell(\boldsymbol{\theta}, c | (\mathbf{x}_1, s_1), \ldots, (\mathbf{x}_n, s_n)) + \log\big(f(\boldsymbol{\theta})\big) + \log\big(g(\gamma)\big),\end{aligned}$$

In this way, the prior knowledge of the parameters can be incorporated into the prior distribution.

2.5.2. PRIOR ON γ

For the prior distribution on γ, the Beta distribution is chosen, such that a preference for $c \in [0,1]$ is achieved. The reason is that this distribution has support on $[0,1]$. After determining the prior on c, we can determine the prior on γ by using (2.19). Also, by choosing the hyperparameters, we can choose the appropriate shape of the probability density function (pdf) of a Beta distribution. For the choice of the hyperparameters, we asked Rabobank to choose the pdf that fits their belief about c the best.

Definition 2.2. *The probability density function of a Beta distribution, with shape parameters α and β, is:*

$$g(c) = \frac{c^{\alpha-1}(1-c)^{\beta-1}}{\text{Beta}(\alpha,\beta)},$$

where Beta *is the Beta function.*

Definition 2.3. *The Beta function is defined as:*

$$\begin{aligned}\text{Beta}(\alpha,\beta) &= \frac{\Gamma(\alpha)\cdot\Gamma(\beta)}{\Gamma(\alpha+\beta)},\\ &= \frac{(\alpha-1)!\cdot(\beta-1)!}{(\alpha+\beta-1)!},\end{aligned}$$

where $\Gamma(n) = (n-1)!$*, when* n *is a positive integer, is the Gamma function.*

Rabobank chose $\alpha = 15$ and $\beta = 1$. The associated pdf is shown in Figure 2.2. Note that $\text{Beta}(15,1) = \frac{1}{15}$. Hence, the prior on c is:

$$g(c) = 15\cdot c^{14}.$$

Because, c was reparametrized to γ, we have:

$$g(\gamma) = \frac{15}{(1+e^{-\gamma})^{14}}.$$

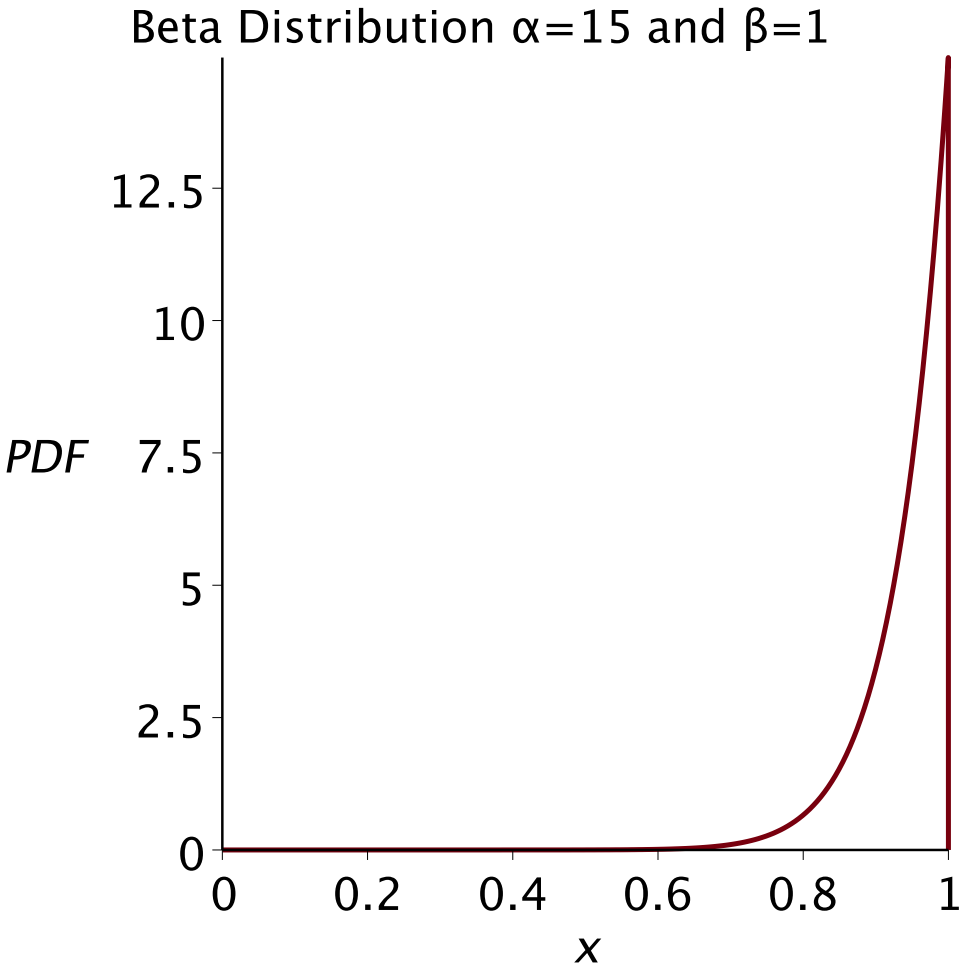

Figure 2.2: Beta prior, shape parameters: $\alpha = 15$ and $\beta = 1$.

2.5.3. PRIOR ON $\boldsymbol{\theta}$

With the choice of the prior on $\boldsymbol{\theta}$, sparsity can be created in the parameters. This prior regulates the complexity of the model. It assures that the most important features get a prominent role in the final parameters, and

the rest of the (less important) features fade out.

SPARSITY-PROMOTING PRIORS

Sparsity in this context refers to a small number of non-zero elements in the model parameter. When a sparsity-promoting prior is chosen, elements of the model parameter can become equal to 0. If such a prior is not used, there could be elements close by 0, but not exactly 0. Elements of the model parameter being 0 has the advantage that they do not need to be incorporated in the calculation. Secondly unimportant variables being excluded is like performing a feature selection, which is desirable when a lot of variables are used.

Definition 2.4. *A Laplacian prior on $\boldsymbol{\theta}$, the Laplace density as prior, looks like*

$$p(\theta_i|\lambda) = \frac{\sqrt{\lambda}}{2} e^{-\lambda|\theta_i|}. \tag{2.24}$$

Lemma 2.5. *The Laplace prior can be rewritten in a hierarchical manner, as done by Figueiredo [6], to avoid problems with the derivative at 0:*

$$\theta_j|\tau_i \sim \mathcal{N}(\theta_j|0,\tau_j) \tag{2.25}$$

$$\tau_j|\lambda \sim \text{Gamma}(\tau_j|1,\lambda/2) = \frac{\lambda}{2} e^{-\lambda\tau_j/2}. \tag{2.26}$$

Each θ_j has a zero-mean Gaussian prior with its own variance τ_j and each τ_j has the same exponential (hyper)prior, with hyperparameter λ.

Proof. Integrate out τ_j to obtain a Laplacian density:

$$p(\theta_j|\lambda) = \int_0^\infty p(\theta_j|\tau_j)p(\tau_j|\lambda)\,\partial\tau_j \tag{2.27}$$

$$= \frac{\sqrt{\lambda}}{2} e^{-\lambda|\theta_j|} \tag{2.28}$$

This shows that the Laplacian prior is equivalent to a two-level hierarchical Bayes model. □

2.5.4. REGULARIZATION

In this section, two versions of the Laplacian prior are purposed, which are equivalent to ℓ_2-regularization and ℓ_1-regularization.

ℓ_2-REGULARIZATION

Using the ℓ_2-norm for regularization, is more easy than using the ℓ_1-norm, since projection on the ℓ_2-ball is more easy. Just by dividing by the ℓ_2-norm, the parameter is projected on the ℓ_2-ball.

Take the following prior distribution:

$$p(\boldsymbol{\theta}|\lambda) \propto \mathrm{e}^{-\lambda\|\boldsymbol{\theta}\|_2^2}.$$

Suppose, the objective is to maximize the sum of log-likelihood and the prior on $\boldsymbol{\theta}$:

$$\arg\max_{\boldsymbol{\theta},c} \left[\ell(\boldsymbol{\theta},c|(\mathbf{x}_1,s_1),\ldots,(\mathbf{x}_n,s_n)) - \lambda\|\boldsymbol{\theta}\|_2^2.\right].$$

This maximization problem is equivalent to:

$$\arg\max_{\boldsymbol{\theta},c} \left[\ell(\boldsymbol{\theta},c|(\mathbf{x}_1,s_1),\ldots,(\mathbf{x}_n,s_n))\right],$$

$$\text{subject to } \|\boldsymbol{\theta}\|_2^2 \leq \lambda,$$

where ℓ_2-regularization is used on $\boldsymbol{\theta}$.

ℓ_1-REGULARIZATION

But using l_2-regularization does not give the sparsity in $\boldsymbol{\theta}$, we are looking for. Using a ℓ_1-norm has two advantages. First, it leads to sparsity in parameter. When the dimension of the parameter is very high, a sparse solution is easier to interpret. Second, using ℓ_1-constraints in some cases leads to better results than ℓ_2-constraints.

Lemma 2.6. *Using Laplacian priors on θ_i, for $i \in \{1,\dots,n\}$, is the same as using ℓ_1-regularization, i.e. maximizing using $\|\boldsymbol{\theta}\|_1 \leq \lambda$.*

Proof. Take the Laplacian prior on θ_i:

$$p(\theta_i|\lambda) = \frac{\sqrt{\lambda}}{2} e^{-\lambda|\theta_j|}.$$

The prior distribution on $\boldsymbol{\theta}$ will be:

$$\begin{aligned} p(\boldsymbol{\theta}|\lambda) &= \prod_{j=1}^{n} p(\theta_j|\lambda) \\ &= \left(\frac{\sqrt{\lambda}}{2}\right)^n e^{-\lambda\|\boldsymbol{\theta}\|_1} \end{aligned}$$

Remember, the objective is to maximize the sum of log-likelihood and the priors on $\boldsymbol{\theta}$ and c:

$$\arg\max_{\boldsymbol{\theta},c} \left[\ell(\boldsymbol{\theta}, c|(\mathbf{x}_1, s_1),\dots,(\mathbf{x}_n, s_n)) - \lambda\|\boldsymbol{\theta}\|_1 + \log(g(c))\right].$$

This maximization problem is equivalent to:

$$\arg\max_{\boldsymbol{\theta},c} \left[\ell(\boldsymbol{\theta}, c|(\mathbf{x}_1, s_1),\dots,(\mathbf{x}_n, s_n)) + \log(g(c))\right],$$

$$\text{subject to } \|\boldsymbol{\theta}\|_1 \leq \lambda,$$

where ℓ_1-regularization is used on $\boldsymbol{\theta}$.

□

2.6. OBJECTIVE FUNCTION

The expression for the log-likelihood function, see expression (2.22), and the prior defined on c, see Section 2.5.2, are combined to form the final objective that we will optimize in the next chapter, using the AdaGrad algorithm.

Minimize:

$$\begin{aligned} Q(\boldsymbol{\theta},\gamma) &= -\sum_{i\in W} \log\left(\frac{f(\boldsymbol{\theta}^T\mathbf{x}_i)}{1+e^{-\gamma}}\right) - \sum_{i\in W^C} \log\left(1 - \frac{f(\boldsymbol{\theta}^T\mathbf{x}_i)}{1+e^{-\gamma}}\right) - \log\big(g(\gamma)\big), \\ &= -\sum_{i\in W} \log\left(\frac{f(\boldsymbol{\theta}^T\mathbf{x}_i)}{1+e^{-\gamma}}\right) - \sum_{i\in W^C} \log\left(1 - \frac{f(\boldsymbol{\theta}^T\mathbf{x}_i)}{1+e^{-\gamma}}\right) - \log\left(\frac{15}{(1+e^{-\gamma})^{14}}\right), \end{aligned} \tag{2.29}$$

subjected to:

$$\|\boldsymbol{\theta}\|_1 \leq \lambda.$$

For this, we will need the gradient of Q. The derivatives of Q with respect to $\boldsymbol{\theta}_j$ is:

$$\frac{\partial Q}{\partial \theta_j} = \sum_{i\in W} \frac{x_{ij}\cdot f'(\boldsymbol{\theta}^T\mathbf{x}_i)}{f(\boldsymbol{\theta}^T\mathbf{x}_i)} - \sum_{i\in W^C} \frac{x_{ij}\cdot f'(\boldsymbol{\theta}^T\mathbf{x}_i)}{1+e^{-\gamma}-f(\boldsymbol{\theta}^T\mathbf{x}_i)},$$

and with respect to γ, will be:

$$\frac{\partial Q}{\partial \gamma} = \left(|W| \cdot \left(1 + e^{-\gamma}\right) - \sum_{i \in W^C} \frac{f(\boldsymbol{\theta}^T \mathbf{x}_i)}{1 - \frac{f(\boldsymbol{\theta}^T \mathbf{x}_i)}{(1+e^{-\gamma})}} \right) \cdot \frac{1}{2 + e^{-\gamma} + e^{\gamma}} - 14 \frac{e^{-\gamma}}{1 + e^{-\gamma}}.$$

3

ADAGRAD ALGORITHM

In Section 2.6, the objective function Q which needs to be minimized, is developed. Also in Chapter 2, adaptive gradient methods are introduced. The AdaGrad algorithm will be used to approximate the parameters $\boldsymbol{\theta}$ and γ. Before we explain the AdaGrad algorithm in detail, first recall the following.

Recall, at every time step t, the gradient g_t is obtained. The predictor $\boldsymbol{\theta}_t$ is moved in the opposite direction of g_t (the vector with the derivatives with respect to $\boldsymbol{\theta}$), while assuring that $\boldsymbol{\theta}_{t+1} \in \Theta$ via a projection update. This is called the Greedy Projection, or Lazy projection:

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \Pi_\Theta(\boldsymbol{\theta}_t - \eta_t g_t), \\ &= \arg\min_{\boldsymbol{\theta}\in\Theta} \|\boldsymbol{\theta} - (\boldsymbol{\theta}_t - \eta_t g_t)\|_2^2, \\ &= \arg\min_{\boldsymbol{\theta}\in\Theta} \langle \boldsymbol{\theta} - (\boldsymbol{\theta}_t - \eta_t g_t), \boldsymbol{\theta} - (\boldsymbol{\theta}_t - \eta_t g_t)\rangle. \end{aligned} \tag{3.1}$$

Since $\gamma \in \mathbb{R}$ has no restriction, we would use:

$$\gamma_{t+1} = \gamma_t - \eta_t \frac{\partial Q}{\partial \gamma}(\boldsymbol{\theta}_t, \gamma_t),$$

to update parameter γ.

3.1. ADAGRAD WITH DIAGONAL MATRICES

The adaptive (sub)gradient method, called AdaGrad, is used. First, we introduce the version with full matrices. In Section 3.1.1, we discuss the version with diagonal matrices, which we will use eventually, because of its easy computation. Both version of AdaGrad are introduced by Duchi, Hazan, and Singer [7]. AdaGrad looks like (3.1) but is slightly different, since AdaGrad uses a different norm to do the projection on Θ.

The AdaGrad algorithm uses the following parameter update:

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \Pi_\Theta^{G_t^{1/2}}\left(\boldsymbol{\theta}_t - \eta G_t^{-1/2} g_t\right), \\ &= \arg\min_{\boldsymbol{\theta}\in\Theta} \left\|\boldsymbol{\theta} - \left(\boldsymbol{\theta}_t - \eta G_t^{-1/2} g_t\right)\right\|_{G_t^{-1/2}}, \\ &= \arg\min_{\boldsymbol{\theta}\in\Theta} \left\langle \boldsymbol{\theta} - \left(\boldsymbol{\theta}_t - \eta G_t^{-1/2} g_t\right), G_t^{-1/2} \cdot \left(\boldsymbol{\theta} - \left(\boldsymbol{\theta}_t - \eta G_t^{-1/2} g_t\right)\right)\right\rangle, \end{aligned} \tag{3.2}$$

where $G_t = \sum_{\tau=1}^t g_\tau g_\tau^T$ is the outer product matrix. The norm used above is called the Mahalanobis norm.

Definition 3.1. *Let $A \in \mathbb{R}^{d\times d}$, where d is the length of $\boldsymbol{\theta}$. Define the Mahalanobis norm on vector $\boldsymbol{\kappa}$:*

$$\begin{aligned} \|\kappa\|_A &= \sqrt{\langle \kappa, A\kappa\rangle}, \\ &= \sqrt{\kappa^T A \kappa}, \end{aligned}$$

Now, the projection used in (3.2) is defined as:

Definition 3.2. *The projection of vector $\boldsymbol{\kappa}$ on Θ according to A is defined as:*

$$\begin{aligned}\Pi_{\Theta}^{A}(\boldsymbol{\kappa}) &= \arg\min_{\boldsymbol{\theta}\in\Theta} \|\boldsymbol{\theta}-\boldsymbol{\kappa}\|_A^2, \\ &= \arg\min_{\boldsymbol{\theta}\in\Theta} \langle \boldsymbol{\theta}-\boldsymbol{\kappa}, A(\boldsymbol{\theta}-\boldsymbol{\kappa})\rangle.\end{aligned}$$

This algorithm can be a computational challenge in high dimensions, since the inverse and root of matrix G_t need to be computed.

3.1.1. DIAGONAL MATRICES

AdaGrad with diagonal matrices is used, which is a special version of (3.2):

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \Pi_{\Theta}^{H_t}\left(\boldsymbol{\theta}_t - \eta H_t^{-1} g_t\right). \\ &= \arg\min_{\boldsymbol{\theta}\in\Theta} \left\|\boldsymbol{\theta} - \left(\boldsymbol{\theta}_t - \eta H_t^{-1} g_t\right)\right\|_{H_t}^2 \\ &= \arg\min_{\boldsymbol{\theta}\in\Theta} \left\langle \boldsymbol{\theta} - \left(\boldsymbol{\theta}_t - \eta H_t^{-1} g_t\right), H_t \cdot \left(\boldsymbol{\theta} - \left(\boldsymbol{\theta}_t - \eta H_t^{-1} g_t\right)\right)\right\rangle\end{aligned} \tag{3.3}$$

where $H_t = \delta I + \text{diag}(G_t)^{1/2}$. In this case, H_t will be diagonal, and both the inverse and root of H_t can be computed easily. Note that δ is a small positive number to ensure that the inverse of H_t exists.

From here on, denote $\tilde{\boldsymbol{\theta}}_{t+1} = \boldsymbol{\theta}_t - \eta H_t^{-1} g_t$, hence:

$$\boldsymbol{\theta}_{t+1} = \arg\min_{\boldsymbol{\theta}\in\Theta} \left\langle \boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}_{t+1}, H_t \cdot \left(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}_{t+1}\right)\right\rangle. \tag{3.4}$$

To update parameter γ, we would use:

$$\gamma_{t+1} = \gamma_t - \eta_t H_t^{-1} \frac{\partial Q}{\partial \gamma}(\boldsymbol{\theta}_t, \gamma_t).$$

Hence, this can be written in two separate steps.

AdaGrad with diagonal matrices:

Step 1: Gradient step: $\tilde{\boldsymbol{\theta}}_{t+1} = \boldsymbol{\theta}_t - \eta H_t^{-1} g_t$

Step 2: Projection step: $\boldsymbol{\theta}_{t+1} = \arg\min_{\boldsymbol{\theta}\in\Theta} \left\langle \boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}_{t+1}, H_t \cdot \left(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}_{t+1}\right)\right\rangle$

Repeat the two steps until all the transactions of the training set are used.

3.2. REGULARIZATION OF $\boldsymbol{\theta}$

In Section 3.1, the AdaGrad algorithm with diagonal matrices is introduced. Now, $\Theta = \{\boldsymbol{\theta} : \|\boldsymbol{\theta}\|_1 \leq \lambda\}$ is used. In Section 2.5.4, we showed that we could use ℓ_1-regularization to obtain the sparse solution we are looking for. Hence:

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \arg\min_{\|\boldsymbol{\theta}\|_1\leq\lambda} \left\langle \boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}_{t+1}, H_t \cdot \left(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}_{t+1}\right)\right\rangle \\ &= \arg\min_{\|\boldsymbol{\theta}\|_1\leq\lambda} \left\{\langle \boldsymbol{\theta}, H_t\boldsymbol{\theta}\rangle - 2\cdot\langle H_t\tilde{\boldsymbol{\theta}}_{t+1}, \boldsymbol{\theta}\rangle + \langle \tilde{\boldsymbol{\theta}}_{t+1}, H_t\tilde{\boldsymbol{\theta}}_{t+1}\rangle\right\} \\ &= \arg\min_{\|\boldsymbol{\theta}\|_1\leq\lambda} \left\{\frac{1}{2}\boldsymbol{\theta}^T H_t\boldsymbol{\theta} - (H_t\tilde{\boldsymbol{\theta}}_{t+1})^T\boldsymbol{\theta} + C\right\}.\end{aligned} \tag{3.5}$$

Note that in the last step, the objective function is divided by 2. Also, $C = 2\cdot\tilde{\boldsymbol{\theta}}_{t+1}^T H_t\tilde{\boldsymbol{\theta}}_{t+1}$ is defined, but does not depend on $\boldsymbol{\theta}$ and can be left out during minimization.

In the projection step of AdaGrad with diagonal matrices, as seen in (3.5), we minimize a quadratic function with $H_t = (\text{diag}(G_t))^{1/2}$ a diagonal matrix with non-negative entries (hence PSD) and we minimize over a convex set (the ℓ_1-cone, which means we have a convex minimization problem. See Appendix B.2 for the exact definition of a convex minimization problem.

The convex minimization problem in (3.5) cannot be solved as easily as projection on the ℓ_2-ball. For the l_2-projection used for AdaGrad with diagonal matrices look in Appendix B.2.2. The problem will be rewritten in such a way, that it will be easier to solve.

Lemma 3.3. *AdaGrad with diagonal matrices:*

$$\boldsymbol{\theta}_{t+1} = \arg\min_{\boldsymbol{\theta}\in\Theta} \langle \boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}_{t+1}, H_t(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}_{t+1})\rangle \tag{3.6}$$

with $\Theta = \{\boldsymbol{\theta} : \|\boldsymbol{\theta}\|_1 \le \lambda\}$, $H_t = \big(\text{diag}(G_t)\big)^{1/2}$ *and* $\tilde{\boldsymbol{\theta}}_{t+1} = \boldsymbol{\theta}_t - \eta H_t^{-1} g_t$, *is equivalent to:*

$$\begin{aligned} &\text{minimize}_{\mathbf{u}} \ \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|_2^2 \\ &\text{subject to} \ \sum_i a_i|u_i| \le \lambda \end{aligned} \tag{3.7}$$

with $\mathbf{u} = H_t^{1/2}\boldsymbol{\theta}$, $\mathbf{v} = H_t^{1/2}\tilde{\boldsymbol{\theta}}_{t+1}$ *and* $a_i = (H_t^{-1/2})_{i,i}$ *is the* i^{th} *diagonal element of* H_t.

Proof. The projection in expression (3.6) can be written as:

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \Pi_{\Theta}^{H_t}\, \tilde{\boldsymbol{\theta}}_{t+1}, \\ &= \arg\min_{\boldsymbol{\theta}\in\Theta} \langle \boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}_{t+1}, H_t(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}_{t+1})\rangle, \\ &= \arg\min_{\boldsymbol{\theta}\in\Theta} \langle H_t^{1/2}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}_{t+1}), H_t^{1/2}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}_{t+1})\rangle, \\ &= \arg\min_{\boldsymbol{\theta}\in\Theta} \big\| H_t^{1/2}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}_{t+1})\big\|_2^2, \\ &= \arg\min_{\boldsymbol{\theta}\in\Theta} \big\| H_t^{1/2}\boldsymbol{\theta} - H_t^{1/2}\tilde{\boldsymbol{\theta}}_{t+1}\big\|_2^2. \end{aligned} \tag{3.8}$$

Divide (3.8) by 2 and make the following substitutions: $\mathbf{u} = H_t^{1/2}\boldsymbol{\theta}$ and $\mathbf{v} = H_t^{1/2}\tilde{\boldsymbol{\theta}}_{t+1}$. Then $\boldsymbol{\theta} \in \Theta \Leftrightarrow \mathbf{u} \in \mathscr{U} = \{\mathbf{u} : \sum_i a_i|u_i| \le \gamma\}$, where $a_i = (H_t^{-1/2})_{i,i}$ are the diagonal elements of H_t. Then the purposed minimization problem comes to light. □

The following algorithm is purposed by Duchi, Hazan, and Singer [7] (page 19). Note that the notation $x \succeq y$, for vector x and y, means $x_i \ge y_i$ for all i. This algorithm can be implemented in $O(d \log d)$ time, where d is the number of elements in $\mathbf{v}$.

Algorithm to solve (3.7)

INPUT: $\mathbf{v} \succeq 0$, $\mathbf{a} \succeq 0$, $\gamma \ge 0$.
if $\sum_i a_i v_i \le c$ **then**
  $\mathbf{u}^* = \mathbf{v}$
end if
SORT v_i/a_i into $\mathbf{w} = [v_{i_j}/a_{i_j}]$ s.t. $v_{i_j}/a_{i_j} \ge v_{i_{j+1}}/a_{i_{j+1}}$
SET $\rho = \max\left\{\rho : \sum_{j=1}^{\rho} a_{i_j} v_{i_j} - \dfrac{v_{i_\rho}}{a_{i_\rho}} \sum_{j=1}^{\rho} a_{i_j}^2 < \gamma\right\}$
SET $\lambda = \dfrac{\sum_{j=1}^{\rho} a_{i_j} v_{i_j} - \gamma}{\sum_{j=1}^{\rho} a_{i_j}^2}$
RETURN $\mathbf{u}^*$ where $\mathbf{u}^* = [v_i - \lambda a_i]_+$.

Then the parameter is updated by $\boldsymbol{\theta}_{t+1} = a_i^T \mathbf{u}^*$.

The MATLAB code of this algorithm is found in Appendix D.1.2.

Proof. This derivation is also given in Appendix E of Duchi, Hazan, and Singer [7]. But I will be more explicit about why each step can be made.

Our original problem (3.7) is symmetric in its objective and constraint. So without loss of generality assume that $\mathbf{v} \succeq 0$ (otherwise, we reverse the sign of each negative component in $\mathbf{v}$, then flip the sign of the corresponding component in the solution vector). This gives:

$$\min_{\mathbf{u}} \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|_2^2 \text{ s.t. } \langle \mathbf{a}, \mathbf{u}\rangle \le \lambda,\ \mathbf{u} \succeq 0.$$

Clearly, if $\langle \mathbf{a}, \mathbf{v}\rangle \le \lambda$, then the optimal $\mathbf{u}^* = \mathbf{v}$. Hence we assume that $\langle \mathbf{a}, \mathbf{v}\rangle > \lambda$. We also assume without loss of generality that $v_i / a_i \ge v_{i+1} / a_{i+1}$ for simplicity of our derivation. Later on we will see that we will have to sort $\mathbf{w}_i = \mathbf{v}_i / \mathbf{a}_i$ in descending order, in order to assure this assumption.

Introducing KKT multipliers $\lambda \in \mathbb{R}_+$ for the constraint that $\langle \mathbf{a}, \mathbf{u}\rangle \le \lambda$ and $\mu \in \mathbb{R}_+^d$ for the positivity constraint on $\mathbf{u}$, we get:

$$\mathscr{L}(\mathbf{u}, \mu_1, \mu_2) = \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|_2^2 + \mu_1(\langle \mathbf{a}, \mathbf{u}\rangle - \lambda) - \langle \mu_2, \mathbf{u}\rangle.$$

Stationarity Computing the gradient of $\mathscr{L}$, we have:

$$\nabla_{\mathbf{u}} \mathscr{L}(\mathbf{u}, \mu_1, \mu_2) = \mathbf{u} - \mathbf{v} + \mu_1 \mathbf{a} - \mu_2.$$

The stationarity condition states that $\nabla_{\mathbf{u}} \mathscr{L}(\mathbf{u}^*, \mu_1^*, \mu_2^*) = 0$. Hence

$$\mathbf{u}^* = \mathbf{v} - \mu_1^* \mathbf{a} + \mu_2^*.$$

Complementary slackness on μ_2 Complementary slackness condition on μ_2 gives us:

$$\langle \mu_2^*, \mathbf{u}^*\rangle = 0.$$

We also use dual feasibility:

$$\mu_2^* \succeq 0,$$

and primal feasibility:

$$\mathbf{u}^* \succeq 0,$$

to see that the solution $\mathbf{u}^*$ satisfies:

$$u_i^* = [v_i - \mu_1^* a_i]_+ = \begin{cases} v_i - \mu_1^* a_i & \text{if } v_i \ge \mu_1^* a_i, \\ 0 & \text{otherwise.} \end{cases}$$

Note that $\mu_2(i) = 0$ if $v_i \ge \lambda^* a_i$, and $\mu_2(i) = -(v_i - \lambda^* a_i)$ otherwise. Then the complementary slackness condition for μ_2, and also primal and dual feasibility, are assured.

Complementary slackness on μ_1 Analogously, the complimentary slackness conditions on μ_1

$$\mu_1^* \left(\langle \mathbf{a}, \mathbf{u}^*\rangle - \lambda\right) = 0,$$

and dual feasibility

$$\mu_1^* \ge 0,$$

gives us that:

$$\mu_1^* = 0 \ \vee \ \langle \mathbf{a}, \mathbf{u}^*\rangle = \lambda.$$

For the optimal solution we must have $\langle \mathbf{a}, \mathbf{u}^*\rangle \le \lambda$. In case of the first condition we have that $\mathbf{u}^* = \mathbf{v}$, but we assumed that $\langle \mathbf{a}, \mathbf{v}\rangle > \lambda$. The first condition cannot hold, hence $\mu_1^* \ne 0$.

The second condition, $\langle \mathbf{a}, \mathbf{u}^*\rangle = \lambda$, gives us:

$$\langle \mathbf{a}, \mathbf{u}^*\rangle = \sum_{i=1}^{d} a_i [v_i - \mu_1^* a_i]_+ = \lambda$$

$$\Leftrightarrow \sum_{i=1}^{d} a_i^2 \left[\frac{v_i}{a_i} - \mu_1^* \right]_+ = \lambda. \tag{3.9}$$

Now, let ρ be the largest index in $\{1, \ldots, d\}$ such that $v_i - \mu_1^* a_i > 0$ for $i \leq \rho$ and $v_i - \mu_1^* a_i \leq 0$ for $i > \rho$. This is possible if we sort $w = v/a$ such that $v_i/a_i \geq v_{i+1}/a_{i+1}$ for all i. Then

$$\sum_{i=1}^{\rho} a_i^2 \left(\frac{v_i}{a_i} - \lambda^* \right) = \lambda,$$

$$\Leftrightarrow \sum_{i=1}^{\rho} a_i v_i - \mu_1^* \sum_{i=1}^{\rho} a_i^2 = \lambda.$$

So, we need to find the largest non-zero integer ρ and take

$$\mu_1^* = \frac{\sum_{i=1}^{\rho} a_i v_i - \lambda}{\sum_{i=1}^{\rho} a_i^2}.$$

□

4

TESTING THE FINAL MODEL

In this chapter, the procedure, on how the model works, and also how the performance of the model is determined, is discussed. First the model is tested on dummy data, for which the intended result is known. Then the model is tested on the data from Rabobank.

4.1. PROCEDURE

First the model is explained in detail. In Figure 4.1 an outline is shown. A data set is chosen and split into a training and test set (see Figure 4.2). Next the complexity of the model is determined by validating λ (see Figure 4.3). Remember the ℓ_1-condition: $\|\boldsymbol{\theta}\|_1 \leq \lambda$. After λ is chosen, the model can be trained used the training set (see Figure 4.4). Then the test set is used to determine the performance (see Figure 4.5).

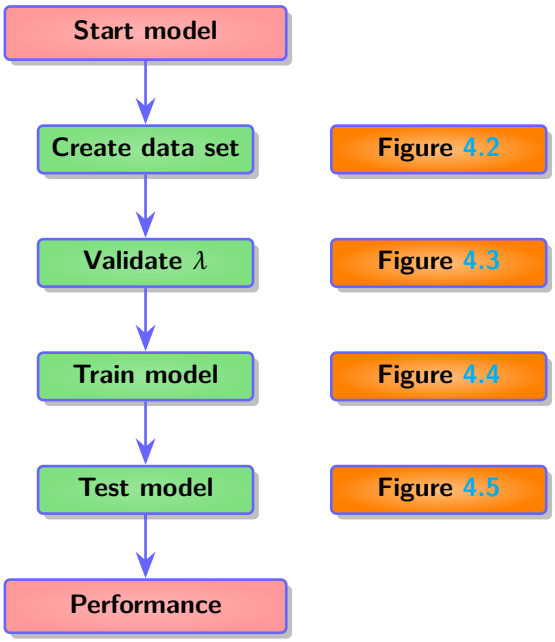

Figure 4.1: How the model works.

For both the training and test set the performance is determined. The performance of the training set is expected to be (slightly) higher than the performance of the test set, since the model probably performance better on examples that are used in training.

4.1.1. CREATE DATA SET

Figure 4.2 illustrates how this phase works. Rabobank delivered the cases and the non-cases separately. The cases and non-cases need to be divided over the training and test set. The total data set contains 805 cases, which are divided over 20 types of cases, and 1382 files of each approximately 1600 non-cases, hence about 2.2 million non-cases. Note the large imbalance between the cases and the non-cases, which is intrinsic to the problem.

In Chawla, Japkowicz, and Kotcz [8], this type of problem is called the *class-imbalance problem* and solutions on data and algorithmic level are discussed. At the data level, several re-sampling techniques (random oversampling, with replacement, random undersampling, etc.) are suggested to solve this imbalance. Also was recommended to use ROC curves instead of evaluation measures as accuracy. The accuracy can be very misleading because of the imbalance. At the algorithmic level, a cost analysis with adjusted costs per class and adjusting the decision threshold are purposed. For this, a cost-matrix is needed. However, it is difficult to determine this cost-matrix. Clever re-sampling and combination methods can do quite more than cost-sensitive learning as they can provide new information or eliminate redundant information for the learning algorithm, as shown in [9], [10], [11], [12] and [13].

In order to decrease the imbalance in the training set, random undersampling of the non-cases and random oversampling with replacement of the cases is used. Random undersampling: First, the data set is taken in which all 805 cases and 13 random files with about 1600 non-cases per file (approximately 21.000 non-cases), which is approximately is 1% of the available files with non-cases, are taken. Random oversampling with replacement: After every n_{samp} non-cases a case, that is selected at random from the pool of cases in the training set, is injected to update the parameters.

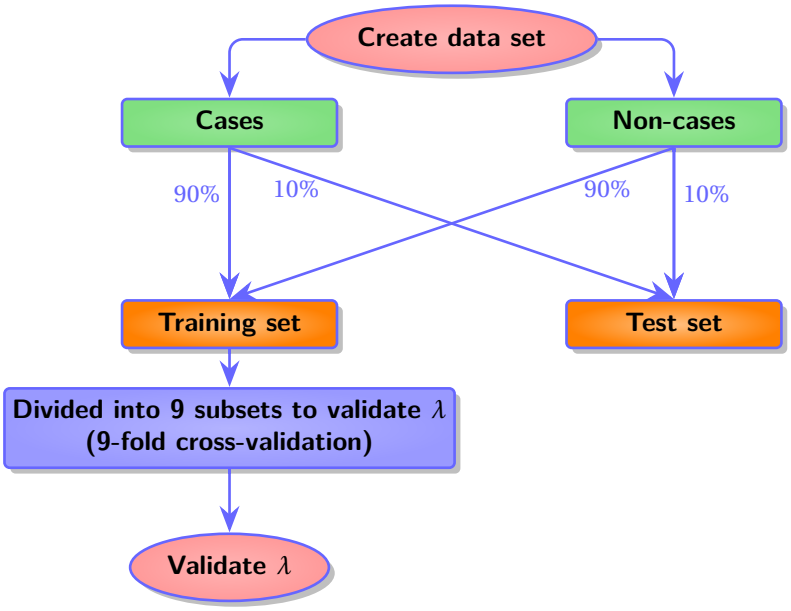

Figure 4.2: How the data set was created.

For the test set 10% of the non-cases and cases are used and the remaining 90% is used to form the training set. The training set is first used to validate λ. To validate λ, stratified k-fold cross-validation is used (with $k = 9$). That is why the training set is split into 9 subsets each containing approximately the same number of cases and non-cases. This is discussed in more detail in 4.1.2.

4.1.2. VALIDATE λ

Figure 4.3 illustrates how this phase works. To validate λ, stratified k-fold cross-validation is used (with $k = 9$). *Stratified* means that in each subset contains approximately the same amount of cases and non-cases, such that each subset contains roughly the same proportions of the two classes. In k-fold cross-validation, the data set is divided in k subsets of equal size. Of the k subsets, one subset is kept for validation, whilst the other

$k-1$ subsets are used for training. The cross-validation is repeated k times, called the *folds*, where each time a different subset is used as validation set. The results of the k folds can be averaged to have one estimation of the performance.

This method is used to validate λ. For several values of λ, the performance on the training set can be estimated, in order to choose the right value for λ. The AUC value on the training set is used to determine which λ gives the best performance. Section 4.1.4 explains how to compute the AUC value.

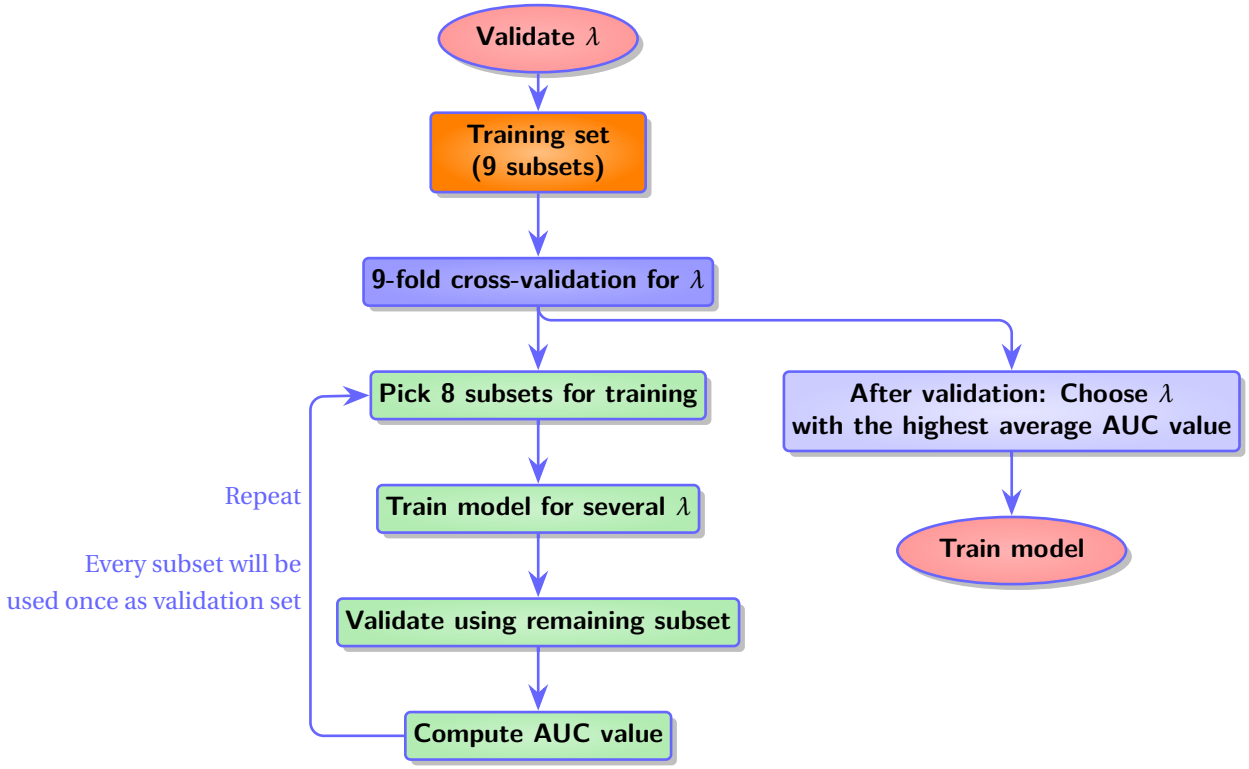

Figure 4.3: How λ was validated.

4.1.3. TRAIN MODEL

Figure 4.4 illustrates how this phase works. The model takes the training set and transaction by transaction the model updates the parameters θ and γ using AdaGrad with diagonal matrices and $\|\boldsymbol{\theta}\|_1 \leq \lambda$.

4.1.4. TEST MODEL

Figure 4.5 illustrates how this phase works. Given the parameters found in the training, the following probability is computed:

$$\hat{y}_i = \mathbb{P}(y_i = 1|\mathbf{x}_i) = f(\boldsymbol{\theta}^T\mathbf{x}_i) = \frac{1}{1+\exp\left(-\boldsymbol{\theta}^T\mathbf{x}_i\right)},$$

which we refer to as the score. Given the scores on the examples of the training and test set and a threshold τ, the Confusion Matrix can be computed and looks like:

		Actual label	
		1	0
Predicted label	1	True Positive (TP)	False Positive (FP)
	0	False Negative (FN)	True Negative (TN)

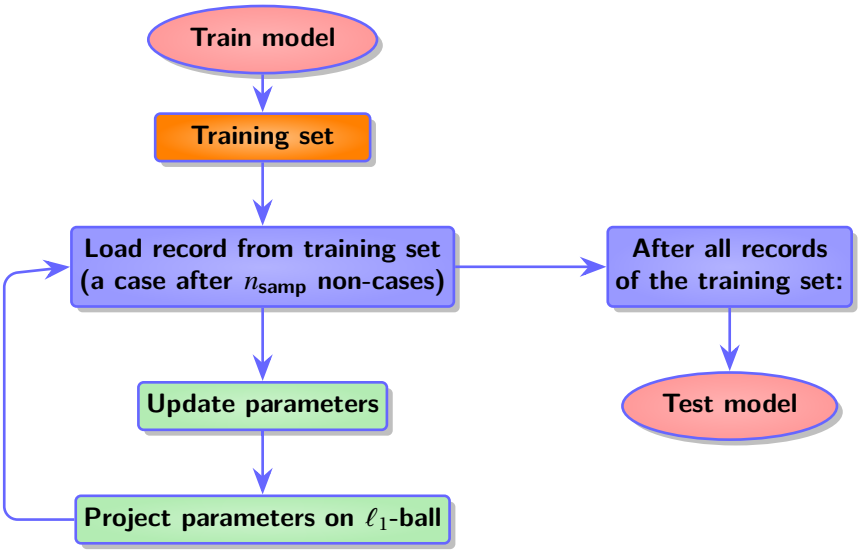

Figure 4.4: How the model was trained.

Examples with a score above the threshold τ will be labelled 1 and the rest will be labelled 0, which can be seen as:

$$\hat{y}_i < \tau \Rightarrow \hat{s}_i(\tau) = 0,$$

$$\hat{y}_i > \tau \Rightarrow \hat{s}_i(\tau) = 1.$$

Here $\hat{s}_i(\tau)$ is the predicted label for record i, given that threshold τ is used to determine the label. The label s and the true label y of every example are known, so the Confusion Matrix can be determined.

The choice of the threshold τ determines the Confusion Matrix and influences the resulting (false) alarms. That is why ROC (Receiver Operating Characteristic) curves are often used. According to Fawcett [14], ROC Curves also have an attractive property: they are insentive to changes in class distribution. If the proportion of cases to non-cases changes in a test set, the ROC curves will not change. To construct a ROC curve the Confusion Matrix is determined for every possible threshold τ between 0 and 1. The ROC curve plots the False Positive Rate (FPR), also called fall-out, against the True Positive Rate (TPR), also called sensitivity, for every possible threshold τ between 0 and 1. The definitions are:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}},$$

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

Then the AUC (Area Under Curve) value, which is the area under the ROC curve, is computed.

Also the specificity (SPC), also called True Negative Rate, and the accuracy (ACC), are computed:

$$\text{SPC} = \frac{\text{TN}}{\text{FP} + \text{TN}} = 1 - \text{FPR},$$

$$\text{ACC} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{TN} + \text{FP}}.$$

Because of the imbalance between the cases and non-cases, the accuracy alone is not a very good number to look at the performance. Note that labelling all the records as “non-cases”, a very good accuracy is achieved. This value is still computed, because in combination with a high AUC value the accuracy is still a valuable number to look at.

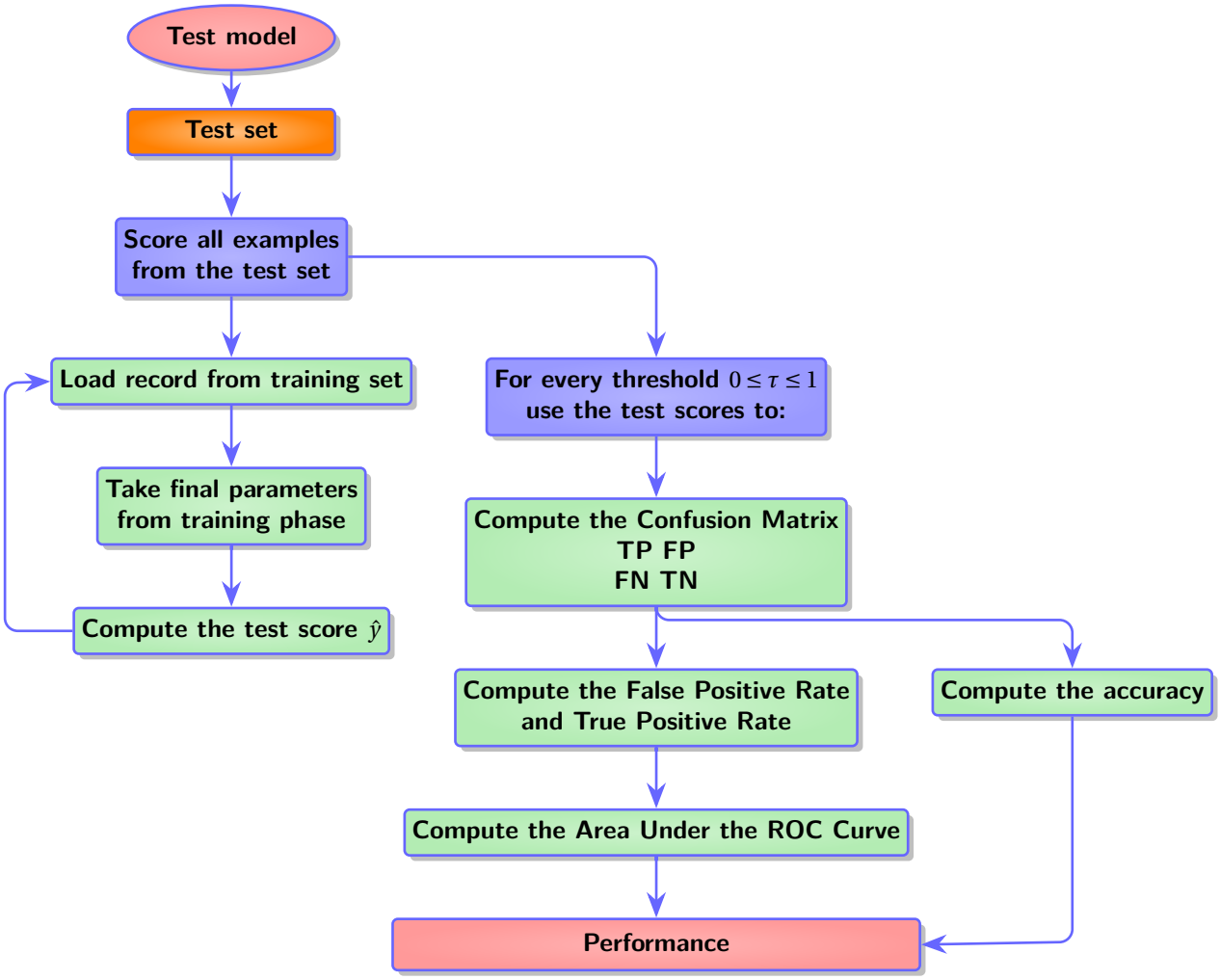

Figure 4.5: How the model was tested.

4.2. TESTING THE MODEL ON DATA GENERATED FROM THE MODEL

First realistic data, which is constructed using the model, is used to ensure that the model works.

The data is constructed as follows. "Transactions" are created containing a number of binary variables. The number of "transactions" created is equal to 100.000. The features are sampled from Bernoulli distributions, $x_i \sim \text{Ber}(\mathbf{p})$. The parameters for these fea are chosen in such a way that the first two variables are the ones determining whether a record is a case. In other words, when x_1 and x_2 are both equal to 1 the probability $\hat{y}_i = \mathbb{P}(y_i = 1|\mathbf{x}_i)$ is close to 1, and for every other combination of x_1 and x_2 the probabilities $\hat{y}_i = \mathbb{P}(y_i = 1|\mathbf{x}_i)$ are low. The combination of $\boldsymbol{\theta}$ and $\mathbf{p}$, where p_j corresponds to the j^{th} feature, determines the number of cases introduced into the data set. We will consider different choices for these.

A large proportion of the cases are labelled, some are left out. Take $s \sim \text{Ber}(c)$, where $c = \mathbb{P}(s = 1|y = 1, \mathbf{x})$.

The data set is split into a training and a test set. First, 90% of the cases are taken for the training set. The remaining 10% of the cases are taken for the test set. The ratio of 5 : 1 (non-cases : cases) is taken for training set. After taken 90% of the case for the training set, 5 times more non-cases are randomly chosen to be in the training set. The remaining non-cases are put in the test set.

4.2.1. EXAMPLES

For every example, 100,000 transactions are used. The number of cases is determined by the choice of $\boldsymbol{\theta}$ and $\mathbf{p}$. The following values are used for the examples that are discussed below:

Example 1 Number of features = 10, $\lambda = 50$, $c = 0.9$,
$\boldsymbol{\theta} = [-15, 10, 10, 0, 0, 0, 0, 0, 0, 0, 0]$, $\mathbf{p} = [0.1, 0.1, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]$

Example 2 Number of features = 10, validate λ (result: $\lambda = 20$), $c = 0.9$,
$\boldsymbol{\theta} = [-15, 10, 10, 0, 0, 0, 0, 0, 0, 0, 0]$, $\mathbf{p} = \left[\sqrt{0.001}, \sqrt{0.001}, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5\right]$

Example 3 Number of features = 1000, $\lambda = 20$, $c = 0.9$,
$\boldsymbol{\theta} = [-15, 10, 10, 0, \ldots, 0]$, $\mathbf{p} = \left[\sqrt{0.001}, \sqrt{0.001}, 0.5, \ldots, 0.5\right]$

Example 4 Number of features = 10, $\lambda = 20$, $c = 0.9$,
$\boldsymbol{\theta} = [-7, 5, 5, 0, 0, 0, 0, 0, 0, 0, 0]$, $\mathbf{p} = \left[\sqrt{0.001}, \sqrt{0.001}, 0.5, \ldots, 0.5\right]$

Example 5 Number of features = 10, $\lambda = 20$, $c_0 = 0.9$, fix $c \in \{0.1, 0.2, \ldots, 1.0\}$ and compute the AUC values,
$\boldsymbol{\theta} = [-15, 10, 10, 0, 0, 0, 0, 0, 0, 0, 0]$, $\mathbf{p} = \left[\sqrt{0.001}, \sqrt{0.001}, 0.5, \ldots, 0.5\right]$

Example 1. *The choice of θ_0, θ_1 and θ_2 makes sure that with high probability the transactions, where $x_1 = 1$ and $x_2 = 1$, are labelled as a case. Namely, when the true values for θ are used to compute the scores $\hat{y} = \mathbb{P}(y = 1|\mathbf{x})$, then:*

$$(x_1, x_2) = (1, 1) \Rightarrow \hat{y} = \frac{1}{1 + e^{-5}} \approx 0.9933$$

$$(x_1, x_2) = (0, 1) \Rightarrow \hat{y} = \frac{1}{1 + e^{5}} \approx 0.0067$$

$$(x_1, x_2) = (1, 0) \Rightarrow \hat{y} = \frac{1}{1 + e^{5}} \approx 0.0067$$

$$(x_1, x_2) = (0, 0) \Rightarrow \hat{y} = \frac{1}{1 + e^{10}} \approx 0.0000$$

Approximately one out of hundred records will be case (since $p_1 \cdot p_2 = 0.1 \cdot 0.1 = 0.01$). Hence about 1000 cases will be in the data set.

Using `glmfit` in MATLAB, which is used to fit a logistic regression model to the data, we got the following parameters $\boldsymbol{\theta}_{GLM}$:

$$\boldsymbol{\theta}_{GLM} = [-12.490, 7.327, 7.625, -0.086, -0.120, -0.192, 0.146, -0.180, -0.248, 0.147, -0.050]$$

The resulting parameters, obtained using the AdaGrad algorithm with diagonal matrices, are:

$\hat{\boldsymbol{\theta}} = [-5.272, 7.238, 7.990, -0.237, -0.309, -0.467, -0.037, 0.285, -0.297, 0.169, -0.170]$, $\hat{\gamma} = 13.6355$ *and* $\hat{c} = 1.0000$.

Figure 4.6 shows the ROC curves for the training set and the test set using both $\boldsymbol{\theta}_{GLM}$ and $\hat{\boldsymbol{\theta}}$. The performance of the result of the model is better than the performance using the result of `glmfit`.

For threshold $\tau = 0.5$:

$SPC_{train} = 0.8166$, $ACC_{train} = 0.8185$, $SPC_{test} = 0.8210$, $ACC_{test} = 0.8228$.

Confusion Matrix for training

TP: 899	FP: 16338
FN: 0	TN: 72762

Confusion Matrix for test

TP: 100	FP: 1772
FN: 0	TN: 8129

The final parameters $\hat{\boldsymbol{\theta}}$ look like the parameters $\boldsymbol{\theta}_{GLM}$, which is positive. Only the intercept (the first parameter is different). Both the Area Under the ROC Curve and accuracy are very high. The performance of the model using the parameters $\boldsymbol{\theta}_{GLM}$, instead of the final parameters of the model, would drastically decrease the AUC values.

Choosing the threshold τ:

Now, the threshold was chosen naively: $\tau = 0.5$. How would we eventually choose the threshold? When the Confusion Matrix is computed for the threshold $\tau = 0.5$, we see that the number of False Positives is approximately 17 times larger than the True Positives. That means that hopefully after every 18 alarms, there is one correct alarm. When looking at the results on the test set, we see that 1872 alarms were given by the model, of which 100 correct alarms (TP) and 1772 false alarms (FP). Rabobank has the capacity to check about 100 alarms per day. This is why choosing the threshold is important when the model needs to give an alarm. Tuning the threshold would prevent that this many alarms would come up. Choose the lowest threshold such that $TP + FP < 100$. Another solution would be to deliver the 100 highest scores to Rabobank.

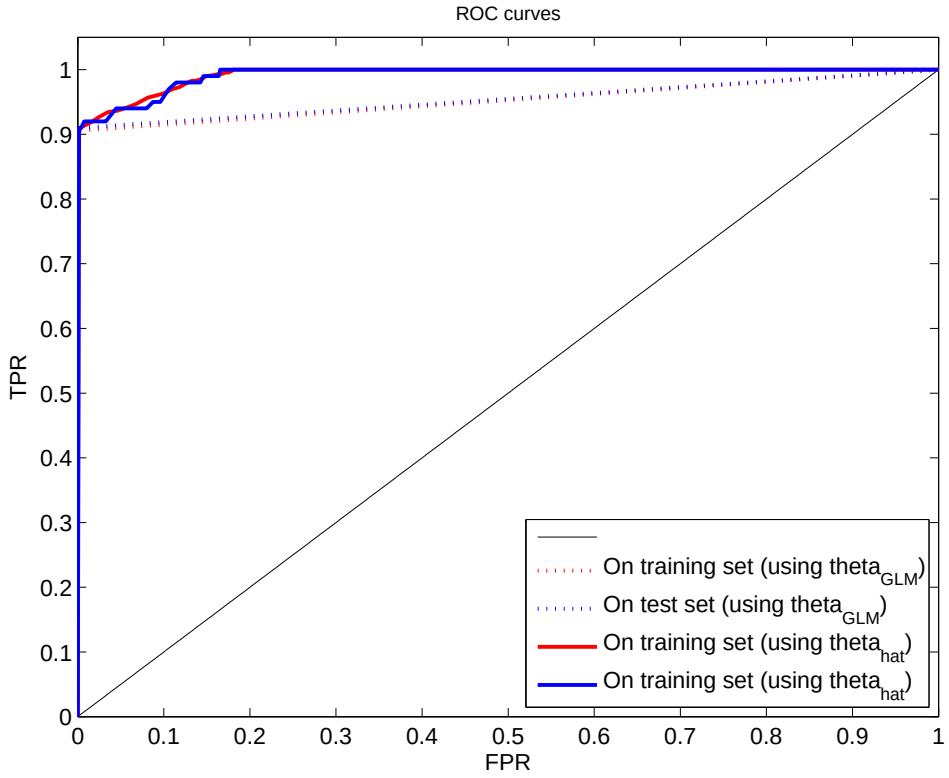

Figure 4.6: ROC curves for Example 1, $AUC_{train} = 0.9920$ (solid red), $AUC_{test} = 0.9914$ (solid blue), $AUC_{train} = 0.9524$ (dotted red), $AUC_{test} = 0.9544$ (dotted blue).

Example 2. *Now cross-validation on λ is used, where $\lambda \in \{5,10,15,20,25,30,50\}$. The highest average AUC value was achieved using $\lambda = 20$, so $\lambda = 20$ was chosen. The probability that a case occurs decreases, because* **p** *is changed. Approximately one out of thousand records will be case (since $p_1 \cdot p_2 = \sqrt{0.001} \cdot \sqrt{0.001} = 0.001$). Hence, about 100 cases will be in the data set.*

Using `glmfit` *in MATLAB, which is used to fit a logistic regression model to the data, we got the following parameters $\boldsymbol{\theta}_{GLM}$:*

$$\boldsymbol{\theta}_{GLM} = [-11.598, 6.605, 6.704, 0.254, 0.575, -0.511, -0.101, -0.138, 0.009, -0.041, -0.098]$$

The resulting parameters, obtained using the AdaGrad algorithm with diagonal matrices, are:

$\hat{\boldsymbol{\theta}} = [-3.636, 6.598, 7.070, 0.667, 0.885, -0.495, -0.217, -0.026, -0.282, 0.013, 0.083]$, $\hat{\gamma} = 13.6398$ *and* $\hat{c} = 1.0000$.

Figure 4.7 shows the ROC curves for the training set and the test set.

For threshold $\tau = 0.5$*:* $SPC_{train} = 0.9373$, $ACC_{train} = 0.9374$, $SPC_{test} = 0.9407$, $ACC_{test} = 0.9408$.

Confusion Matrix for training

$TP: 114$	$FP: 5636$
$FN: 0$	$TN: 84249$

Confusion Matrix for test

$TP: 13$	$FP: 592$
$FN: 0$	$TN: 9396$

The final parameters $\hat{\boldsymbol{\theta}}$ look like the parameters $\boldsymbol{\theta}_{GLM}$, which is positive. Again, the first element of $\hat{\boldsymbol{\theta}}$ is different from the GLM approximation. The performance is this time also better (not shown in the figure). Both the Area Under the ROC Curve and accuracy are very high, but slightly worse than in Example 1. Considering there are less cases to train and test on, this difference is expected.

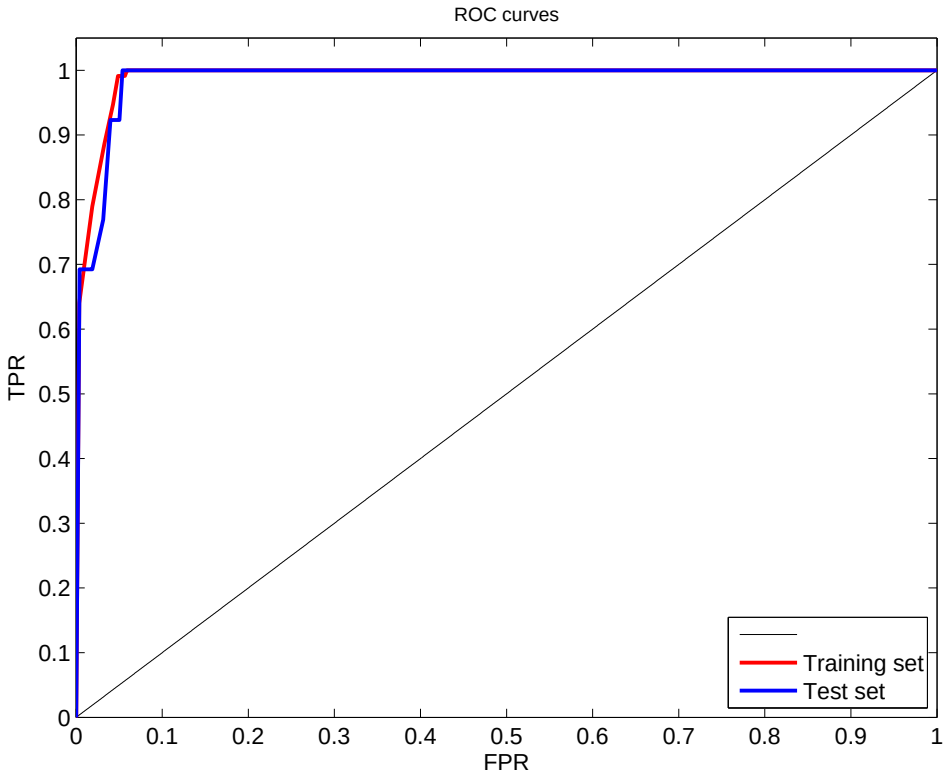

Figure 4.7: ROC curves for Example 2, $AUC_{train} = 0.9899$, $AUC_{test} = 0.9871$.

Example 3. *Now, 1000 variables are used, for which the first two are still the ones determining the cases. The question is whether the model suffers from all the noise which is added. Only part of the resulting parameters is displayed below. The resulting parameters, obtained using the AdaGrad algorithm with diagonal matrices, are:*

$$\hat{\boldsymbol{\theta}} = [-1.064, 3.597, 4.204, -0.007, -0.020, -0.012, -0.007, -0.003, -0.006, -0.014, -0.006, -0.001, 0.015, \ldots$$

$$\ldots, 0.010, -0.014, 0.009, -0.005, 0.000, 0.012, -0.013, 0.006, 0.001, -0.002, -0.006, -0.019]$$

Figure 4.8 shows the ROC curves for the training set and the test set.

For threshold $\tau = 0.5$*:*

$SPC_{train} = 0.9383$, $ACC_{train} = 0.9383$, $SPC_{test} = 0.9378$, $ACC_{test} = 0.9379$.

Confusion Matrix for training

TP: 120	FP: 5549
FN: 0	TN: 84330

Confusion Matrix for test

TP: 14	FP: 621
FN: 0	TN: 9366

The results are similar to Example 2, also slightly worse than Example 3. But not very alarming.

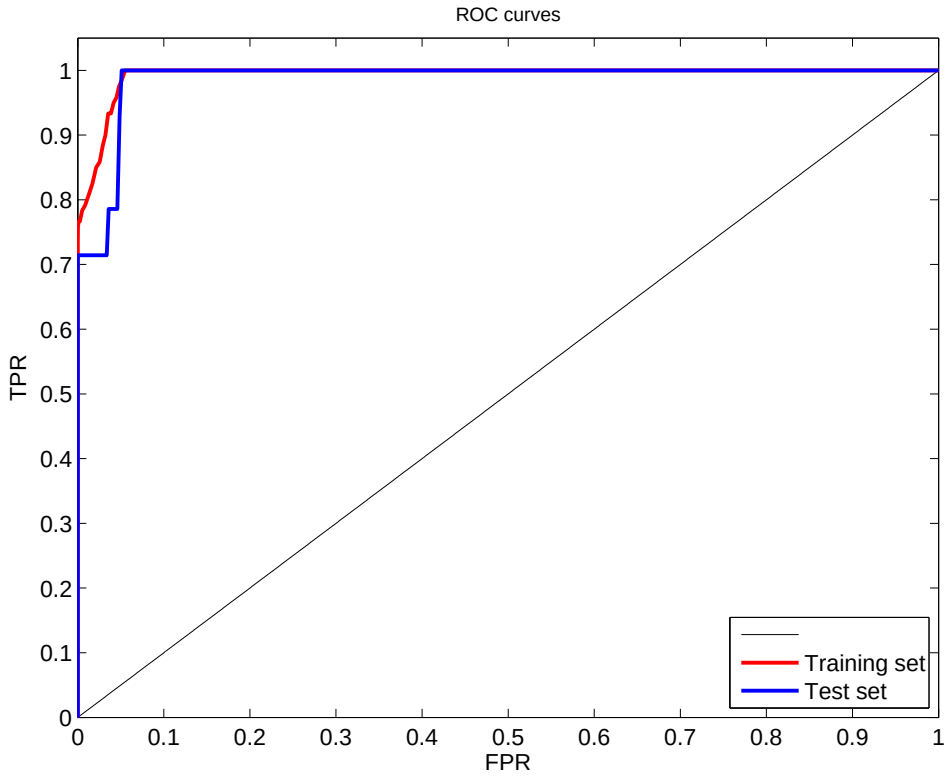

Figure 4.8: ROC curves for Example 3, $\text{AUC}_{\text{train}} = 0.9933$, $\text{AUC}_{\text{test}} = 0.9872$.

Example 4. *Use the same settings as in Example 2, but take* $\boldsymbol{\theta} = [-7, 5, 5, 0, 0, 0, 0, 0, 0, 0, 0]$ *instead.*

The choice of θ_0*,* θ_1 *and* θ_2 *makes sure that with high probability the records where* $x_1 = 1$ *and* $x_2 = 1$*, are labelled as a case. But, this time the chosen parameters also let some false alarms slip into the cases. Namely, when the true values for* θ *are used to compute the scores* $\hat{y} = \mathbb{P}(y = 1|\mathbf{x})$*, then:*

$$(x_1, x_2) = (1,1) \Rightarrow \hat{y} = \frac{1}{1+e^{-3}} \approx 0.9526$$

$$(x_1, x_2) = (0,1) \Rightarrow \hat{y} = \frac{1}{1+e^{2}} \approx 0.1192$$

$$(x_1, x_2) = (1,0) \Rightarrow \hat{y} = \frac{1}{1+e^{2}} \approx 0.1192$$

$$(x_1, x_2) = (0,0) \Rightarrow \hat{y} = \frac{1}{1+e^{7}} \approx 0.0009$$

The resulting parameters, obtained using the AdaGrad algorithm with diagonal matrices, are:

$\hat{\boldsymbol{\theta}} = [-2.511, 4.857, 5.028, 0.275, 0.318, 0.008, 0.015, -0.011, 0.139, 0.036, -0.117]$, $\hat{\gamma} = 13.4667$ *and* $\hat{c} = 1.0000$.

Figure 4.9 shows the ROC curves for the training set and the test set.

For $\tau = 0.5$*:*

$SPC_{train} = 0.9450$, $ACC_{train} = 0.9447$, $SPC_{test} = 0.9437$, $ACC_{test} = 0.9438$.

Confusion Matrix for training

TP: 689	FP: 4904
FN: 73	TN: 84333

Confusion Matrix for test

TP: 81	FP: 558
FN: 4	TN: 9358

The AUC values are slightly lower than the previous examples, because the altering of $\boldsymbol{\theta}$ *caused some noise in the cases.*

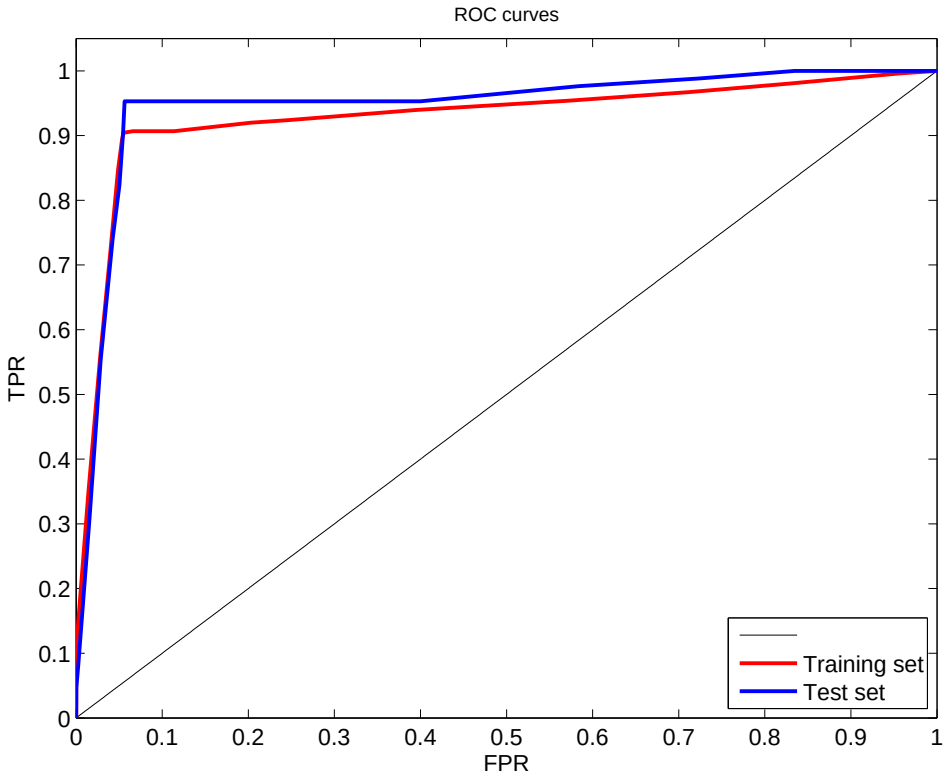

Figure 4.9: ROC curves for Example 4, $\text{AUC}_{\text{train}} = 0.9282$, $\text{AUC}_{\text{test}} = 0.9467$.

Example 5. *Now, take different values for c and don't use the estimate for c provided by the model. The question is whether that changes the performance. The same settings as in Example 2 are used.*

Figure 4.10 shows the AUC values for the training set and the test set, computed using the model, and the AUC values for fixed values of c. Note that the differences between the AUC values are not large, for large values of c. For low values of c, the AUC value on the test set is increased for a fixed c.Also, the performance on the test set in Example 2 is comparable to the performance here. We do not exactly know why this behaviour is seen. Probably, it has to do with the gradient (especially the derivative with respect to θ_j) in which c determines the influence of the unlabelled transactions.

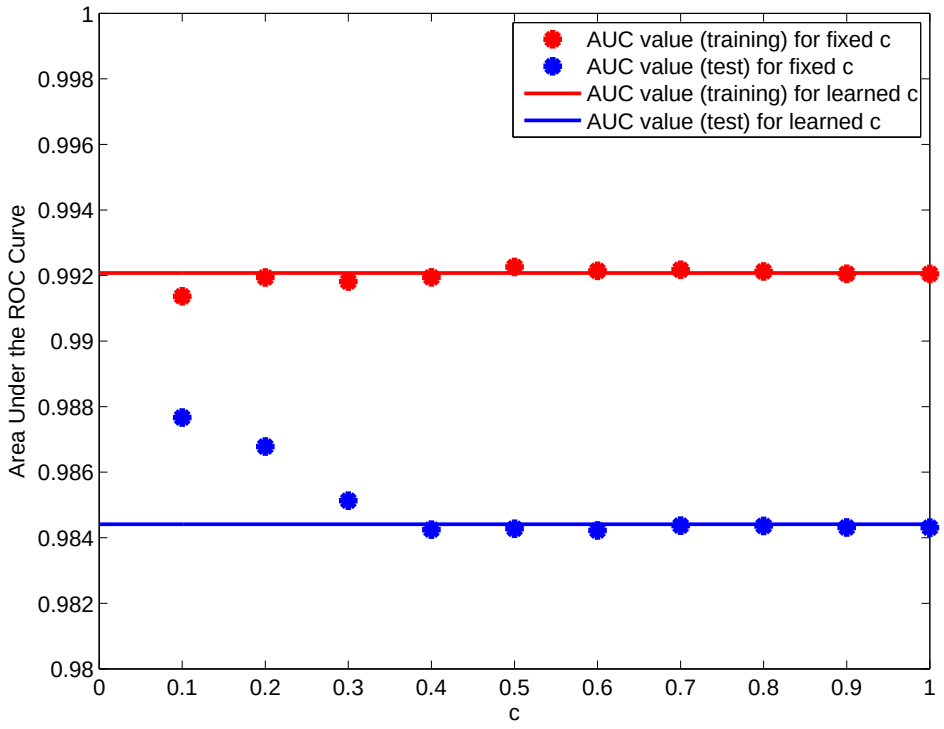

Figure 4.10: AUC values for Example 5.

4.3. TESTING THE MODEL ON RABOBANK DATA

The data set provided by Rabobank is used to test the model. The data consists of two types of features, categorical and numerical features. Categorical features are converted to a binary setting. Numerical features are normalized. Amongst those features, constant features and exact copies of other features were detected and left out. See Chapter 1 for more details.

The procedure for choosing λ, which is used to limit the ℓ_1-norm ($\|\boldsymbol{\theta}\|_1 \leq \lambda$), was discussed in Section 4.1.2. The value of λ determines the complexity of the model. If λ is too small the parameter cannot reflect the complexity of the model and if λ is too high the parameter will not be sparse.

Again, the AUC value and the accuracy for both the training and test sets are computed.

Lastly, the estimation of c is discussed. The model is constructed to have a value $c = \mathbb{P}(s = 1|\mathbf{x}, y = 1)$ incorporated. The question is whether that was really necessary and whether learning c while running the model (instead of choosing a constant c) was needed. The log-likelihood function looked almost as a log-likelihood of a simple regression model. Note that, for $c = 1$ the log-likelihoods would be equal:

$$\ell(\boldsymbol{\theta}, 1) = \log \mathscr{L}(\boldsymbol{\theta}, 1) = \sum_{i \in W} \log\big(f(\boldsymbol{\theta}^T \mathbf{x}_i)\big) + \sum_{i \in W^C} \log\big(1 - f(\boldsymbol{\theta}^T \mathbf{x}_i)\big),$$

During the training the value of c is updated, using the gradient of the log-likelihood and step size. The question is whether it is was needed to use another model than a simple regression model. In order to compare the result, the model is run several times, where constant values for c are used, and the results are compared to when c was estimated by the model.

4.3.1. TEST RESULTS

Figure 4.11 shows the result, after running the model on the Rabobank data. The performance is really bad.

For this, the following settings are used:

$\lambda = 5$ (validated choosing from $[5, 10, 15, 20, 25]$), $\eta = 1$, $\delta = 0.01$, $n_{\text{samp}} = 22$, number of features = 1435.

1% of the non-cases used (13 files of each approximately 1600 transactions) : 11 files used for training, 2 files used for testing.

805 cases: 724 cases used for training, 81 cases used for testing.

SUBSET OF FEATURES

The performance of the model using all 1435 features was not so good. We would like to see whether the model works, when features are used which are more distinctive than others. That's why, we take the top-75 features. How the features were selected, was discussed in Section 1.1.6. Figure 4.12 shows the result. The performance is much better.

$\lambda = 27$, $\eta = 1$, $\delta = 0.01$, $n_{\text{samp}} = 22$, number of features = 75.

1% of the non-cases used (13 files of each approximately 1600 transactions) : 9 files used for training, 4 files used for testing.

805 cases: 563 cases used for training, 242 cases used for testing.

Figure 4.13 and 4.14 show the resulting scores computed using the model. The first shows the scores for all the cases in the test set. And the second shows the scores for all the non-cases in the test set. Note that, using a threshold, e.g. $\tau = 0.5$, would mean that all the alarmed transactions were indeed cases.

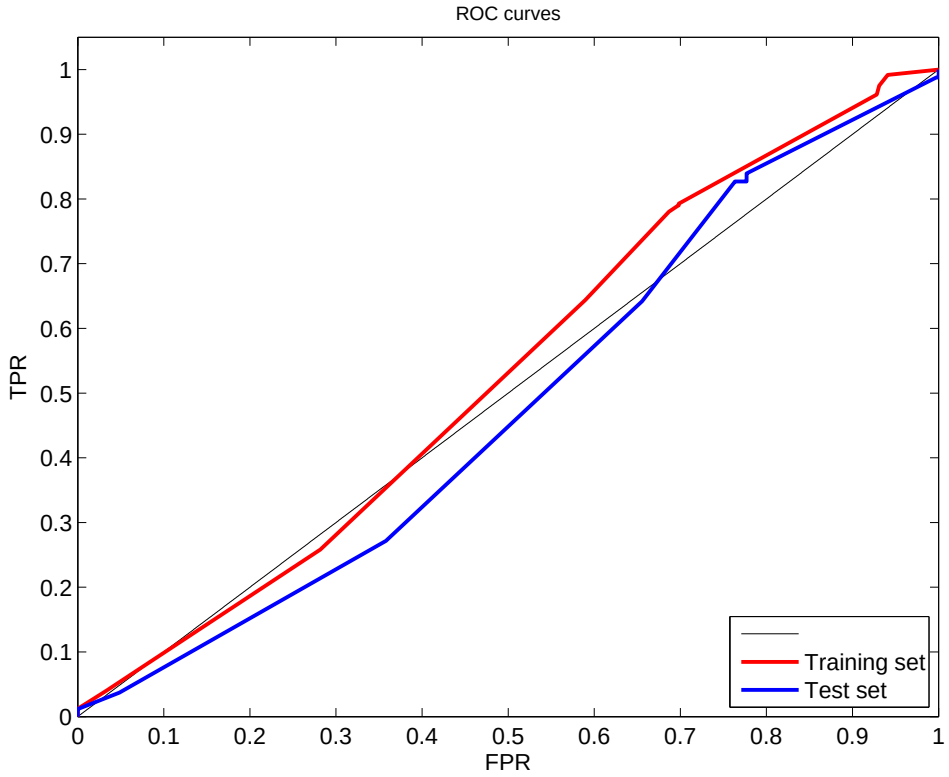

Figure 4.11: ROC curves, $AUC_{train} = 0.5279$, $AUC_{test} = 0.4792$.

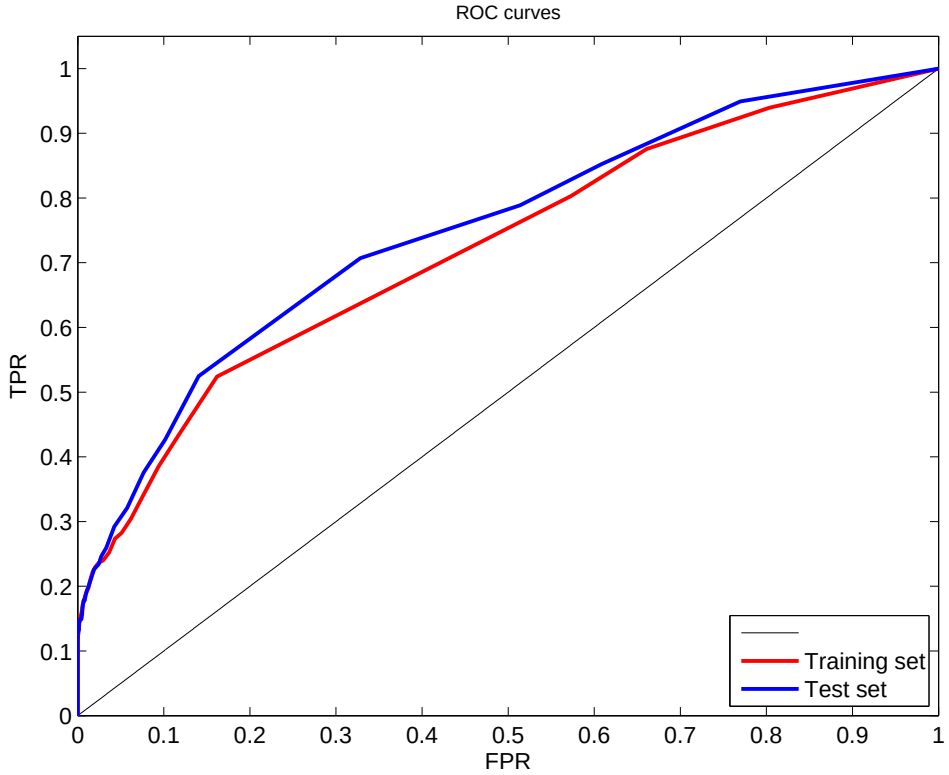

Figure 4.12: ROC curves, $AUC_{train} = 0.7234$, $AUC_{test} = 0.7510$.

Figure 4.13: All the scores on the cases, when the top-75 features are used.

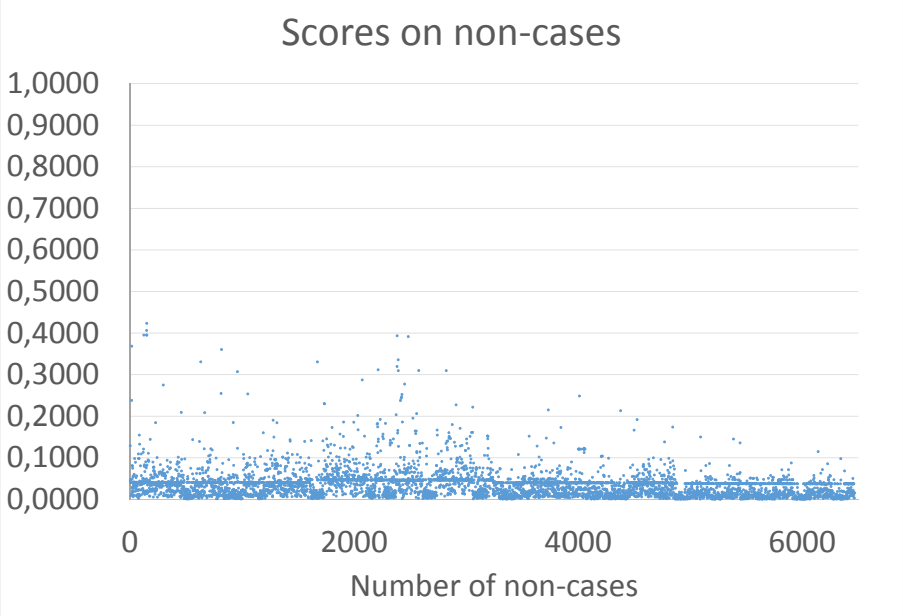

Figure 4.14: All the scores on the non-cases, when the top-75 features are used.

4.3.2. TAKING A CONSTANT VALUE FOR c

The model looks like a standard regression model, but is slightly different because of the appearance of c. Note that for $c = 1$, the model is precisely a (simple) regression model. We want to see whether it was needed to incorporate the estimation of c. So a data set is taken, split it into a training and test set, λ is chosen and the model is used without estimation of c. Instead a couple of constant values for c are used to see what the performance of the model is. Figure 4.15 shows the result. From this, we see that if c is fixed and chosen really low, then the performance on the training and test set is higher. For higher values of c, we see that fixing c or estimating c is similar in performance.

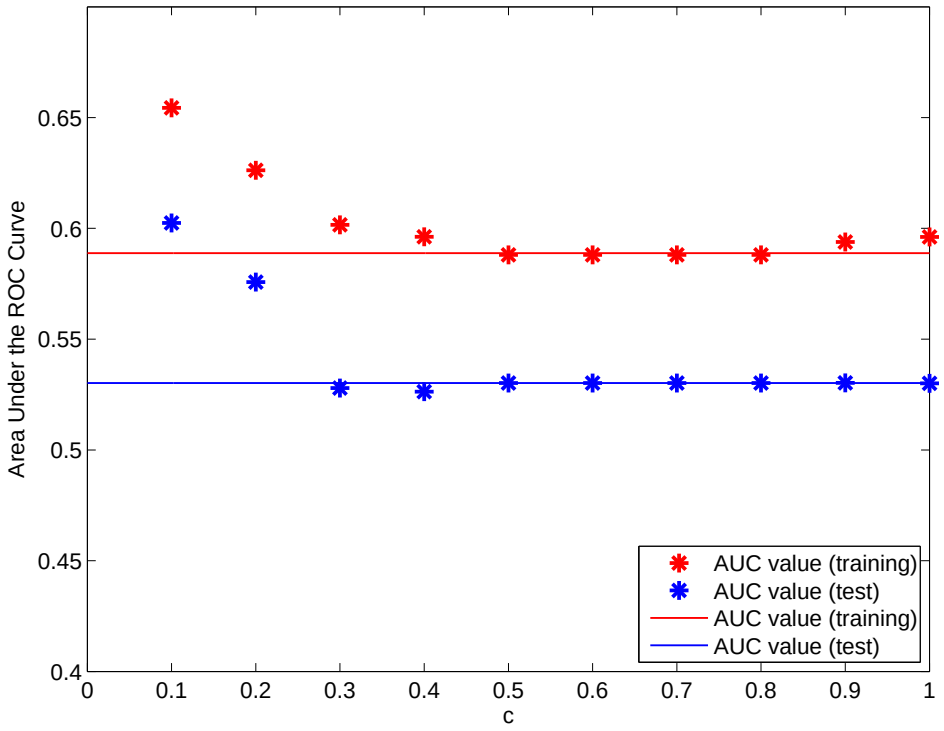

Figure 4.15: Red line and blue line indicate AUC values when c is estimated by the model. The stars indicate the AUC values when c is fixed.

5

CONCLUSIONS

In this chapter, the main results and the conclusions are discussed. The main conclusion is that the Adagrad algorithm with diagonal matrices has the potential to discover cases, when a subset of the features is taken. On the complete set of features, we wanted to utilize the sparsity in $\boldsymbol{\theta}$ to work as feature selection, where unimportant features eventually get filtered out and important features are kept. This goal was not achieved.

Sparsity in $\boldsymbol{\theta}$. By restricting $\boldsymbol{\theta}$ with a ℓ_1-norm ($\|\boldsymbol{\theta}\|_1 \leq \lambda$), we hoped to achieve that only rare but important features would determine the final score, and thereby performance could be increased. In other words, sparsity in $\boldsymbol{\theta}$ would work like feature selection. As we have seen, this does not work as we had hoped. Probably, the choosing of λ is more difficult than we anticipated. On forehand, a set of possible λ's are selected to choose from using cross-validation. Reducing the number of features (in which the top-75 features is used) showed that in that case the performance was not to bad.

Choosing an appropriate λ is important, because λ determines the complexity of the model. If λ is to low, important features could be excluded, and if λ is too high the model takes into account unimportant features. Stratified k-fold cross-validation is used to determine λ. Before this cross-validation, a couple of values for λ needed to be chosen, for which the performance is tested. It is hard to choose the right values for λ.

Estimating c. In the model, c was incorporated to suggest that there are cases missing, which can be found in the unlabelled transactions. The question is whether estimating c is necessary. We could choose a fixed $0 < c < 1$ throughout the iterations and see what the result would be. If $c = 1$, then the suggestion would be that all the cases are labelled. Also, it would suggest that simple logistic regression, using the transactions and the labels, would suffice to determine the parameters $\boldsymbol{\theta}$.

The advantage of our model that it can be used on large data sets and also is usable in case of a constantly changing data set (as in online learning). The model can be used to score an incoming transaction and then this transaction is used to update the model. That is why, we chose this model in the first place. Also, since our model is trained using only one transaction at one time, the amount of memory needed to stored the data is not that much.

6

DISCUSSION

In this chapter, the improvements and remarks are discussed that can be made about this thesis project.

In the beginning, we started building a model that incorporates c into the estimation of parameters $\boldsymbol{\theta}$. Both $\boldsymbol{\theta}$ and c are estimated using an Adaptive Gradient Descent method in combination with a ℓ_1-norm restriction on $\boldsymbol{\theta}$, in the hope that we could use them to compute scores using either $\mathbb{P}(y = 1|\mathbf{x})$ or $\mathbb{P}(y = 1|\mathbf{x}, s = 0)$. We say in both a generated data set and the data set of Rabobank that choosing a low value for c would increase the performance of the model. Possible explanation for this could be that a low c decreases the influence of unlabelled examples on the gradient.

The question arises when discussing Assumption 2 (see expression 2.6), which is: 'Is this assumption reasonable?' or even 'Is this assumption realistic?'. The answers are 'yes' and 'probably not' in the data set of Rabobank. One could say that the probability that Rabobank missed a case, after some fixed time, is always approximately the same. On the other hand, the cases are divided in 20 different groups of different case types. But, in our analysis we used all the cases as one group. To make the model more realistic, the value of c could vary for each case type. Intuitively, it is possible that some case type can be labelled easily and the other case type is harder to identify. Therefore, the values of c would be different. The likelihood function could in this case be split in more groups of labelled examples.

The data set, that was given by Rabobank, contains only labelled cases. There is no case that is left unlabelled. So, we separated the data set into a training set and a test set. On the training set, we used all labelled and unlabelled transactions to determine the parameters θ and c. But in the training set, there was no unlabelled case, so it is not weird that we got $c = 1$ after running the model. On the test set, we tested the model. All the transactions got a score and were then compare to the actual label. Now, looking back at the procedure we described, I think that the following could make the test more reliable. Take all the cases (y=1) and take $s \sim \text{Ber}(c)$ for some $0 < c < 1$. Use the labelled transactions and unlabelled transactions to estimate θ. Now, score all the unlabelled transactions and then compare this to actual label. This procedure looks more like the real situation, where transactions are always unlabelled at first, since they only just took place. Our model gives a score on the transaction and then uses it to update the model. Later on, a employee can decide to label the transaction as a case, and is added to the labelled transactions. For this procedure, there is no need to split up the complete data set into two sets, but only the cases need to be split (labelled and unlabelled).

AdaGrad with diagonal matrices uses a correction factor $(H_t)_j$, for the j^{th} feature, to correct the step size η (also called learning rate). This correction factor makes sure that frequent unimportant features get cancelled out and that rare but important features get important. Also, we used oversampling of the cases and undersampling of the non-cases. Rare features that would predict cases will also fade out since the cases are presented much more than they should. Also, the undersampling of non-cases causes unimportant features to stay longer important. In a realistic situation, all transactions are scored and used to update the model, even before they were labelled as case. After evaluation of an employee, some transactions would be considered as case. This group of cases needs to be presented to the model, where this time the label is 1. The question is whether a case needs to be presented more times or only once to the model. After a couple updates of the model using non-cases, the model could be forgotten which features were important to look

at. So, the question remains what the best frequency is to present cases to the model. For our analysis, we injected a case after each 20 unlabelled transactions (acquired by validation).

PU–learning is a hard problem. There are other methods purposed to solve this problem. For example, in Liu, Dai, Li, Lee, and Yu [15], they try to built a set of negative examples and choose a classifier, from a set of classifiers, that predicts both the negative and positive examples correctly. In Appendix A.4, this method is discussed in more detail.

A

PREVIOUS WORK

In this chapter, previous work done in the field of machine learning is discussed. In particular, some background is given about supervised learning, online learning and PU–learning.

A.1. MACHINE LEARNING

Machine learning is a subfield of pattern recognition. Also data mining is considered closely related to these fields. Machine learning, pattern recognition and data mining are subfields of computer science and artificial intelligence, and also are strongly dependent on statistics and optimization. In machine learning data is used in order to understand the underlying system. We try to reveal the underlying pattern. That's why machine learning is a form of pattern recognition. Within machine learning we have supervised and unsupervised learning. The difference lies in the availability of labels in the data. In unsupervised learning we don't have these labels.

A.2. SUPERVISED LEARNING

Supervised learning is the machine learning task where we have labelled training data. The training data consists of training examples. These training examples could be positive and negative examples. Each example is a pair of the input (vector) and the expected output (label). The supervised learning algorithm uses the training data to produce a function that can be used for mapping the new records to a label. In the optimal scenario the supervised learning algorithm produces a function that maps all the new records to the correct label.

A.3. ONLINE LEARNING

Traditionally machine learning uses a static training set to train the model. With static, we mean that during training the training set does not change. It is also referred to as batch learning, or offline learning. Also the underlying distribution from which the training examples are taken does not change over time.

In our case, we have to assume that the underlying distribution changes over time, because anomalies change through time. That is where online learning comes in. Online learning is used when data becomes available through time. An advantage, when a new training example presents itself, is that the mapping to create the output is simply updated. In batch algorithms the whole data set needs to be represented to train the model. This can save time and also the memory needed is much less than when a batch algorithm is used and ideally the memory needed is constant.

Online machine learning is a subfield of supervised machine learning where the model learns one example at a time. The algorithm consists of three sub steps. First an example is presented to the algorithm. Then the model predicts the label of the example. In the third step we get the true label of the example. This

information is then used to improve the model. The model is updated. The goal is to predict labels that are close to the true labels.

A.4. PU-LEARNING

PU (Positive Unlabelled) learning, a special form of supervised learning, has only labels on interesting cases. For the rest of the records we don't whether it is uninterested to look at. They are called unlabelled.

In Elkan and Noto [4], PU–learning is explained and c is introduced as the approximation of $\mathbb{P}(s = 1|\mathbf{x}, y = 1)$, which is independent of the features $\mathbf{x}$. The data used contains positive examples (which are labelled) and unlabelled examples for which is unknown whether it is positive or negative. Hence, the label s is known, but the question is whether all positive examples are indeed labelled. They propose to use a classical learning classifier using the labels s to approximate the probability $\mathbb{P}(y = 1|\mathbf{x}, s = 0)$. Also, for this the value of c is needed. They estimate this by averaging the result of the classical learning classifier for all the positive examples.

In Liu, Dai, Li, Lee, and Yu [15], all sorts of PU–learning algorithms are build. They use a different approach that contains two steps:

1. Identify a set of reliable negative examples from the unlabelled set.
2. Build a set of classifiers applying a classification algorithm and then choose a good classifier.

These two steps can be repeated to increase the number of negative examples that are classified as negative, while maintaining that positive examples are correctly classified.

In Lee and Liu [16], PU–learning is used with Weighted Logistic Regression. They define:

$$\mathbb{P}(s = 1|y = 1) = 1 - \alpha$$

and

$$\mathbb{P}(s = 0|y = 1) = \alpha$$

where α is unknown. To avoid the problem of not knowing α, the expected sum of false positive and false negative error frequencies is minimized. This is shown to be equal to expected weighted error where false positives are multiplied by $\mathbb{P}(s = 0)$ and false negatives by $\mathbb{P}(s = 1)$. Logistic regression with regularization (adding the sum of squared weights) is performed to approximate the conditional probability $\mathbb{P}(s = 1|\mathbf{x})$. This function is convex, en therefore with simple gradient descent the optimum can be found (under some conditions).

B

BACKGROUND INFORMATION

In this chapter, some background information is given about probit regression, convex optimization and ℓ_2-optimization.

B.1. PROBIT REGRESSION

Probit regression is a type of statistical classification model. It is used to predict a binary response variable, in our case $y \in \{0,1\}$. The name *probit* originates from **prob**ability un**it**. The model gives us the probability that $y = 1$. Then a threshold can be chosen to determine which probability is high enough to be classified as a case.

The probit function is equal to

$$f(x) = \Phi(x) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right],$$

which is the link function for probit regression. The link function links the variables and the response variable in a continuous way.

THE DERIVATIVE OF THE PROBIT FUNCTION

The derivative of the function f is needed, further on. If the probit model is used, hence $f(x) = \Phi(x) = \frac{1}{2}\left[1 + \text{erf}(\frac{x}{\sqrt{2}})\right]$, then

$$f'(x) = \phi(x) = \frac{e^{\frac{1}{2}x^2}}{\sqrt{2\pi}},$$

where $\phi(x)$ is the pdf and $\Phi(x)$ is the cdf of the standard normal distribution.

B.2. CONVEX OPTIMIZATION

Convex optimization, a subfield of optimization, looks at minimizing convex functions over convex sets. Because of the convexity property a local minimum must be a global minimum.

An optimization problem, finding some $\mathbf{w}^* \in \mathcal{W} \subset \mathbb{R}^d$ such that

$$f(\mathbf{w}^*) = \min\{f(\mathbf{w}) : \mathbf{w} \in \mathcal{W}\},$$

is called convex, if $\mathcal{W}$ is a closed convex set and $f(\mathbf{w})$ is convex in $\mathcal{W}$. We call $\mathcal{W}$ the feasible set, which is the set of all the possible values of the parameters $\mathbf{w}$, and $f(\mathbf{w}) : \mathbb{R}^d \Rightarrow \mathbb{R}$ is the objective function, which we'd like to minimize.

In Dattorro [17], the following definitions for convex sets and convex functions are used:

Definition B.1. *A set $\mathcal{W}$ is convex $\Leftrightarrow$ for all $w_1, w_2 \in \mathcal{W}$ and $0 \le \mu \le 1$:*

$$\mu w_1 + (1-\mu) w_2 \in \mathcal{W}.$$

Definition B.2. *Let $\mathcal{W}$ be a convex set and let $f : \mathcal{W} \to \mathbb{R}$ be a function. Function f is called convex in $\mathcal{W}$ $\Leftrightarrow$ for all $w_1, w_2 \in \mathcal{W}$ and $0 \le \mu \le 1$:*

$$f(\mu w_1 + (1-\mu) w_2) \le \mu f(w_1) + (1-\mu) f(w_2). \tag{B.1}$$

If the inequality in (B.1) in Definition B.2 can be replaced by a strict inequality, then we have that function f is strictly convex. Also by reversing the inequality sign in Definition B.2 flips the definition to concavity. In other words:

Definition B.3. *Function $f : \mathcal{W} \to \mathbb{R}$ is concave $\Leftrightarrow$ $-f$ is convex.*

The standard form is often used to describe optimization problems. In Boyd and Vandenberghe [18], the standard form is discussed more in detail and also a couple of variants are discussed.

The standard form for a optimization problem is:

$$\begin{aligned} &\text{minimize}_{\mathbf{w}} f(\mathbf{w}) \\ &\text{subject to } g_i(\mathbf{w}) \le 0 \quad i = 1, \dots, m \\ &\qquad\qquad h_j(\mathbf{w}) = 0 \quad j = 1, \dots, p \end{aligned}$$

Definition B.4. *An optimization problem, in the standard form, is called convex $\Leftrightarrow$ $f(\mathbf{w})$ and $g_i(\mathbf{w})$, for every $i \in \{1, \dots, m\}$, are convex functions and $h_j(\mathbf{w})$ is linear, or affine (both convex and concave).*

The use of the standard form is to make it more easy to compare different optimization problems and also to define when we call it a convex optimization problem.

Note that $h_j(\mathbf{w}) = 0$ can be written as two inequality constrains $h_j(\mathbf{w}) \le 0$ and $-h_j(\mathbf{w}) \le 0$. This shows that the equality constraints are unneeded, but in practice can be convenient. Furthermore it shows why $h_j(\mathbf{w})$ should be affine. Because if $h_j(\mathbf{w})$ is convex, then $h_j(\mathbf{w}) \le 0$ is convex, but $-h_j(\mathbf{w}) \le 0$ would be concave. And vice versa, hence $h_j(\mathbf{w})$ cannot be convex or concave, so it must be affine (linear).

B.2.1. PENALIZED CONVEX OPTIMIZATION

In penalized convex optimization, we try to find a solution for vector $\mathbf{w}$ that minimizes a convex objective function in $\mathbf{w}$ with an additional term that regulates the complexity of $\mathbf{w}$. Two common choices for this penalty are the ℓ_1-norm and the squared ℓ_2-norm.

Mathematically equivalent is formulating a constrained convex optimization problem. The penalty can be implemented as an inequality where the penalty is smaller or equal than a certain number. In this way, we can optimize the objective function in order to find a solution with a bounded norm.

B.2.2. USING ℓ_2-PROJECTION WITH ADAGRAD WITH DIAGONAL MATRICES

Suppose $\Theta = \{\boldsymbol{\theta} : \|\boldsymbol{\theta}\|_2^2 \leq \lambda\}$.

Lemma B.5. *The convex minimization problem is:*

$$\begin{aligned} &\text{minimize}_{\boldsymbol{\theta}} \ \frac{1}{2}\boldsymbol{\theta}^T H_t \boldsymbol{\theta} + q^T \boldsymbol{\theta} \\ &\text{subject to} \ \|\boldsymbol{\theta}\|_2^2 \leq \lambda. \end{aligned} \tag{B.2}$$

where $q = -H_t \tilde{\boldsymbol{\theta}}_{t+1}$, solved by:

$$\boldsymbol{\theta}_{t+1} = \tilde{\boldsymbol{\theta}}_{t+1}, \qquad \text{if } \|\boldsymbol{\theta}^*\|_2^2 \leq \lambda.$$

Otherwise, solve

$$\sum_{i=1}^{n} \frac{q_i^2}{(a_i + \mu)^2} = \lambda,$$

for μ and insert μ into

$$\boldsymbol{\theta}_{t+1} = (H_t + \mu I)^{-1} H_t \tilde{\boldsymbol{\theta}}_{t+1}$$

to get the solution.

Proof. Note that $\|\boldsymbol{\theta}\|_2^2 \leq \lambda$ is equivalent to $\boldsymbol{\theta}^T \boldsymbol{\theta} \leq \lambda$. The augmented Lagrangian will look like:

$$\mathscr{L}(\boldsymbol{\theta}, \mu) = \frac{1}{2}\boldsymbol{\theta}^T H_t \boldsymbol{\theta} + q^T \boldsymbol{\theta} + \frac{1}{2}\mu(\boldsymbol{\theta}^T \boldsymbol{\theta} - \lambda)$$

where μ is twice the Lagrange multiplier.

The Karush–Kuhn–Tucker (KKT) conditions, that must hold for the optimal point $\boldsymbol{\theta}^*$:, are used:

$$\begin{aligned} \text{Stationarity:} \quad & \nabla_{\boldsymbol{\theta}} \mathscr{L}(\boldsymbol{\theta}^*, \mu) = H_t \boldsymbol{\theta}^* + q + \mu \boldsymbol{\theta}^* = 0, \\ \text{Complementary slackness:} \quad & \mu \cdot ((\boldsymbol{\theta}^*)^T \boldsymbol{\theta}^* - \lambda) = 0, \\ \text{Condition:} \quad & \mu \geq 0. \end{aligned}$$

Complementary slackness gives two cases:

- **(a)** $\mu = 0$, which gives us the unconstrained case,
- **(b)** $(\boldsymbol{\theta}^*)^T \boldsymbol{\theta}^* = \lambda$.

Case (**a**): $\mu = 0$.

Then, stationarity gives us the solution:

$$\begin{aligned} H_t \boldsymbol{\theta}^* + q &= 0 \\ H_t \boldsymbol{\theta}^* &= -q \\ \boldsymbol{\theta}^* &= -H_t^{-1} q. \end{aligned}$$

Check whether $\|\boldsymbol{\theta}^*\|_2^2 \leq \lambda$. Otherwise, go to case (**b**).

Case (**b**): $(\boldsymbol{\theta}^*)^T \boldsymbol{\theta}^* = \lambda$.

The following three conditions need to be met:

$$H_t \boldsymbol{\theta}^* + q + \mu \boldsymbol{\theta}^* = 0 \tag{B.3}$$

$$(\boldsymbol{\theta}^*)^T \boldsymbol{\theta}^* = \lambda \tag{B.4}$$

$$\mu > 0 \tag{B.5}$$

From Condition (B.3), we get:

$$(H_t + \mu I)\boldsymbol{\theta}^* = -q.$$

If μ was known, then the solution will be equal to:

$$\boldsymbol{\theta}^* = -(H_t + \mu I)^{-1} q. \tag{B.6}$$

Condition (B.3) is used to find μ. Multiply (B.6) by $(\boldsymbol{\theta}^*)^T$ from the left side:

$$(\boldsymbol{\theta}^*)^T \boldsymbol{\theta}^* = q^T (H_t + \mu I)^{-2} q, \tag{B.7}$$

and from Condition (B.3), we get:

$$q^T (H_t + \mu I)^{-2} q = \lambda. \tag{B.8}$$

Suppose that the quadratic form (B.8) has two solution μ_1 and μ_2.

Take $\overline{\mu} = \max\{\, 0,\ \max\{\, \mu_1, \mu_2 \,\}\,\}$ and use $\mu = \overline{\mu}$ in (B.6) to compute $\boldsymbol{\theta}^*$.

Since matrix H_t is diagonal, we can rewrite (B.8).

Suppose a_i is the i$^{\text{th}}$ element on the diagonal of H_t. Find μ, such that:

$$\sum_{i=1}^{n} \frac{q_i^2}{(a_i + \mu)^2} = \lambda.$$

□

C

MISSING DATA PROBLEM

In this chapter, the assumption is that there is missing data in our problem. Rabobank provided a list of cases, transactions for which the label $s = 1$. For these transactions, we know that $y = 1$. For the rest for the transactions, the assumption is that the value for y, their "true label", is missing. Two methods, that both handle this missing data, will be discussed. In Section C.2, the Expectation-Maximization method, which is a frequentistic method, is explained. In Section C.3, the Data Augmentation method, which is a Bayesian method, is discussed. For both models, the likelihood function is needed. In Section C.1, the likelihood function will be derived.

These models could not incorporate the value of $c = \mathbb{P}(s = 1|\mathbf{x}, y = 1)$. Because, we though this was essential information to use in our model, we chose to take a slightly different model.

C.1. LIKELIHOOD FUNCTION

Suppose that each transaction i is labelled (so $s_i = 0$ or $s_i = 1$). Remember that transactions with label $s_i = 1$, are *cases* (so $y_i = 1$), because of **Assumption 1** (2.2). For the rest of the transactions, labelled $s = 0$, the value of y is unknown. The complete data set would be for every transaction i with features $\mathbf{x}_i$ to know whether $y_i = 1$, if the transaction is a case, else $y_i = 0$. But the sequence of y_i's is incomplete.

The following assumption holds for every transaction i, when the value of y_i is unknown:

$$\mathbb{P}(y_i = 1|\mathbf{x}_i) = f(\boldsymbol{\theta}^T \mathbf{x}_i), \tag{C.1}$$

where $f : \mathbb{R} \to [0,1]$ is either the logistic function $\psi(x) = \dfrac{1}{1+e^{-x}}$ or the standard normal distribution function $\Phi(x) = \dfrac{e^{-\frac{1}{2}x^2}}{\sqrt{2\pi}}$, depending on whether logistic regression or probit regression is used. Both functions have the same co-domain, namely $[0,1]$, which is the allowed area of a probability.

Only when the complete data is available (i.e. $\mathbf{y}$ complete), the likelihood can be computed. This likelihood is called the complete data likelihood, and denote it as $\mathscr{L}^C$, where a superscripted C indicates that the complete data is used. Hence, as if all y_i were available.

C.1.1. THE COMPLETE DATA LIKELIHOOD

The complete data likelihood function will become:

$$\begin{aligned}\mathscr{L}^C(\boldsymbol{\theta}|y_1,\ldots,y_n) &= \prod_{i=1}^{n} \mathbb{P}(y_i|\boldsymbol{\theta},\mathbf{x}_i) \\ &= \prod_{i=1}^{n} \left[f(\boldsymbol{\theta}^T\mathbf{x}_i)\right]^{y_i} \cdot \left[1 - f(\boldsymbol{\theta}^T\mathbf{x}_i)\right]^{1-y_i}.\end{aligned}$$

Then the logarithm of the complete data likelihood is taken:

$$\begin{aligned}\ell^C(\boldsymbol{\theta}|y_1,\ldots,y_n) &= \log \mathscr{L}^C(\boldsymbol{\theta}|y_1,\ldots,y_n)\\ &= \sum_{i=1}^{n}\left[y_i\cdot\log(f(\boldsymbol{\theta}^T\mathbf{x}_i)) + (1-y_i)\cdot\log\left(1-f(\boldsymbol{\theta}^T\mathbf{x}_i)\right)\right]\end{aligned} \tag{C.2}$$

Because of assumption 1, equation (C.2) can be rewritten. The complete log-likelihood function is split into two parts, the first part for the labelled transactions (which are cases) and the other for the non-labelled transactions. Let $W = \{\text{transaction } i : s_i = 1\}$ the set of labelled transactions, which implicates that for these transactions also holds $y_i = 1$ for $i \in W$. Equation (C.2) will become:

$$\ell^C(\boldsymbol{\theta}|y_1,\ldots,y_n) = \sum_{i\in W}\log\left(f(\boldsymbol{\theta}^T\mathbf{x}_i)\right) + \sum_{i\in W^C}\left[y_i\cdot\log(f(\boldsymbol{\theta}^T\mathbf{x}_i)) + (1-y_i)\cdot\log\left(1-f(\boldsymbol{\theta}^T\mathbf{x}_i)\right)\right] \tag{C.3}$$

C.2. EXPECTATION–MAXIMIZATION ALGORITHM

The Expectation-Maximization (EM) algorithm is first introduced in Dempster, Laird, and Rubin [19]. Since then the EM algorithm is used to solve a variety of problems.

In Borman [20], the following description is given: *"The Expectation–Maximization (EM) algorithm is an efficient iterative procedure to compute the Maximum Likelihood (ML) estimate in the presence of missing or hidden data. In ML estimation, we wish to estimate the model parameter for which the observed data are the most likely.*
Each iteration of the EM algorithm consists of two processes: The E-step, and the M-step. In the expectation, or E-step, the missing data are estimated given the observed data and current estimate of the model parameters. This is achieved using the conditional expectation, explaining the choice of terminology. In the M-step, the likelihood function is maximized under the assumption that the missing data are known. The estimate of the missing data from the E-step are used in the M-step as if they were the actual missing data.
Convergence is assured since the algorithm is guaranteed to increase the likelihood at each iteration."

Expectation–Maximization algorithm

1. INITIATE $\boldsymbol{\theta}_0$.
2. COMPUTE $Q(\boldsymbol{\theta}|\boldsymbol{\theta}_0,\mathbf{x}) = \mathbb{E}_{\boldsymbol{\theta}_0}[\log\mathscr{L}^C(\boldsymbol{\theta}|\mathbf{x},\mathbf{y})] = \mathbb{E}_{\boldsymbol{\theta}_0}[\ell^C(\boldsymbol{\theta}|\mathbf{x},\mathbf{y})]$ where the expectation is with respect to $k\left(\mathbf{y}|\boldsymbol{\theta}_0,\mathbf{x}\right)$.
3. COMPUTE $\boldsymbol{\theta}_1 = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}|\boldsymbol{\theta}_1,\mathbf{x})$.
4. Stop when the increment of the likelihood is small. When this is not the case go to Step 2. using $\boldsymbol{\theta}_0 = \boldsymbol{\theta}_1$.

For the EM algorithm, logistic regression was chosen, hence the logit function $f(x) = \dfrac{1}{1+e^{-x}}$ is used to predict $\mathbf{y}$. The function $k(\mathbf{y}|\boldsymbol{\theta},\mathbf{x}_i)$, from step 2. in the EM algorithm, is equal to the function f, namely

$$\mathbb{P}(Y=1|\tilde{\boldsymbol{\theta}},\mathbf{x}_i) = k\left(\mathbf{y}|\tilde{\boldsymbol{\theta}},\mathbf{x}_i\right) = f\left(\tilde{\boldsymbol{\theta}}^T\mathbf{x}_i\right) = \frac{1}{1+e^{-\tilde{\boldsymbol{\theta}}^T\mathbf{x}_i}} = p_i.$$

The power of the EM algorithm lies in easy expressions that comes out of step 2 and 3.

C.2.1. E - STEP

$Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}},\mathbf{x})$ is computed, using expression ( ) from Section 2.2.1:

$$\begin{aligned}Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}},\mathbf{x}) &= \mathbb{E}_{\tilde{\boldsymbol{\theta}}}[\ell^C(\boldsymbol{\theta}|\mathbf{x},\mathbf{y})]\\ &= \mathbb{E}_{\tilde{\boldsymbol{\theta}}}\left[-\sum_{i=1}^{n}\log\left(1+e^{-\boldsymbol{\theta}^T\mathbf{x}}\right) + \sum_{i=1}^{n}y_i\boldsymbol{\theta}^T\mathbf{x}\right]\end{aligned}$$

$$= -\sum_{i=1}^{n} \log\left(1+e^{-\boldsymbol{\theta}^T\mathbf{x}}\right) + \mathbb{E}_{\tilde{\boldsymbol{\theta}}}\left[\sum_{i=1}^{n} y_i \boldsymbol{\theta}^T\mathbf{x}\right]$$

$$= -\sum_{i=1}^{n} \log\left(1+e^{-\boldsymbol{\theta}^T\mathbf{x}}\right) + \sum_{i=1}^{n} \mathbb{E}_{\tilde{\boldsymbol{\theta}}}\left[y_i \boldsymbol{\theta}^T\mathbf{x}\right]$$

$$= -\sum_{i=1}^{n} \log\left(1+e^{-\boldsymbol{\theta}^T\mathbf{x}}\right) + \sum_{i=1}^{n} \sum_{y \in \{0,1\}} \left[\mathbb{P}(Y=y|\tilde{\boldsymbol{\theta}},\mathbf{x}_i)\cdot y_i \cdot \boldsymbol{\theta}^T\mathbf{x}\right]$$

$$= -\sum_{i=1}^{n} \log\left(1+e^{-\boldsymbol{\theta}^T\mathbf{x}}\right) + \sum_{i=1}^{n} \left[p_i \cdot 1 \cdot \boldsymbol{\theta}^T\mathbf{x} + (1-p_i)\cdot 0 \cdot \boldsymbol{\theta}^T\mathbf{x}\right]$$

$$= -\sum_{i=1}^{n} \log\left(1+e^{-\boldsymbol{\theta}^T\mathbf{x}}\right) + \sum_{i=1}^{n} \left[p_i \cdot \boldsymbol{\theta}^T\mathbf{x}\right]$$

where

$$p_i = f\left(\tilde{\boldsymbol{\theta}}^T\mathbf{x}_i\right) = \frac{1}{1+e^{-\tilde{\boldsymbol{\theta}}^T\mathbf{x}_i}}.$$

C.2.2. M - STEP

Then $Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}},\mathbf{x})$ in $\boldsymbol{\theta}$ is maximized. Suppose that $\boldsymbol{\theta}$ is d-dimensional. The derivative with respect to θ_j is taken:

$$\begin{aligned}\frac{\partial Q}{\partial \theta_j}\left(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}},\mathbf{x}\right) &= -\sum_{i=1}^{n} \frac{\mathbf{x}_{ij}e^{-\boldsymbol{\theta}^T\mathbf{x}_i}}{1+e^{-\boldsymbol{\theta}^T\mathbf{x}_i}} + \sum_{i=1}^{n} p_i \cdot \mathbf{x}_{ij} \\ &= -\sum_{i=1}^{n} \frac{\mathbf{x}_{ij}}{1+e^{\boldsymbol{\theta}^T\mathbf{x}_i}} + \sum_{i=1}^{n} p_i \cdot \mathbf{x}_{ij} \\ &= \sum_{i=1}^{n} \mathbf{x}_{ij} \cdot \left(p_i - \frac{1}{1+e^{\boldsymbol{\theta}^T\mathbf{x}_i}}\right),\end{aligned}$$

where $\mathbf{x}_{ij}$ is the element of $\mathbf{x}$ in the i^{th} row and the j^{th} column.

The derivatives need to be equal to 0 in order to find a maximal value for $Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}},\mathbf{x})$. When looking at the derivatives above, there is no explicit form for $\boldsymbol{\theta}$, for which the derivatives are all equal to zero. The power of the EM algorithm should lie in a direct way to compute $\boldsymbol{\theta}$ given the $\mathbf{x}_i$ and the previous parameter $\tilde{\boldsymbol{\theta}}$.

C.3. DATA AUGMENTATION METHOD

The Data Augmentation (DA) algorithm, a special case of the Gibbs sampler, which is a special case of the Metropolis-Hastings algorithm, is explained. The Gibbs sampler is a Markov Chain Monte Carlo (MCMC) method for obtaining samples from a multivariate probability function, when direct sampling is difficult. These samples can be used to approximate the joint distribution. For example, the joint distribution of two random variables, called W_1 and W_2, is of interest, but direct sampling is impossible. By sampling from the conditional distributions $W_1|W_2$ and $W_2|W_1$ iteratively, samples from the joint distribution are obtained.

The missing values of $\mathbf{y}$ are simulated and used to update our parameter θ. First, the needed assumptions are stated. Then the two steps of the Data Augmentation algorithm are explained.

The basic model will be logistic regression. Hence

$$\mathbb{P}(y=1|s=0) = f(\boldsymbol{\theta}^T\mathbf{x}),$$

where $f(x)$ is the logistic function $\psi(x)$.

We observe the pairs $(\mathbf{x}_i, s_i)$ for $i = 1,\ldots,n$. For each j, where $s_j = 1$, we have $y_j = 1$. The rest of the $\mathbf{y}$ are missing values. We would like to have the pairs $(\mathbf{x}_i, y_i)$ for $i = 1,\ldots,n$. Remember the definition of $W = \{\text{event } i : s_i = 1\}$.

Inspiration for the data augmentation implemented for the problem comes from Example 5 in Hobert [21].

With the data augmentation algorithm wee want to generate samples from the distribution of $\boldsymbol{\theta}$ and $\{y_i : i \notin W\}$ given $\{y_i : i \in W\}$. In this way we predict our parameter $\boldsymbol{\theta}$ and impute the missing values $\{y_i : i \notin W\}$. In order to so we will have to sample from the following two distributions iteratively:

1. $\{y_i : i \notin W\}$ given $\boldsymbol{\theta}$ and $\{y_i : i \in W\}$,
2. $\boldsymbol{\theta}$ given $\{y_i : i \notin W\}$ and $\{y_i : i \in W\}$.

Sampling from the first distribution can be done easily, as seen in step 1 below. Sampling from the second is more difficult. We will have to introduce a latent variable z in order to do so. Actually we sample from the second distribution using the data augmentation again. So we will sample from

2. $\boldsymbol{\theta}$, z given $\{y_i : i \notin W\}$ and $\{y_i : i \in W\}$,

by sampling from

2a. $\boldsymbol{\theta}$ given z, $\{y_i : i \notin W\}$ and $\{y_i : i \in W\}$,

2b. z given $\boldsymbol{\theta}$, $\{y_i : i \notin W\}$ and $\{y_i : i \in W\}$.

Below, we discuss this procedure in more detail.

C.3.1. STEP 0: CHOOSE A START VALUE FOR $\boldsymbol{\theta}$

First, choose a start value for the parameters $\boldsymbol{\theta}$. Call the choice $\boldsymbol{\theta}_0$.

C.3.2. STEP 1: DATA AUGMENTATION

In this step, the missing values of $\mathbf{y}$ are simulated. For each $j \in W^C$, so where $s_j = 0$:

$$y_j = \begin{cases} 1, & \text{with probability } f(\boldsymbol{\theta}^T \mathbf{x}), \\ 0, & \text{with probability } 1 - f(\boldsymbol{\theta}^T \mathbf{x}). \end{cases} \tag{C.4}$$

Hence, from the distribution of $\{y_i : i \notin W\}$ given $\boldsymbol{\theta}$ and $\{y_i : i \in W\}$ is sampled.

C.3.3. STEP 2: UPDATING PARAMETER $\boldsymbol{\theta}$

The missing values of $\mathbf{y}$ are simulated in Step 1. Hence, now a complete sequence y_i for $i = 1, \ldots, n$ is available. With this sequence, the parameter $\boldsymbol{\theta}$ is updated. Now generate a sample from $\pi(\boldsymbol{\theta}, z|y)$, where z is a dummy variable. Below, is discussed how the z is chosen. In order to sample from $\pi(\boldsymbol{\theta}, z|y)$, we sample from $\pi(z|\boldsymbol{\theta}, y)$ and $\pi(\boldsymbol{\theta}|z, y)$. A common problem is that z and $\boldsymbol{\theta}$ influence each other. In order to lower the dependence between z and $\boldsymbol{\theta}$, z is scaled:

$$z' = \sqrt{v} z.$$

This step consists of three sub steps. The first sub step is generating z from the truncated normal distribution. Let $\text{TN}(\mu, \sigma^2, u)$ denote a normal distribution with mean μ and variance σ^2 that is truncated to be positive if $u = 1$ and negative if $u = 0$. The second sub step draws a sample v from a Gamma distribution and then $z' = \sqrt{v} z$ is computed. The third step updates the parameter $\boldsymbol{\theta}$.

Adding step 2a to the data augmentation model, makes it easier to get samples from $\pi(\boldsymbol{\theta}, z|y)$. Adding step 2b, makes the model a Haar PX-DA model, which means it's a fast mixing Parameter eXpanded DA algorithm using an improper Haar density. The PX-DA algorithm is sometimes called Marginal Augmentation and can be used with a variety of densities. Step 2b can be excluded, if there is no sign of autocorrelation, by using just z for step 2c.

STEP 2A: GENERATING z

Given the $\boldsymbol{\theta}$ and y the z_i's are independent. Draw $Z_1, \ldots, Z_n$ independently such that $Z_i \sim \text{TN}(\boldsymbol{\theta}^T \mathbf{x}, 1, y_i)$, and call the result $z = (z_1, \ldots, z_n)^T$. A method called the inverse transform method is used to generate the samples from the truncated normal distribution.

If you want z to be sampled from a truncated standard normal distribution on (a, b), then:

$$z = \Phi^{-1}\left(\Phi(a) + (\Phi(b) - \Phi(a)) \cdot U\right),$$

where Φ is the cumulative distribution function, Φ^{-1} its inverse and U a vector of random numbers on $[0, 1]$.

The truncated normal distribution on $(-\infty, 0)$ is needed, when $y_i = 0$, and on $(0, \infty)$, when $y_i = 1$. Remember that $\Phi(-\infty) = 0$ and $\Phi(\infty) = 1$. Hence:

$$z_i = \begin{cases} \tilde{\Phi}^{-1}\left(\tilde{\Phi}(0) \cdot u_i\right), & \text{if } y_i = 0, \\ \tilde{\Phi}^{-1}\left(\tilde{\Phi}(0) + \left(1 - \tilde{\Phi}(0)\right) \cdot u_i\right), & \text{if } y_i = 1, \end{cases} \tag{C.5}$$

where $\tilde{\Phi}$ and $\tilde{\Phi}^{-1}$, with mean $\boldsymbol{\theta}^T\mathbf{x}$ and variance 1, are used. The expression could be rewritten using the cdf of the standard normal distribution $\Phi(\cdot)$, using the following identities:

$$\tilde{\Phi}(\alpha) = \Phi\left(\frac{\alpha - \mu}{\sigma}\right),$$

and

$$\tilde{\Phi}^{-1}(\beta) = \Phi^{-1}(\beta) \cdot \sigma + \mu,$$

where μ is the mean of the normal distribution, in this case $\boldsymbol{\theta}^T\mathbf{x}$, and σ is the variance, which is 1 in our case. Remember also that $\Phi(-x) = 1 - \Phi(x)$ and $\Phi(x) = -\Phi(1 - x)$ hold. Then, the second part of equation (C.5) can be rewritten:

$$\begin{aligned} z_i \cdot 1_{\{y_i=1\}} &= \tilde{\Phi}^{-1}\left(\tilde{\Phi}(0) + \left(1 - \tilde{\Phi}(0)\right) \cdot u_i\right), \\ &= \Phi^{-1}\left(\Phi(-\mu) + \left(1 - \Phi(-\mu)\right) \cdot u_i\right) + \mu, \\ &= -\Phi^{-1}\left(1 - \Phi(-\mu) - \left(1 - \Phi(-\mu)\right) \cdot u_i\right) + \mu, \\ &= -\Phi^{-1}\left(\Phi(\mu) - \Phi(\mu) \cdot u_i\right) + \mu, \\ &= -\Phi^{-1}\left(\Phi(\mu) \cdot (1 - u_i)\right) + \mu, \\ &= -\Phi^{-1}\left(\Phi(\mu) \cdot \tilde{u}_i\right) + \mu, \end{aligned}$$

where $\tilde{u}_i$ is also a sample from a uniform distribution on $[0, 1]$. Hence

$$z_i = \begin{cases} \Phi^{-1}\left(\Phi(-\mu) \cdot u_i\right) + \mu, & \text{if } y_i = 0, \\ -\Phi^{-1}\left(\Phi(\mu) \cdot u_i\right) + \mu, & \text{if } y_i = 1, \end{cases} \tag{C.6}$$

with $\Phi(\cdot)$ the cdf of the standard normal distribution, its inverse $\Phi^{-1}(\cdot)$, and u_i sampled from a uniform distribution on $[0, 1]$.

STEP 2B: GENERATING z'

This step is to lower the dependence between z and $\boldsymbol{\theta}$. Draw $v \sim \text{Gamma}\left(\frac{n}{2}, \frac{z^T(I-H)z}{2}\right)$, where $H = \mathbf{x}(\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T$. Set $z' = \sqrt{v}z$.

STEP 2C: UPDATING PARAMETER $\boldsymbol{\theta}$

Draw the next state $X_{n+1} \sim \text{N}\left(\tilde{\boldsymbol{\theta}}(z'), (\mathbf{x}^T\mathbf{x})^{-1}\right)$, where $\tilde{\boldsymbol{\theta}}(z') = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T z'$. This sample will be our new $\boldsymbol{\theta}$.

In order to avoid taking the inverse of $\mathbf{x}^T\mathbf{x}$, we consider a QR decomposition on $\mathbf{x}$. Hence two matrices are found, a orthogonal matrix Q and an upper triangular matrix R, such that $\mathbf{x} = QR$.

From the following equation $\tilde{\boldsymbol{\theta}}(z)$ is obtained:

$$\tilde{\boldsymbol{\theta}} = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T z',$$

$$
\begin{aligned}
&\Leftrightarrow \mathbf{x}^T \mathbf{x}\tilde{\boldsymbol{\theta}} = \mathbf{x}^T z', \\
&\Leftrightarrow (QR)^T QR\tilde{\boldsymbol{\theta}} = (QR)^T z', \\
&\Leftrightarrow R^T Q^T QR\tilde{\boldsymbol{\theta}} = R^T Q^T z', \\
&\Leftrightarrow R^T R\tilde{\boldsymbol{\theta}} = R^T Q^T z', \\
&\Leftrightarrow R\tilde{\boldsymbol{\theta}} = Q^T z',
\end{aligned}
$$

by using Gauss elimination instead of the inverse of R.

Sampling from the multivariate distribution of X_{n+1} can be done in the following way:

$$
\begin{cases} R\mathbf{v} = \mathbf{w} \\ X_{n+1} = \tilde{\boldsymbol{\theta}} + \mathbf{v}\,, \end{cases} \tag{C.7}
$$

where $\mathbf{w}$ is a vector of samples from the standard normal distribution. Since:

$$
X_{n+1} \sim \mathrm{N}\left(\tilde{\boldsymbol{\theta}}, (R^T R)^{-1}\right),
$$

$$
X_{n+1} = \tilde{\boldsymbol{\theta}} + R^{-1}\mathbf{w}.
$$

This can be rewritten as $R(X_{n+1} - \tilde{\boldsymbol{\theta}}) = \mathbf{w}$. Call $\mathbf{v} = X_{n+1} - \tilde{\boldsymbol{\theta}}$, then this is the same as in (C.7).

D

MATLAB CODE

This chapter displays the files created with MATLAB.

D.1. AdaGrad algorithm

D.1.1. Run model

This code uses the following other codes:

- `startup_model.m` (see D.1.1)
- `make_training_and_test_set.m` (see D.1.1)
- `save_facts.m` (see D.1.1)
- `pick_case.m` (see D.1.1)
- `pick_record.m` (see D.1.1)
- `update_parameter.m` (see D.1.1)
- `test_record.m` (see D.1.1)
- `test_training_record.m` (see D.1.1)
- `compute_AUC_value.m` (see D.1.1)
- `performance.m` (see D.1.1)

```
%% First time run startup_model
% Choose whether you want to clear and close everything
closing = 1;
if closing == 1
    close all
end
clearing = 1;
if clearing == 1
    clear all
end

timer_total = tic;
normalizing = 0;
scaling = 0;
maximum_value = [];
sample_mean = [];
```

```
17 sample_variance = [];
18
19 % Initiate parameters, learning rate, l_1-restriction, etc.
20 casetype = 'all';          % Choose casetype
21 lambda = 5;                % l_1-constraint: 1-norm leq delta
22 eta = 1;                   % Learning rate eta
23 versie = 100;                % Version
24 prior = 1;                 % 0: take no prior on c // 1: take a prior on c.
25
26 load('Result/selected_variables.mat','selected_variables')
27 selected_variables = sort(selected_variables(:));
28
29 map_result = ['Result\',casetype,'_version',num2str(versie),'\'];   % Save in
       map_result
30 mkdir(map_result)
31
32 [m,type,ext1,map1,ext2,map2,list,list2,likelihood,w_last,wt,G,H,projv,s,delta] =
       startup_model(casetype,lambda,selected_variables);
33
34 %% Prepare
35 %%%%%%%%%%%%%%%%%%%%%%%%%%%%
36 % Make training and test set
37 make_new_set = 1;
38 n_sub = 0.01;  % Take this percentage of the non-cases.
39 n_samp = 20;    % Once every n_samp non-cases, insert a case.
40 perc = 0.9;    % Percentages used for the training set.
41 n_subsets = 9;
42 [n,perm,n_subtotal,n_train,n_test,n_save,number_of_cases,m_train,m_test,perm2,
       perm_train,perm_test,count] = make_training_and_test_set(make_new_set,list2,list,
       map1,casetype,n_sub,n_samp,perc,map_result);
43 count_infile = cumsum(count);
44
45 % Scaling or normalizing
46 sample_struct = load('sample.mat','maximum','sample_mean','sample_variance');
47 normalizing = 1;
48 %scaling = 1;
49 i_selected = selected_variables(find(selected_variables>1523))-1523;
50 if scaling == 1
51     maximum_value2 = sample_struct.maximum;
52     maximum_value(1,:) = maximum_value2(i_selected)';
53     clear maximum_value2
54 elseif normalizing == 1
55     sample_mean2 = sample_struct.sample_mean;
56     sample_variance2 = sample_struct.sample_variance;
57
58     sample_mean = sample_mean2(i_selected);
59     sample_variance = sample_variance2(i_selected);
60     clear sample_mean2
61     clear sample_variance2
62 end
63 clear i_selected
64 clear sample_struct
65
66 %% Validate
67 validating = 0;
68
```

```
69 if validating == 1
70 lambda_test = 1:5;
71
72 % Cases
73 m8 = ceil(m_train/n_subsets);
74 m9 = m_train - 8*m8;
75 start = 0; start2=0;
76 for i=1:n_subsets
77     if i==n_subsets
78         trainingset_cases(i,1:m9)=perm_train((1:m9)+start);
79         trainingset_noncases(i,1)=perm(1+start2);
80     else
81         trainingset_cases(i,1:m8)=perm_train((1:m8)+start);
82         trainingset_noncases(i,1)=perm(1+start2);
83         start = start + m8;
84         start2 = start2 + 1;
85     end
86 end
87
88 % Validate
89 for i=1:n_subsets
90     disp(' ')
91     disp(['Validation step ',num2str(i),' has started.'])
92     disp(' ')
93     numbers=1:n_subsets;
94     numbers(i)=[];
95     clear train_indices
96     n_end = m9;
97
98     for j=1:length(lambda_test)
99         AUC(i,j+10) = compute_AUC(lambda_test(j),trainingset_cases,
               trainingset_noncases,numbers,i,n_end,w_last,type,delta,eta,list2,map2,
               list,map1,count_infile,prior,selected_variables,sample_mean,
               sample_variance,maximum_value,normalizing,scaling);
100    end
101    disp(' ')
102    disp(['Validation step ',num2str(i),' is finished.'])
103 end
104
105 average_AUC = sum(AUC,1)/size(AUC,1);
106 [sorted_average_AUC,indices_sort] = sort(average_AUC,'descend');
107 lambda = lambda_test(indices_sort(1));
108 else
109     %lambda = 15;
110 end
111
112 %% Save facts
113 permission = 'w';  % overwrite
114 %permission = 'a'; % add
115 save_facts(permission,casetype,lambda,eta,type,delta,n_subtotal,n_train,n_test,
       n_save,n_samp,m_train,m_test,m,number_of_cases,map_result,perc,selected_variables
       );
116
117 % Load format specifications
118 fileID = fopen('FormatSpec.txt','r');
119 FormatSpec = fscanf(fileID,'%s');
```

```
120 fclose(fileID);
121
122 %% Train
123 % lambda = 5
124 % save_facts(permission,casetype,lambda,eta,type,delta,n_subtotal,n_train,n_test,
      n_save,n_samp,m_train,m_test,m,number_of_cases,[map_result,'lambda5\'],perc,
      selected_variables);
125 %%%%%%%%%%%%%%%%%%%
126 % Start training
127 %%%%%%%%%%%%%%%%%%%
128 disp(' ')
129 disp('-- Training has started --')
130 disp(' ')
131 last = 0;
132 indices_train = perm_train(randperm(length(perm_train)));
133 k_case = 1;
134
135 for i = 1:n_train
136     % Load .csv-file, pick all records and store in 'records'
137     records = pick_record(list2(perm(i)).name,map2);
138     disp(['File #',num2str(i),': Found ',num2str(size(records,1)),' records.'])
139     disp(['Name: ',list2(perm(i)).name])
140     disp(' ')
141
142     % For this file, use each record for training
143     for j = 1:size(records,1)
144         % Start with a case and after every n_samp non-cases inject a case.
145         if mod(last+j,n_samp)==0 || last+j == 1
146             % Pick a random case from the training set
147             place = indices_train(k_case);
148             temp = find(count_infile >= place);
149             k = temp(1);
150             clear temp
151             if k == 1
152                 place_infile = place;
153             else
154                 place_infile = place - count_infile(k-1);
155             end
156
157             % Load variables from case
158             record = pick_case(list(k).name,map1,place_infile);
159             disp(' ')
160             disp(['Case #',num2str(place_infile),', from ',list(k).name,', is loaded
                  .'])
161
162             % Update parameters
163             [randomID(last+j,1),timestamp(last+j,1),label{last+j,1},s(last+j),
                  likelihood(last+j,1),G,~,w_new,~] = update_parameter(record,w_last,G,
                  delta,type,m,last+j,eta,lambda,prior,selected_variables,sample_mean,
                  sample_variance,maximum_value,normalizing,scaling); %#ok<SAGROW>
164
165             w_last = w_new;
166             disp(['Case #',num2str(place),', from ',list(k).name,', is finished.'])
167
168             last = last + 1;
169             if k_case+1 <= length(perm_train)
```

```
170                 k_case = k_case + 1;
171             else
172                 k_case = 1;
173             end
174         end
175         % Use a non-case from 'records' to update model
176         disp(' ')
177         disp(['File #',num2str(i),': Record #',num2str(j),'/',num2str(size(records
                ,1)),' is loaded.'])
178 
179         % Take one of the non-cases in 'records'
180         record = records{j,1};
181 
182         % Update parameters
183         [randomID(last+j,1),timestamp(last+j,1),label{last+j,1},s(last+j),likelihood
                (last+j,1),G,H,w_new] = update_parameter(record,w_last,G,delta,type,m,
                last+j,eta,lambda,prior,selected_variables,sample_mean,sample_variance,
                maximum_value,normalizing,scaling); %#ok<SAGROW>
184 
185         % Save parameter in w_last
186         w_last = w_new;
187         disp(['File #',num2str(i),': ','Record #',num2str(j),'/',num2str(size(
                records,1)),' is finished.'])
188     end
189     last = last + size(records,1);
190     clear records
191     disp(['File #',num2str(i),' is finished.'])
192 end
193 
194 %% Train on a case one more time
195 k_case = 1;
196 
197 % Pick a random case from the training set
198 for i = 1:1
199     place = indices_train(k_case);
200     temp = find(count_infile >= place);
201     k = temp(1);
202     clear temp
203     if k == 1
204         place_infile = place;
205     else
206         place_infile = place - count_infile(k-1);
207     end
208 
209     % Load variables from case
210     record = pick_case(list(k).name,map1,place_infile);
211     disp(' ')
212     disp(['Case #',num2str(place),', from ',list(k).name,', is loaded.'])
213 
214     % Update parameters
215     last = size(randomID,1);
216     [randomID(last+1,1),timestamp(last+1,1),label{last+1,1},s(last+j),likelihood(
            last+1,1),G,H,w_new] = update_parameter(record,w_last,G,delta,type,m,last+j,
            eta,lambda,prior,selected_variables,sample_mean,sample_variance,maximum_value
            ,normalizing,scaling); %#ok<SAGROW>
217 
```

```
218     k_case = k_case + 1;
219 end
220 w_last = w_new;
221
222 % Save parameters w_last and w_new
223 save_parameter(w_last,w_new,map_result);
224 disp(['Case #',num2str(place),', from ',list(k).name,', is finished.'])
225
226 % Scale gamma to c again
227 c_final = 1/(1+exp(w_last(m+1)));
228
229 % End of training
230 disp([num2str(last),' records used during training'])
231 disp(' ')
232 disp('-- Training has ended --')
233 disp(' ')
234
235 %% Testing the result
236 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
237 %  Testing on the test set
238 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
239 disp(' ')
240 disp('-- Testing has started (part 1) --')
241 disp(' ')
242 last = 0;
243
244 % Test on non-cases in test set
245 for i = n_train+1:n_subtotal
246     records = pick_record(list2(perm(i)).name,map2);
247     disp(['File #',num2str(i),': Found ',num2str(size(records,1)),' records.'])
248     disp(['Name: ',list2(perm(i)).name])
249
250     for j = 1:size(records,1)
251         record = records{j,1};
252
253         [test_randomID(last+j,1),test_timestamp(last+j,1),true_label{last+j,1},
              true_s(last+j,1),test_y(last+j,1)] = test_record(record,w_last,type,
              selected_variables,sample_mean,sample_variance,maximum_value,normalizing,
              scaling); %#ok<SAGROW>
254     end
255     last = last + size(records,1);
256     clear records
257     disp(['File #',num2str(i),' is finished.'])
258     disp(' ')
259 end
260
261 % Test on cases in test set
262 perm_test = sort(perm_test);
263 for j = 1:length(perm_test)
264     place = perm_test(j);
265     temp = find(count_infile >= place);
266     k = temp(1);
267     clear temp
268     if k == 1
269         place_infile = place;
270     else
```

```
271         place_infile = place - count_infile(k-1);
272     end
273     record = pick_case(list(k).name,map1,place_infile);
274 
275     disp(['Case #',num2str(place_infile),' from: ',list(k).name])
276 
277     [test_randomID(last+j,1),test_timestamp(last+j,1),true_label{last+j,1},true_s(
            last+j,1),test_y(last+j,1)] = test_record(record,w_last,type,
            selected_variables,sample_mean,sample_variance,maximum_value,normalizing,
            scaling); %#ok<SAGROW>
278 end
279 
280 % Save test result
281 save([map_result,'\scores_on_testset_label',casetype,'.mat'],'perm_test','test_y','
        true_s','test_randomID','test_timestamp');
282 disp(' ')
283 disp('-- Testing has ended (part 1) --')
284 disp(' ')
285 
286 %% Testing on training set
287 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
288 disp(' ')
289 disp('-- Testing has started (part 2) --')
290 disp(' ')
291 last = 0;
292 
293 % Test on non-cases in training set
294 for i = 1:n_train
295     records = pick_record(list2(perm(i)).name,map2);
296     disp(['File #',num2str(i),': Found ',num2str(size(records,1)),' records.'])
297     disp(['Name: ',list2(perm(i)).name])
298 
299     for j = 1:size(records,1)
300         record = records{j,1};
301 
302         [train_randomID(last+j,1),train_timestamp(last+j,1),true_label2{last+j,1},
                true_s2(last+j,1),train_y(last+j,1)] = test_training_record(record,w_last
                ,type,selected_variables,sample_mean,sample_variance,maximum_value,
                normalizing,scaling); %#ok<SAGROW>
303     end
304     last = last + size(records,1);
305     clear records
306     disp(['File #',num2str(i),' is finished.'])
307     disp(' ')
308 end
309 
310 % Test on cases in training set
311 perm_train = sort(perm_train);
312 for j = 1:length(perm_train)
313     place = perm_train(j);
314     temp = find(count_infile >= place);
315     k = temp(1);
316     clear temp
317     if k == 1
318         place_infile = place;
319     else
```

```
320             place_infile = place - count_infile(k-1);
321         end
322         record = pick_case(list(k).name,map1,place_infile);
323 
324         disp(['Case #',num2str(place_infile),' from: ',list(k).name])
325 
326         [train_randomID(last+j,1),train_timestamp(last+j,1),true_label2{last+j,1},
                true_s2(last+j,1),train_y(last+j,1)] = test_training_record(record,w_last,
                type,selected_variables,sample_mean,sample_variance,maximum_value,normalizing
                ,scaling); %#ok<SAGROW>
327 end
328 
329 % Save test result
330 save([map_result,'scores_on_trainingset_label',casetype,'.mat'],'perm_train','
        train_y','true_s2','train_randomID','train_timestamp');
331 disp(' ')
332 disp('-- Testing has ended (part 2) --')
333 disp(' ')
334 
335 %% Gathering results
336 %%%%%%%%%%%%%%%%%%%%%%
337 fontsize = 12;
338 width = 2;
339 stepsize = 0.01;
340 interesting_threshold = 0.5;
341 
342 % On the test set
343 [AUC_test,TPR1,FPR1,SPC1,ACC1,CF1] = compute_AUC_value(test_y,true_s,stepsize);
344 
345 test_result = [test_y,true_s];
346 
347 x_standard = linspace(0,1,1000);
348 
349 % Save results
350 save([map_result,'FPRandTPR_test',casetype,'.mat'],'FPR1','TPR1','SPC1','ACC1','
        AUC_test','CF1','test_result');
351 
352 % On the training set
353 [AUC_train,TPR4,FPR4,SPC4,ACC4,CF4] = compute_AUC_value(train_y,true_s2,stepsize);
354 
355 train_result = [train_y,true_s2];
356 
357 % Plot for the training set
358 h1 = figure;
359 plot(x_standard,x_standard,'k');
360 hold on
361 xlabel('FPR','FontSize',fontsize)
362 ylabel('TPR','FontSize',fontsize)
363 title('ROC curves')
364 plot(FPR4,TPR4,'r','LineWidth',width)
365 plot(FPR1,TPR1,'b','LineWidth',width)
366 legend('','Training set','Test set','Location','SouthEast')
367 hold off
368 set(gca,'XLim',[0 1],'YLim',[0 1.05],'FontSize',fontsize)
369 
370 % Save results
```

```
371 save([map_result,'FPRandTPR_train',casetype,'.mat'],'FPR4','TPR4','SPC4','ACC4','CF4
        ','AUC_train','train_result');
372
373 % Save pictures
374 saveas(h1,[map_result,'ROCcurves.eps'],'psc2')
375 saveas(h1,[map_result,'ROCcurves.jpg'])
376
377
378 % Displaying the AUC values
379 disp(['Test: The AUC value for testing is equal to ',num2str(AUC_test),'.'])
380 disp(['The AUC value for training is equal to ',num2str(AUC_train),'.'])
381
382 total_time = toc(timer_total);
```

STARTUP_MODEL.M

```
1  function [m,type,ext1,map1,ext2,map2,list,list2,likelihood,w_last,wt,G,H,projv,s,
       delta] = startup_model(casetype,lambda,selected_variables)
2  %% Initiate
3  ind = sum(selected_variables<=1523);
4  ind2 = sum(selected_variables>1523);
5  % Choose start values for our parameters: theta and c
6  c_0 = 0.9;
7  gamma_0 = log(c_0)-log(1-c_0);
8  theta_0 = ones(1+ind*26+ind2,1)/(1+ind*26+ind2)*lambda;
9  m = length(theta_0);
10
11 % Logistic regression or probit regression
12 type = 'logit';
13 % type = 'probit';
14
15 % Making a list of files
16 if strcmp(casetype,'all') == 1
17     casetype = '*';
18 end
19 ext1 = ['*label',casetype,'.csv'];
20 map1 = 'Data_csv\Anon TM Cases CSV\';
21 nameoffiles = [map1,ext1];
22 len = length(dir(nameoffiles));
23 list(1:len,1) = dir(nameoffiles);
24 ext2 = '*.csv';
25 map2 = 'Data_csv\Anon TM CSV\';
26 nameoffiles2 = [map2,ext2];
27 list2(1:length(dir(nameoffiles2))) = dir(nameoffiles2);
28
29 % Initiate
30 likelihood = [];
31 w_last = [theta_0;gamma_0];
32 wt = zeros(m+1,1);
33 G = zeros(m+1,1);
34 H = zeros(m+1,1);
35 projv = zeros(m+1,1);
36 s = [];
37 delta = 0.01;
38
39 end
```

MAKE_TRAINING_AND_TEST_SET.M

```
function [n,perm,n_subtotal,n_train,n_test,n_save,number_of_cases,m_train,m_test,
    perm2,perm_train,perm_test,count] = make_training_and_test_set(make_new_set,list2
    ,list,map1,casetype,n_sub,n_samp,perc,map_result)
if make_new_set == 1
    % Number of files
    n = length(list2);
    perm = randperm(n,n);

    % Take a sub set of the data
    n_subtotal = floor(n_sub*n);

    % Splitting non-cases into a training and a test set
    n_train = floor(perc*n_subtotal);
    n_test = n_subtotal - n_train;

    % Save every n_save iteration
    n_save = 10000;

    % Splitting cases into a training and a test set
    number_of_cases = 0;
    for i = 1:length(list)
        [~, ~, count(i)] = pick_case(list(i).name,map1,0);
        number_of_cases = number_of_cases + count(i);
    end
    m_train = floor(perc*number_of_cases);
    m_test = number_of_cases - m_train;
    perm2 = randperm(number_of_cases,number_of_cases);
    perm_train = perm2(1:m_train);
    perm_test = perm2(m_train+1:number_of_cases);

    % Save permutations
    save([map_result,'permutations_label',casetype,'.mat'],'perm','perm2','
        perm_train','perm_test','n_subtotal','n','n_subtotal','n_train','n_test','
        n_save','n_samp','m_train','m_test','number_of_cases','n_sub');
else
    load([map_result,'permutations_label',casetype,'.mat'],'perm','perm2','
        perm_train','perm_test','n_subtotal','n','n_subtotal','n_train','n_test','
        n_save','n_samp','m_train','m_test','number_of_cases','n_sub');
end

end
```

SAVE_FACTS.M

```
function save_facts(permission,casetype,lambda,eta,type,delta,n_subtotal,n_train,
    n_test,n_save,n_samp,m_train,m_test,m,number_of_cases,map_result,perc,
    selected_variables)
nov = length(selected_variables);
if nov == 3046
    lov = 'all variables';
else
    lov = num2str(selected_variables);
end

fileID = fopen([map_result,'facts_label',casetype,'.txt'],permission);
```

```
% Save the facts used to run the model
fprintf(fileID,'%s\r\n',['All facts about the run for label ',casetype]);
fprintf(fileID,'%s %i\r\n','Number of cases for this casetype: number_of_cases = ',
    number_of_cases);
fprintf(fileID,'%s %i\r\n','l1-restriction: lambda = ',lambda);
fprintf(fileID,'%s %i\r\n','Learning rate: eta = ',eta);
fprintf(fileID,'%s %s\r\n','Logit/ Probit model: type = ',type);
fprintf(fileID,'%s %i\r\n','H_t = delta + diag(G_t)^(1/2): delta = ',delta);
fprintf(fileID,'%s %i %s\r\n','Percentage used for training is ',perc,'%.');
fprintf(fileID,'%s %i\r\n','Number of files of non-cases selected: n_subtotal = ',
    n_subtotal);
fprintf(fileID,'%s %i\r\n','Number of files of non-cases selected for training:
    n_train = ',n_train);
fprintf(fileID,'%s %i\r\n','Number of files of non-cases selected for testing:
    n_test = ',n_test);
fprintf(fileID,'%s %i\r\n','Save every n_save th iteration: n_save = ',n_save);
fprintf(fileID,'%s %i\r\n','1 case injected after n_samp non-cases: n_samp = ',
    n_samp);
fprintf(fileID,'%s %i\r\n','Number of files of cases selected for training: m_train
    = ',m_train);
fprintf(fileID,'%s %i\r\n','Number of files of cases selected for testing: m_test =
    ',m_test);
fprintf(fileID,'%s %i\r\n','Number of parameters: m = ',m+1);
fprintf(fileID,'%s %i\r\n','Number of variables: nov',nov);
fprintf(fileID,'%s %s\r\n','List of variables: ',lov);
fclose(fileID);
end
```

PICK_CASE.M

```
function [M,i,file_length] = pick_case(filename,map,place)
filename = [map,filename];

% Read whole file
fileID = fopen(filename);

% Pick record at spot n+1
M = textscan(fileID,'%s',3049,'HeaderLines',place,'Delimiter',{',','\r'});

% Close file
fclose(fileID);

if nargout > 1
    i = place;
end

if nargout > 2
    file_length = length(csvread(filename,1,3048));
end

end
```

PICK_RECORD.M

```
function M = pick_record(filename,map)
filename = [map,filename];

```

```
% Read whole file
fileID = fopen(filename);
file_length = length(csvread(filename,1,3048));

% Pick record at spot n+1
for i = 1:file_length
    M{i,1} = textscan(fileID,'%s',3049,'HeaderLines',1,'Delimiter',{',','\r'});
end

% Close file
fclose(fileID);
end
```

UPDATE_PARAMETER.M

This code uses `myfun.m` (see D.1.3), `projectv.m` (see D.1.2) and `determinelocation.m` (see D.1.3).

```
function [randomID,timestamp,label,s,likelihood,G,H,w_new,X] = update_parameter(
    record,w_last,G,delta,type,m,current_record,eta,lambda,prior,selected_variables,
    sample_mean,sample_variance,maximum_value,normalizing,scaling)
% Extract information from record
randomID     = record{1,1}(1);
timestamp    = record{1,1}(2);
label        = char(record{1,1}(3));
s            = double(strcmp(label,'O0')~=1);

% Take data from record
data = zeros(1,length(selected_variables));
t_values = sum(selected_variables<=1523);
t_indices = find(selected_variables<=1523);
data(1,1:t_values) = double( char( record{1,1}(3 + t_indices ) ) )-64;          %
    Top-values T0001 - T1523

v_indices = find(selected_variables>1523);
temp1 = record{1,1}(1527:length(record{1,1}));
data2(1,1:1523) = sscanf(sprintf('%s*', temp1{:}), '%i*');    % Values V0001 - V1523
data(1,t_values+1:length(selected_variables)) = data2(1,selected_variables(v_indices
    )-1523); %#ok<FNDSB>
clear data2
clear temp1

% Normalize numerical variables
if normalizing == 1
    data(1,t_values+1:length(selected_variables)) = normalize(data(1,t_values+1:
        length(selected_variables)),sample_mean,sample_variance);
elseif scaling == 1
    data(1,t_values+1:length(selected_variables)) = scale(data(1,t_values+1:length(
        selected_variables)),maximum_value);
end

% Store data in a cell 'X'
X{1,1} = data;

% Compute g_t, G_t and H_t
[likelihood,g] = myfun(w_last,X,s,type,prior,selected_variables);
G = G + g.^2;
H = delta + sqrt(G);
```

```
35
36 % Gradient step
37 i_up = determinelocation(X{1,1},selected_variables);
38 wt = w_last;
39 wt(i_up,1) = w_last(i_up,1) - eta*g(i_up)./(H(i_up));
40
41 % Change of variables
42 v = sqrt(H).*wt;
43 a = H.^(-1/2);
44
45 % Projection step
46 projv = projectv(v(1:m),a(1:m),lambda);
47 projv = [projv;v(m+1)];
48
49 % Return to original variables
50 w_new = a.*projv;
51 if sum(isnan(w_new)) >= 1
52     disp('NaN value in the parameter.')
53     return
54 end
55 teller = 1;
56 while sum(abs(w_new(1:m))) > lambda+0.5
57     if teller == 1
58         disp(' Parameters do not fullfill l_1-constraint.')
59         return
60     end
61     disp([' Try #',num2str(teller),': l_1 norm = ',num2str(sum(abs(w_new(1:m)))),' >
           ',num2str(lambda)])
62     teller = teller + 1;
63     w_new = a.*projectv(sqrt(H).*w_new,a,lambda);
64     if teller == 1000
65         disp(['Stopped at record #',num2str(current_record),'.'])
66         return
67     end
68 end
69
70 % Clear variables collected from .csv-file
71 clear record
72 clear data
73 end
```

TEST_RECORD.M

```
1 function [test_randomID,test_timestamp,true_label,true_s,test_y] = test_record(
      record,w_last,type,selected_variables,sample_mean,sample_variance,maximum_value,
      normalizing,scaling)
2 % Extract information from record
3 test_randomID     = record{1,1}(1);
4 test_timestamp    = record{1,1}(2);
5 true_label        = char(record{1,1}(3));
6 true_s            = double(strcmp(true_label,'O0')~=1);
7
8 % What is theta?
9 theta = w_last(1:size(w_last,1)-1,1);
10
11 % Take data from record
```

```
data = zeros(1,length(selected_variables));
t_values = sum(selected_variables<=1523);
t_indices = find(selected_variables<=1523);
data(1,1:t_values) = double( char( record{1,1}(3 + t_indices ) ) )-64;          % Top-values T0001 - T1523

v_indices = find(selected_variables>1523);
temp1 = record{1,1}(1527:length(record{1,1}));
data2(1,1:1523) = sscanf(sprintf('%s*', temp1{:}), '%f*');
data(1,t_values+1:length(selected_variables)) = data2(1,selected_variables(v_indices)-1523);
clear data2
clear temp1

% Normalize numerical variables
if normalizing == 1
    data(1,t_values+1:length(selected_variables)) = normalize(data(1,t_values+1:length(selected_variables)),sample_mean,sample_variance);
elseif scaling == 1
    data(1,t_values+1:length(selected_variables)) = scale(data(1,t_values+1:length(selected_variables)),maximum_value);
end

% Store data in a cell 'X'
X{1,1} = data;
[j,border] = determinelocation(X{1,1},selected_variables);

% Compute test score, the probability that y = 1.
test_y = f(prodthetax(theta,X{1,1},j,border),type);

% Clear variables collected from .csv-file
clear record
clear data

end
```

TEST_TRAINING_RECORD.M

```
function [test_randomID,test_timestamp,true_label,true_s,test_y] = test_training_record(record,w_last,type,selected_variables,sample_mean,sample_variance,maximum_value,normalizing,scaling)
% Extract information from record
test_randomID    = record{1,1}(1);
test_timestamp   = record{1,1}(2);
true_label       = char(record{1,1}(3));
true_s           = double(strcmp(true_label,'O0')~=1);

% What is theta?
theta = w_last(1:size(w_last,1)-1,1);

% Take data from record
data = zeros(1,length(selected_variables));
t_values = sum(selected_variables<=1523);
t_indices = find(selected_variables<=1523);
data(1,1:t_values) = double( char( record{1,1}(3 + t_indices ) ) )-64;          % Top-values T0001 - T1523

```

```
v_indices = find(selected_variables>1523);
temp1 = record{1,1}(1527:length(record{1,1}));
data2(1,1:1523) = sscanf(sprintf('%s*', temp1{:}), '%f*');
data(1,t_values+1:length(selected_variables)) = data2(1,selected_variables(v_indices)-1523);
clear data2
clear temp1

% Normalize numerical variables
if normalizing == 1
    data(1,t_values+1:length(selected_variables)) = normalize(data(1,t_values+1:length(selected_variables)),sample_mean,sample_variance);
elseif scaling == 1
    data(1,t_values+1:length(selected_variables)) = scale(data(1,t_values+1:length(selected_variables)),maximum_value);
end

% Store data in a cell 'X'
X{1,1} = data;
[j,border] = determinelocation(X{1,1},selected_variables);

% Compute test score, the probability that y = 1.
test_y = f(prodthetax(theta,X{1,1},j,border),type);

% Clear variables collected from .csv-file
clear record
clear data

end
```

COMPUTE_AUC_VALUE.M

This code uses `performance.m` (see D.1.1).

```
function [AUC_value,TPR,FPR,SPC,ACC,CF] = compute_AUC_value(scores,true_label,stepsize)
FPR = 1;
TPR = 1;
threshold = stepsize;
AUC_value = 0;
i=2;

while threshold < 1
    train_label = double(scores > threshold);

    % Confusion_matrix = [TP, FP; FN, TN]
    % False Positive Rate / fall-out: FPR = FP / (FP + TN)
    % True Postive Rate / Sensitivity: TPR = TP / (TP + FN)
    % Specificity or True Negative Rate: SPC = TN / (TN + FP)
    % Accuracy: (TP + TN) / (TP + TN + FP + FN)
    if nargout>3
        [CF,TPR(i),FPR(i),SPC(i),ACC(i)] = performance(true_label,train_label);
    else
       [~,TPR(i),FPR(i),~,~] = performance(true_label,train_label);
    end

    % AUC value
```

```
    AUC_value = AUC_value + (FPR(i-1)-FPR(i))*TPR(i)+0.5*(FPR(i-1)-FPR(i))*(TPR(i-1)
        -TPR(i));

    % Next threshold
    threshold = threshold + stepsize;
    i = i+1;
end

TPR = [TPR,0];
FPR = [FPR,0];
AUC_value = AUC_value + (FPR(i-1)-FPR(i))*TPR(i)+0.5*(FPR(i-1)-FPR(i))*(TPR(i-1)-TPR
    (i));

end
```

PERFORMANCE.M

```
function [Confusion_matrix,TPR,FPR,SPC,ACC] = performance(v1,v2)
%% How did we perform?
% v1 is the real value
% v2 is the labeled value
TN = length(find(v1==0&v2==0));
FP = length(find(v1==0&v2==1));
FN = length(find(v1==1&v2==0));
TP = length(find(v1==1&v2==1));

Confusion_matrix = [TP, FP; FN, TN];

if nargout > 1
% True positive rate a.k.a. sensitivity
TPR = TP /(TP + FN);
if nargout > 2
% Specificity = 1 - false positive rate
FPR = FP /(FP + TN);
if nargout > 3
SPC = 1 - FPR;

if nargout > 4
% Accuracy
ACC = (TP + TN) / (TP + TN + FP + FN);
end
end
end
end
% Adjusted accuracy
ACC_adjusted = TN / (TN + FP + FN);

end
```

D.1.2. PROJECTION ALGORITHM

The projection algorithm, described in Section 3.2, has the following code:

```
function z_star = projectv(v,a,c)
% v = -eta t H_t^(-1/2)overline(g)_t
% a(i) = (diag(G_t)^(-1/2))(i,i)
ind = find(v < 0);
pv = abs(v);

% Check input
if sum(a<0)~= 0 % Not every element >= 0
    disp('Vector a has a element below zero.')
    return
end
if c<0
    disp('Scalar c cannot be below zero.')
    return
end

% Is projection needed?
if sum(a.*pv)<=c
    % No
    z_star = pv;
    %disp('Project v: No projection was needed.')
else
%     disp('Project v: Projection is needed.')
    % Yes
    [mu,i] = sort(pv./a,1,'descend');
%     disp('Sorted.')
    % Determine rho, the largest integer s.t.
    % sum(a(i(1:rho)).*pv(i(1:rho))) - mu(i(rho))*sum(a(i(1:rho)).^2) < c
    minus = 0;
    rho = floor(0.5*length(mu));
    maxus = length(mu);
    while (sum(a(i(1:rho)).*pv(i(1:rho))) - mu(rho)*sum(a(i(1:rho)).^2) >= c) || (
        sum(a(i(1:rho+1)).*pv(i(1:rho+1))) - mu(rho+1)*sum(a(i(1:rho+1)).^2) < c)
        if sum(a(i(1:rho)).*pv(i(1:rho))) - mu(rho)*sum(a(i(1:rho)).^2) >= c
            maxus = rho;
            rho = floor((minus+maxus)/2);
        else
            minus = rho;
            rho = floor((minus+maxus)/2);
        end
        if rho == length(mu)-1
            %disp('Maximal')
            break
        elseif rho == 1
            disp('Minimal')
            break
        end
    end
%     disp('Found rho.')
    if rho == 0
        disp('Error: Rho is equal to 0. No rho found...')
        return
    end
```

```
53
54      % Determine the factor that needs to be extracted
55      theta = (sum(a(i(1:rho)).*pv(i(1:rho))) - c)/(sum(a(i(1:rho)).^2));
56
57      % Correction
58      z_star = max(pv - theta*a, 0);
59  %     disp('Projection is finished.')
60  end
61
62  % Put back the original signs
63  z_star(ind) = - z_star(ind);
64  end
```

D.1.3. OBJECTIVE FUNCTION FOR ADAGRAD

Main file to compute the objective function and the gradient, defined in Section 2.6. The code uses two other MATLAB files: `determinelocation.m` (see Section D.1.3) and `prodthetax.m` (see Section D.1.3).

```
function [h1,h2] = myfun(variable,X,s,type,prior,selected_variables)
% Renaming the variables
theta = variable(1:size(variable,1)-1,1);
gamma = variable(size(variable,1),1);
c = 1/(1+exp(-gamma));

% Number of records
N = size(X,2);

% Initiate
func = zeros(N,1);
func_deriv = zeros(N,1);

% Iterate
for i = 1:N
    % Gathering the indices of theta worth using
    [j,border] = determinelocation(X{1,i},selected_variables);

    % Compute f(theta^T x)
    func(i,1) = f(prodthetax(theta,X{1,i},j,border),type);
    func_deriv(i,1) = f_accent(prodthetax(theta,X{1,i},j,border),type);
end
cf = c .* func;

% Prior for c
if prior == 1
    a = 15;
    b = 1;
    prior_c = c^(a-1)*(1-c)^(b-1)/beta(a,b);
    factor_c = (a-1)/c-(b-1)/(1-c);
end

% Calculate objective function
if prior == 1
    h1 = -(sum(log(s.*cf+(1-s).*(1-cf))) + log(prior_c));
else
    h1 = -(sum(log(s.*cf+(1-s).*(1-cf))));
end

if nargout > 1                % if gradient required
    n = size(variable,1);
    h2 = zeros(n,1);
    h2(1) = sum(s.*(func_deriv./func) - (1-s).*(c.*(func_deriv./(1-cf))));
    if prior == 1
        h2(n) = sum(s)/c - sum((1-s).*(func./(1-cf))) + factor_c;
    else
        h2(n) = sum(s)/c - sum((1-s).*(func./(1-cf)));
    end
    for i = 1:N               % per record
        % Which X are non-zero?
        [j,~] = determinelocation(X{1,i},selected_variables);
        % What is the label?
```

```
53          if s(i)==1
54              h2(j) = h2(j) + (func_deriv(i)./func(i));        % Theta
55          else
56              h2(j) = h2(j) - (c*(func_deriv(i)./(1-cf(i))));     % Theta
57          end
58      end
59      h2(n) = h2(n) / (2+exp(-gamma)+exp(gamma));
60      h2(:) = -h2(:);
61  end
62
63  end
```

DETERMINELOCATION.M

```
1  function [vector,number] = determinelocation(x,selected_variables)
2  % Record gives us 1523 categorical and 1523 numerical variables. We'd like
3  % to determine the indices in theta to which these values correspond.
4  ind = sum(selected_variables<=1523);
5  ind2 = sum(selected_variables>1523);
6
7  vector(1,1:ind) = x(1:ind) + (0:26:(ind-1)*26) + 1;
8  vector(1,ind+1:ind+ind2) = (1:ind2)' + ind*26 + 1;
9
10 number = ind*26 + 1;
11 end
```

PRODTHETAX.M

```
1  function number = prodthetax(theta,x,indices,border)
2  % We determine the inner product between theta and x. In order to do so we
3  % use indices to save time. In indices the locations in theta and x are
4  % saved.
5  ind = find(indices<=border);
6  ind2 = indices(ind);
7
8  ind3 = find(indices>border);
9  ind4 = indices(ind3);
10
11 % Theta_0
12 number = theta(1);
13
14 % Theta's corresponding to the categorical variables
15 number = number + sum(theta(ind2));
16
17 % Theta's corresponding tot the numerical variables (multiplied by the
18 % value for x
19 number = number + x(length(ind)+1:length(ind)+length(ind3)) * theta(ind4);
20 end
```

BIBLIOGRAPHY

[1] T. F. Chan, G. H. Golub, and R. J. LeVeque, *Algorithms for computing the sample variance: Analysis and recommendations,* The American Statistician **Vol. 37**, pp. 242 (1983).

[2] B. P. Welford, *Note on a method for calculating corrected sums of squares and products,* Technometrics **4**, 419 (1962).

[3] D. E. Knuth, *Seminumerical Algorithms*, 3rd ed., Art of Computer Programming, Vol. 2 (Addison-Wesley Longman Publishing Co., Boston, 1997).

[4] C. Elkan and K. Noto, *Learning classifiers from only positive and unlabeled data,* (2008).

[5] H. Robbins and S. Monro, *A stochastic approximation method,* The Annals of Mathematical Statistics **Vol. 22**, pp. 400 (1951).

[6] M. A. Figueiredo, *Adaptive sparseness for supervised learning,* (2003).

[7] J. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization,* (2010).

[8] N. V. Chawla, N. Japkowicz, and A. Kotcz, *Editorial: special issue on learning from imbalanced data sets,* SIGKDD Explor. Newsl. **6**, 1 (2004).

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, *Smote: synthetic minority over-sampling technique,* J. Artif. Int. Res. **16**, 321 (2002).

[10] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, *Smoteboost: Improving prediction of the minority class in boosting,* in *PKDD*, Lecture Notes in Computer Science, Vol. 2838, edited by N. Lavrac, D. Gamberger, H. Blockeel, and L. Todorovski (Springer) pp. 107–119.

[11] H. Guo and H. L. Viktor, *Learning from imbalanced data sets with boosting and data generation: the databoost-im approach,* SIGKDD Explor. Newsl. **6**, 30 (2004).

[12] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, *A study of the behavior of several methods for balancing machine learning training data,* SIGKDD Explor. Newsl. **6**, 20 (2004).

[13] M. Kubat and S. Matwin, *Addressing the curse of imbalanced training sets: One-sided selection,* (1997).

[14] T. Fawcett, *An introduction to roc analysis,* Pattern Recogn. Lett. **27**, 861 (2006).

[15] B. Liu, Y. Dai, X. Li, W. S. Lee, and P. S. Yu, *Building text classifiers using positive and unlabeled examples,* (2003).

[16] W. S. Lee and B. Liu, *Learning with positive and unlabeled examples using weighted logistic regression,* (2003).

[17] J. Dattorro, *Convex optimization and euclidean distance geometry,* (Meboo Publishing, 2005) Book section 2 and 3.

[18] S. Boyd and L. Vandenberghe, *Convex Optimization*, 7th ed. (cambridge university press, 2009).

[19] A. P. Dempster, N. M. Laird, and D. B. Rubin, *Maximum likelihood from incomplete data via the em algorithm,* Journal of the Royal Statistical Society. Series B (Methodological) **Vol. 39**, pp. 1 (1977).

[20] S. Borman, *The expectation maximization algorithm: A short tutorial,* (2004), introduces the Expectation Maximization (EM) algorithm and fleshes out the basic mathematical results, inclusing a proof of convergence. The Generalized EM algorithm is also introduced.

[21] J. P. Hobert, *The data augmentation algorithm: Theory and methodology*, Handbook of Markov Chain Monte Carlo (Chapman and Hall/CRC Press, 2011).