Classify Neural Networks in Credit Scoring area based on the Financial Ratios

by

Zhentian Qiu (CID: 00856071)

Department of Mathematics
Imperial College London
London SW7 2AZ
United Kingdom

Thesis submitted as part of the requirements for the award of the MSc in Mathematics and Finance, Imperial College London, 2017-2018

Declaration

The work contained in this thesis is my own work unless otherwise stated.

Signature and date: [illegible]

2018.09.11

Acknowledgements

I want to express my gratitude to my external supervisor Dr. William Perraudin for his guidance and encouragement throughout this difficult project.

I want to extend my gratitude to my internal supervisor Prof. Harry Zheng for his expert advice and great efforts during the research.

I wish to express my special thanks to Jozsef Kutas for his support. His patiently teaching of SQL and python languages gives me invaluable help during the data cleaning. I would also like to thank Zhen Hu for her guidance of applying Logistic Regression.

Lastly, I thank my fiend Chao Tang for great support.

This thesis would not be completed without the support from all the people mentioned above. Once again, I appreciate all friends, Imperial College and Risk Control Limited for their precious resources and encouragement.

List of Acronyms

AGD	Adam Gradient Descent
AI	Artificial Intelligence
AR	Accuracy Ratio
AUC	Area Under the ROC Curve
DBN	Deep Brief Network
DT	Decision Tree
FAR	Fuzzy Adaptive Resonance
GD	general Gradient Descent
GPR	Gaussian Process Regression
GRBM	Gaussian RBM
KD	Kernel Density
KNN	K-Nearest Neighbor
LDA	Linear Discriminant Analysis
LR	Logistic Regression
LVQ	Learning Vector Quantization
MGD	Momentum Gradient Descent
MLP	Multi-Layer Perceptron
MoE	Mixture of Experts
NF	Neurofuzzy sustem
PDF	Probability Density Function
PNN	Probabilistic Neural Network
RBFN	Radial Basis Function Network
RBM	Restricted Boltzmann Machine
RGD	RMSprop Gradient Descent
ROC	Receiver Operating Characteristics
ROLS	Recursive Orthogonal Least Squares
SD	Sdandard Deviation

Contents

0 Introduction **9**

0.1 Thesis Goal 9

0.2 Background and Related Work 9

0.3 Structure of the Thesis 10

1 Data Cleaning **12**

1.1 Data Sources and Company Filter 12

1.2 Default Definition and Period Decision 13

1.3 Fields Decision 13

2 Model Scoring Process **16**

2.1 Introduction of Model Scoring Processes 16

2.2 Methods for Verification 16

2.3 K-Fold Cross-Validation 17

2.4 Gradient Descent Optimisation 18

2.5 Default Balance 20

3 Logistic Regression **21**

3.1 Description 21

3.2 Results 22

4 Multi-Layer Perceptron **23**

4.1 Description 23

4.2 Main Processes 24

4.3 Activation Functions 25

4.4 Hidden Layer Structure 27

4.5 Results 28

5 Probabilistic Neural Network **29**

5.1 Description 29

5.2 Main Processes 30

5.3 Dimension of Variance 31

5.4 Results 32

6 Radial Basis Function Network **33**

6.1 Description 33

6.2 Main Process 33

6.3 Relation with Gaussian Process Regression 35

6.4 Recursive Orthogonal Least Squares Training and Centre Selectors 36

6.5 K-Means Clustering Centre Selector 38

6.6 Results 39

7 Restricted Boltzmann Machine **41**

7.1 Description 41

7.2 Gaussian RBM and Further Improvements 42

7.3 Hybrid Model 43

7.4 Missing Data 45

7.5 Problems and Result 45

8 Further Directions **46**

8.1 RBM Dimensional Reduction 46

8.2 Voting Model 46

9 Conclusion **47**

A Appendix: Bisection method and Boundary Reduction method **49**

B Appendix: Gibbs sampling **51**

C Appendix: Hybrid RBM Formula Provement **52**

D Appendix: Tables **54**

0 Introduction

0.1 Thesis Goal

When a bank lends money to companies, the measure of **default** (Insolvency) risk is very significant as a reference for making a certain decision. An unpredicted bankrupt might cause over millions of losses which is undesirable for everyone. Credit Scoring is the topic of how to measure the default probability based on the past data. It helps companies to reduce losses of investment, and find some new chances for profit. Also, those model, which can produce a probability from the past data, is called **statistic model**.

Until now, one of the most commonly used statistic models in credit scoring has been the Logistic Regression (LR). It has been used in this area for over 50 years [1] and shows excellent performance consistently. However, the larger Random Access Memory, the faster CPU, and the even more powerful GPU have been developed, and computers can process a more extensive data with a higher dimension and more observations. A more massive data-set has more complexity, which reduces the accuracy of LR. For a highly dimensional data, the most effective way to increase the accuracy of LR is to reduce the dimension of data [2]. Finding an alternative to LR has become an unavoidable problem.

The idea of Neural Network comes from the biological neural network which is made by a group of chemically connected neurons. Similarly, a **Neural Network** is a group of linked neurons which receive information from other neurons and pass the processed information to next neurons through links. Since Google get a great achievement on alpha-go and alpha-zero [3][4], the Artificial Intelligence (AI) has become a popular topic in machine learning area. As the foundation of AI, Neural Networks has been continuously developed and have much application in different areas, e.g., Face Recognition, Advertising Serving, and Motion Capture. Neuron Networks require a much more extended training period (sometimes require weeks) than traditional machine learning, but they have a unique feature to handle large data size (over millions).

In this thesis, the database is financial balance sheets collected from the public website from 2013 until 2018. The size of the data is significantly different for each year (from thousands to hundreds thousand). The purpose of this thesis is to compare the accuracy of small and medium-size companies' bankrupt as predictions between Neuron Networks and the traditional statistic method based on the financial ratio from 2008 to 2015 and find some relations between model type and data size.

0.2 Background and Related Work

The idea of Neural Networks already exists over 50 years. Lots of published researches contain comparisons of different Neural Networks and traditional statistic models. In these papers, for different data sources or different data sizes, the best model may not be the same.

Desai Vijay S. applies LR, Linear Discriminant Analysis (LDA) and Multi-Layer Perceptron

(MLP) on three classes of data and 18 variables per observation in 1997 [5]. Three classes have 962, 918 and 853 observations respectively. After thousands of iterations, none of them outperform any other. In average, LR has slightly higher accuracy, but the MLP has a better result in some cases.

Piramuthu Selwyn compares MLP and Neurofuzzy system (NF) on real banks' data which includes 59 failed banks and 59 non-failed banks [6]. For each observation, there are 18 variables to predict the failure. A 10-hidden-neuron MLP gets the best result in classification.

Instead of comparing two or three models, West David analysed ten different models from three types of models with 1000 real data [7]. The representative of the Neural Networks is Mixture of Experts (MoE), Radial Basis Function Networks (RBFN), Learning Vector Quantization (LVQ), Fuzzy Adaptive Resonance (FAR) and MLP. LR and LDA represent the parametric models. K-Nearest Neighbour (KNN), Decision Tree (DT), and Kernel Density (KD) belong to the non-parametric models. LR achieved the best fit, and MoE becomes the second most used model by only 1% worse than LR. RBFN and MLP are third and fourth respectively.

In 2002, Lee, Chiu, Lu, and Chen developed a Hybrid Neural Discriminant Model and compared it with LR and MLP [8]. They found LR had the same performance as MLP on 6000 9-dimensional observations.

Besides, Bensic, Sarlija, and Zekic tested LR, MLP, RBFN, Probabilistic Neural Network (PNN), LVQ, FAR and DT [9]. A 50-hidden-neurons MLP, the LR, and the DT have best results on a database containing 160 observations and 31 variables per observation. In conclusion, MLP is the best model as it has the lowest Type I error (predicting bad credit applications as good).

However, there are some different conclusions which says something unusual about LR. In 2005, Ong, Huang, and Tzeng applied LR, MLP, and DT on 20 fields of 1000 observations from real-world data, and the best model is the DT [10]. In the same year, Cuicui, Desheng, and Dexiang got an incredible 100% accuracy result of Deep Brief Network (DBN) on 20 fields of 661 observations' database [11]. This result was much better than MLP, LR and Support Vector Machine (SVM).

0.3 Structure of the Thesis

The main content of this thesis contains 9 Chapters. Chapter 1 describes the process of data cleaning which includes the observation filter and fields selection. Then this chapter declares the definition of the default based on the filtered data. Finally, the chapter shows the benefit from the winsorization.

Chapter 2 introduces a general process of model training. Then it describes some model's performance measurement, such as Area Under the Receiver Operating Characteristics Curve (AUC) and Accuracy Ratio (AR). The chapter also explains the K-Fold Cross-Validation to improve the stability of accuracy measurements. After this, the Chapter compares some popular gradient descent methods and finds the best one. The best gradient descent method will apply to LR, MLP, PNN, RBFN, and RBM. In the end, the Chapter introduces some methods of balancing data which can solve the problem caused by the small percentage of defaults.

According to the listed previous researches, the five most representative models are chosen in this thesis. Firstly, LR is the only traditional statistic model used in this thesis. Except for LR, there are many other well-performed traditional models in credit scoring. KNN, DT, and SVM frequently have a better result than LR. However, the LR has a solid result and transparent process. It is always the best or one of the best models in the previous research mention before. Hence this thesis only takes LR as a representative of traditional scoring models. In Chapter 3, this thesis describes the iterative training and prediction process of LR. Besides, the Chapter shows the effect of the default balance and finds the best result from LR. The result is going to be a benchmark for all other models.

The second chosen model is the MLP, because all researches listed above has shown it is one of the best models to be used for this type of analysis (credit scoring). Chapter 4 describes the MLP model, and it explains the primary processes of MLP: the Forward activation and the Backwards propagation. Also, this chapter compares the different activation functions and different structure of Hidden layers. The best result is listed at the end of the Chapter.

Chapter 5 illustrates the structure of the PNN and describes its training and prediction processes. It also proves that increasing the dimension of variance can improve the accuracy of prediction.

Chapter 6 formulates the RBFN and shows its relation to the Gaussian Process Regression (GPR). Then the Chapter introduces two methods of centre reduction: Recursive Orthogonal Least Squares (ROLS) Training centre filtering and K-mean Clustering centre reduction.

Chapter 7 reviews binary Restricted Boltzmann Machine (RBM) and Gaussian Restricted Boltzmann Machine (GRBM). The Chapter also includes a new Hybrid Model which can handle binary and non-binary variable together. Then the Chapter explains how RBM deal with missing data.

The Further analysis is in the Chapter 8. It includes the RBM Dimension Reduction and the Voting model. These approaches have not been implemented. In addition, the conclusion is in the Chapter 9.

1 Data Cleaning

1.1 Data Sources and Company Filter

The financial data are taken from **Companies House** public data source in **XBRL & iXBRL** form, which including Assets, Liabilities, and Equity data between 2004 and 2018. The main signal of **default** (Insolvency), insolvency notices, are listed by **The Gazette**, the UKs official public record, from 2008 to 2018. The financial data is taken from the annual balance sheet of each company. As the real reported date may be not the date of balance sheet, so the month and date information are not considered in this thesis. Table 1 contains all fields of raw data. Turnover and Number of Employees fields are sparsely populated, hence they will only be involved in the data filtering. Other Current Assets, Other Fixed Assets, Other Current Liabilities, Total Long-term Liability, and Other shareholders Funds did not appear in the balance sheet, so they are calculated by other fields.

Code	Field name	Calculation
ca	Total Current Assets	
ca1	Cash Flow	
ca2	Trade and other Receivables (Debtors)	
ca3	Stock and Inventory	
ca4	Other Current Assets	ca4=ca-ca1-ca2-ca3
fa	Total Fixed Assets	
fa1	Tangible Fixed Assets	
fa2	Intangible Fixed Assets	
fa3	Other Fixed Assets	fa3=fa-fa1-fa2
stl	Total Current Liability (Less than 1 Year)	
stl1	Short-term Borrowing (Including Overdrafts)	
stl2	Trade and Other Payables	
stl3	Other Current Liabilities	stl3=stl-stl1-stl2
ltl	Total long-term Liabilities (Greater than 1 Year)	ltl=ltl1+ltl2+ltl3
ltl1	Long-term Borrowings (Debt and Loans)	
ltl2	Accruals Deferred Income	
ltl3	Long-term Provisions	
e	Total Shareholders funds or Equity	
e1	Shareholder funds from Issuance	
e2	Shareholder funds from Remained Earnings	
e3	Other Shareholders Funds	e3=e-e1-e2-e4
e4	Balance Sheet Minorities	
T	Turnover	
NE	Number of Employees	

Table 1: Fields of the Raw Data. If the field is not the first-hand data, then the calculation formula are listed in the column of Calculation.

In this thesis, only small and medium size companies are being analysed. One way of improving the quality of the data is by applying a few filters to the raw data, see table 2. Filter 1 and Filter 2 select small-size and medium-size companies. Filter 4 drops Dormant and Inactive companies. The rest of the filters are used to increase the quality of data.

ID	Filters detail	Equation
1	Remove any records with more than 500 Employees in that year.	NE<500
2	Remove any entities with more than 60 million on both of the Turnover and the Total Assets or more than 200 millions Total Assets.	(T<60 or ca+fa<60) and ca<200
3	Remove any records where the Shareholder Funds are greater than the Total Assets.	e<ca+fa
4	Remove any records with less than 0.5 million Total Assets.	ca+fa>0.5
5	Remove any records which failed the balancing test (absolute difference exceeds 0.002 million).	abs(ca+fa-stl-ltl-e)<0.002
6	Remove any records which do not have at least two consecutive fillings.	
7	Remove abbreviated accounts (Total Other Current Liability is equal to the Current Liabilities and Long-term Borrowing is 0.)	stl=stl3 and ltl1=0

Table 2: Filters applied on the raw data. Equations take field codes from Table 1.

1.2 Default Definition and Period Decision

The relation between the final financial reported year (the Last Filling Year) and the first insolvency notices year is listed in the Table 3. Only 434 defaults (7.2%) happen before the last filing year. Therefore, the default signal can be defined as

$$\text{Default Year} = \min\left\{\text{First Default Year}, \quad \text{Last Filing Year}{+}1\right\}.$$

Following this default definition, the default rate and the number of records after filters are shown in the Table 4. All the years before 2008 will be dropped because their default sample size is less than 100. 2016, 2017 and 2018 will also be removed as their low default rates, which means the data has a tremendous amount of uncaught insolvency notice after 2018. Therefore, only records between 2008 and 2015 are used for further analysis.

1.3 Fields Decision

Comparing with the first-hand data, Beaver and William H have proven financial ratios are more powerful to predict failures for at least five years in future [1] [12]. There are few further analysis on the topic of financial ratios in bankrupt prediction. Their selected fields are listed in the Table 5.

	First Default Year												
Last Filing Year	**<=07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**Total**
2006	0	0	1	1	0	0	0	0	0	1	0	0	3
2007	0	9	23	13	6	2	0	2	0	0	1	0	56
2008	0	0	53	42	22	13	1	9	2	3	1	0	146
2009	0	1	8	123	94	33	2	14	3	6	1	0	285
2010	0	0	0	23	194	140	8	22	18	12	7	0	424
2011	0	1	3	4	22	225	16	75	44	29	16	0	435
2012	0	0	3	1	6	33	43	236	86	60	22	6	496
2013	0	3	1	6	2	7	18	358	224	66	32	4	721
2014	0	0	2	7	5	1	0	48	424	265	53	9	814
2015	0	2	0	2	12	5	2	4	66	566	273	28	960
2016	0	4	8	10	15	19	1	18	18	99	594	207	993
2017	1	5	15	33	48	41	3	32	47	25	170	243	663
2018	0	1	1	1	3	0	0	2	1	1	2	5	17
Total	1	26	118	266	429	519	94	820	933	1133	1172	502	6013

Table 3: Default year analysis on filtered data

Report Year	**Total Observations**	**Will Not Default**	**Will Default**	**Default Rate**
<=2005	3	3	0	0.0%
2006	51	48	3	5.88%
2007	1205	1147	58	4.81%
2008	3914	3766	148	3.78%
2009	8876	8579	297	3.34%
2010	14985	14576	409	2.72%
2011	21498	21070	428	1.99%
2012	28960	28557	403	1.39%
2013	38853	38224	629	1.61%
2014	46480	45799	681	1.46%
2015	54288	53534	754	1.38%
2016	65695	65029	666	1.01%
2017	80103	79893	210	0.26%
2018	4664	4664	0	0.0%

Table 4: Default ratio and number of records after filters

Field	**Equation**	**I**	**II**	**III**	**usable**
Quick ratios	(ca-ca3) / stl	X			X
Current ratio	ca / stl	X	X		X
Inventary/Net working captital	ca3 / (ca-stl2)	X			X
Net working capital/Total assets	(ca-stl2) / (ca+fa)	X	X	X	X
Current assets/Total debt	ca / (stl+ltl)	X			X
Total debt/Equity	(stl+ltl) / e	X			X
Fixed assets/Equity	fa / e	X			X
Cash flow/Current liability	ca / stl	X			X
Current liability/Equaty	stl / e	X			X

Field	Equation	I	II	III	usable
Equity and Long-term debt/Fixed assets	(e + ltl) / fa	X			X
Inventory/Sales		X			
Fixed assets/Sales		X			
Total assets/Sales		X			
Net working capital/Sales		X			
Equity/Sales		X			
Earning before taxed (EBT)/Sales		X			
EBT/Total assets		X			
EBT/Equity		X			
EBT and Deprociation/Total debt		X			
Retained Earning/Total Assets	e2 / (ca + fa)		X	X	X
Earning Before Interest & Taxes/Total Assets				X	
Market value of Equity/Book value of Total debt				X	
Sales/Total Assets				X	
log(Total Assets)	log(ca + fa)		X		X
Total Liability/Total Assets	(stl + ltl) / (ca + fa)		X		X
Indicator of Total liability larger than Total Assets			X		
Funds provided by operation/Total Liability			X		
Indicator of negative Retained Earning	$\mathbb{1}$(e2<0)		X		X

Table 5: Fields analysis from I: Edmister, Robert O 1972 [13], II: Ohlson, James A. 1980 [15], III: Altman, Edward I. 1968 [14]. "X" in usable gives the data fields available. Other "X" means this field is included in the paper.
$\mathbb{1}(e2 < 0) = 1$ when $e2 < 0$, otherwise $\mathbb{1}(e2 < 0) = 0$

Some fields in the Table 5 (Earning Before Interest and Taxed, Sales, Market value of Equity and Fund from Operation) are not included in the raw data, so only 14 ratios can be used in further analysis, which is marked in the "usable" column of table 5.

Some values in the selected fields are unusually extreme ($10,000,000$ times larger than the median), which strongly affect the training process. Therefore, for each field, the top 1% largest data will be replaced by the minimum of this 1% data. Table 13 shows the LR results of default-balanced (default-balance is shown in Section 2.5) but non-winsorized data. Comparing with the results of default-balanced and winsorized data, shown in Table 7, 1% top winsorization effectively increases AR from 3.2% to the 28%. Hence the winsorized data is more accessible for prediction comparing to the non-winsorized data. It will be used as the final cleaned data for each model in the next few sections.

In this thesis, all models will be applied to the clean data on an annual basis. One **observation** contains the 14 selected fields for one company in one year. Each **observation** only have one **True label** that is 1 for default in next year, 0 for not default in next year. Each year's data-set is separated into a **Training data-set**, a **Validation data-set** and a **Test data-set** which have 67.5%, 22.5%, 10% of whole year's observations respectively.

2 Model Scoring Process

2.1 Introduction of Model Scoring Processes

To do the comparison between different model, a quantification process is necessary, in which a more accurate model should have a higher score. And the process of getting this score is called **Model Scoring Process**, which contains three process: the **Prediction process**, the **Training process** and the **Verification process**.

All statistic models in this thesis can get a probability of default (1) and a probability of non-default (0) for each observation. The label with higher probability is the **prediction** of the observation. The process of getting prediction is called the **Prediction process**. The main aim of the **Training process** is to obtain the minimal cost on the Training data-set, in which the **cost function** is the mean of Loss functions. A **Loss function** can measure the difference between the prediction and the True label for one observation. The common used Loss function is the Mean Square Error, $L(Y_i, \hat{Y}_i) = (Y_i - \hat{Y}_i)^2$, and the log loss, $L(Y_i, \hat{Y}_i) = -Y_i \log(\hat{Y}_i) + (1 - Y_i)\log(1 - \hat{Y}_i)$, where Y_i is the True label of ith observation and $\hat{Y}_i$ is its prediction. The cost reduction can be achieved by adjusting hyper-parameters (e.g., weights, biases, number of neurons). In the meantime, this thesis applies the statistic model to the Validation data-set as a stopping signal. The Training process keeps running until the model gets the lowest cost on the Validation data. It can efficiently avoid the problem of the **Over-Fitting** on the Training data-set, i.e., the model only gets a highly accurate result on the Training data-set and a much lower result on other data-sets. Then this thesis applies the Prediction process of the trained model onto the Test data-set, the score of a trained model is depended on the accuracy between its predictions and the True labels of the Validation data-set. The process of getting this score is called the **Verification process**. The best model should have high score each year by using this method described.

2.2 Methods for Verification

There are four types of possible results shown below.

		Observation	
		Default	**Non-Default**
Prediction	**Default**	dD	dN
	Non-Default	nD	nN

A good model will have less dN and nD results. Three common ways of measuring the accuracy of classifiers are

$$
\begin{aligned}
\text{Sensitivity} &= \frac{dD}{dD + nD}, \\
\text{Specificity} &= \frac{nN}{nN + dN}, \\
\text{Predictive Accuracy} &= \frac{dD + nN}{dD + nD + nN + dN}.
\end{aligned} \tag{1}
$$

Based on those, another measure of the performance is Area Under the ROC curve (AUC).

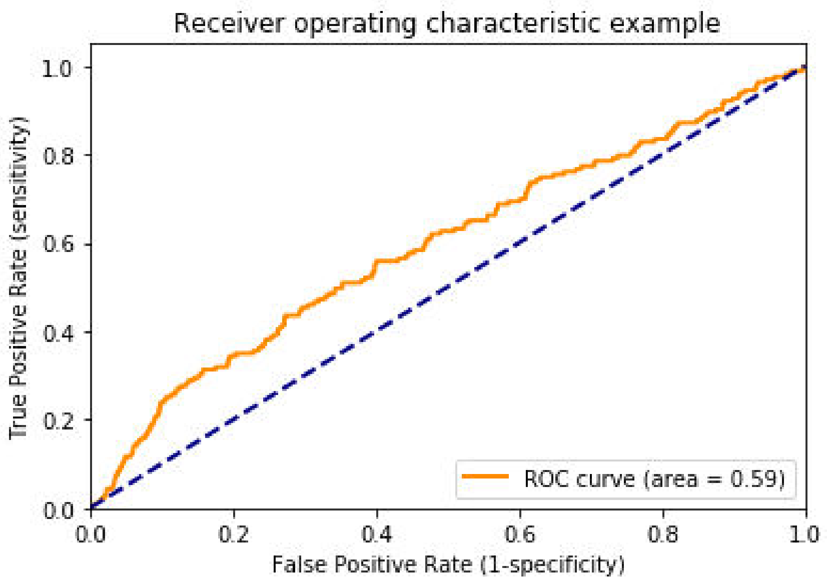

Figure 1: ROC of a MLP model on 2015 year

The Receiver Operating Characteristic (ROC) curve is a two-dimensional measure of binary classification performance [16] in which 1-specificity is plotted on the x-axis and sensitivity is plotted on the y-axis. An example of a ROC curve is shown in Figure 1. The AUC is the area under the ROC curve.

A random model has relation of

$$\frac{dD}{dN} = \frac{nD}{nN}.$$

Therefore Sensitivity $= 1 -$ Specificity for all samples. The ROC will be curved $y = x$, and half of the area is under the ROC, hence the AUC will be 0.5. On the other side, a perfect model has zero dN and nD. The ROC is $y = 1$, and the AUC is 1. For a more meaningful result, the Accuracy Ratio (AR), $AR = 2 * AUC - 1$ is used to be the measure of model performance in this thesis.

2.3 K-Fold Cross-Validation

Once the performance measure of the model is decided, the upcoming question is how consistent the measure is. According to the result in Table 4, 2008 and 2009 have a small size of default data. The selection of Test observations will strongly affect the result and cause high variance. K-fold Cross-Validation is an excellent solution to reduce the variance. Bengio, Yoshua, and Yves Grandvalet have explained K-Fold Cross-Validation can generate an unbiased expected error of the whole database [17] in 2004.

In practice, each year's data is randomly and uniformly partitioned into K disjoint subsets $(M_1, M_2, \ldots, M_k)$ by the Cross-Validation. Between any two of subsets, there are no same observations, and the difference in size can only be one or zero. Let each subset be an individual Test data-set. Their corresponding experimental data-sets, $(M_1', M_2', \ldots, M_k')$, are obtained by removing the elements of the Test data-set from the whole data-set. For each pair of data-sets, $(M_i, M_i')_i$, this thesis splits the M_i' into the Training data-set and the Validation data-set randomly, and apply the Training process of each statistic model on them.

In this thesis, ten sets of predictions are produced by 10 Test data-sets of 10-Fold Cross-Validation for each year's data. According to the K-Fold Cross-Validation. The observations of 10 prediction sets are disjoint, which compose a total prediction set. The comparison of the total prediction set and the True label is the **Annual AR** calculated by the AUC, see Section 2.2. The average of annual ARs from 2008 to 2015 is going to be the **Overall AR** of the model. The split of default observations set will strongly affect the result because of the small size of the default samples. Therefore, after repeating the whole model validation process five times (with different random seed), the average of the Overall ARs is the **Final AR** which is the final score of the model.

The measurement used to quantify the stability of the results is the standard deviation of 10 ARs from 10-Fold Cross-Validation. For a more stable model, the prediction accuracy should be less affected by the split of data-sets, hence a small standard deviation means a more consistent result.

2.4 Gradient Descent Optimisation

Sebastian Ruber describes more than eight gradient descent optimisation algorithms in his work [24]. In this thesis, only a few classical optimisation algorithms will be discussed: Momentum Gradient Descent (MGD), RMSprop Gradient Descent (RGD) and Adam Gradient Descent (AGD).

If the first derivative of the cost function to parameters, $\triangledown_\theta L(\theta^{(n)})$, is calculable, the most basic optimisation should be general Gradient Descent (GD).

$$\theta_{n+1} = \theta_n - \eta \triangledown_\theta L(\theta_n),$$

where L is the cost function, η is learning rate, $\theta_n = (\theta_n^{(1)}, \dots, \theta_n^{(m)})$ is the combination of all parameters after the nth iteration and $\theta^{(0)}$ is initialised randomly.

MGD is based on Newton's Second Law, and it helps accelerate the training process in the right direction. From the paper [24], the equation of MGD is:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \triangledown_\theta L(\theta_t) \\ \theta_{t+1} &= \theta_t - v_t \end{aligned} \tag{2}$$

where γ is commonly used as 0.9 and $v_0 = 0$.

The learning rate of RGD is dependent on the derivative of the cost function and slowly decrease to zero during the training process. It can reduce the rate of the Over-shooting which means one training iteration changes the sign of derivative.

$$\begin{aligned} \gamma_t &= 0.9\gamma_{t-1} + 0.1(\triangledown_\theta L(\theta_t))^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\gamma_t + \epsilon}} \triangledown_\theta L(\theta_t), \end{aligned} \tag{3}$$

where $\gamma_0 = 0$ and $\epsilon = 10^{-8}$.

AGD is similar to a mixture of MGD and RGD, and it has benefits of both models. The following is the equation of AGD from the paper:

$$\begin{aligned} v_t &= \beta_1 v_{t-1} + (1-\beta_1)\nabla_\theta L(\theta_t) \\ \gamma_t &= \beta_2 \gamma_{t-1} + (1-\beta_2)(\nabla_\theta L(\theta_t))^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\gamma_t}+\epsilon} v_t \end{aligned} \tag{4}$$

According to Kingma, Diederik P. and Jimmy Ba.'s research [25], 0.9, 0.999 and 10^{-8} are suitable default values for β_1, β_2 and ϵ respectively for the training process.

ID	GD		MGD		RGD		AGD	
	Cost	Loop	Cost	Loop	Cost	Loop	Cost	Loop
1	0.6377	7692	0.5886	6548	0.5153	9951	0.5264	7060
2	0.6368	9559	0.5542	8903	0.437	10028	0.5315	6842
3	0.6692	1629	0.5981	4835	0.4986	9931	0.5736	4992
4	**0.6698**	5442	**0.6437**	2552	0.516	9993	0.5103	7714
5	0.6359	5513	0.5303	5019	0.5316	**10032**	0.5592	5586
6	0.6294	9342	0.4965	9685	0.5675	4205	0.5405	8870
7	0.6559	**17483**	0.5755	**9750**	0.5363	7187	0.5117	7475
8	0.6728	3463	0.6072	9511	0.543	10014	0.5564	6190
9	0.6578	8724	0.5767	7369	**0.5939**	1328	0.5773	**9255**
10	0.637	9548	0.6238	2610	0.5637	2887	**0.5773**	5229
Average	0.6502	7839.5	0.5794	6678.2	0.5303	7555.6	0.5464	6921.3

Table 6: Cost is the final cost after 50,000 training iteration. Loop is the minimal number of iteration to achieve a cost which only has ±1% different to the final cost.

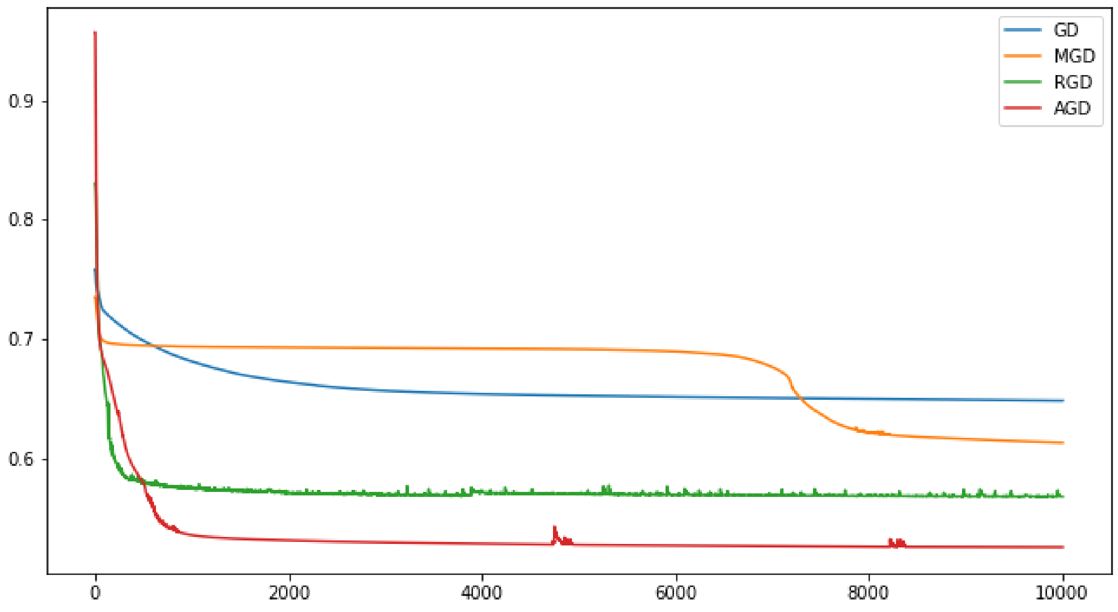

Figure 2: The x-axis is the number of training loop. The y-axis is the cost on training data.

The results of two-neurons single hidden layer MLP model on 2008 data-set is in Table 6. The

Cost is the final cost which is the mean of the log loss function after $50,000$ training iterations, see Section 4.2. The loop is the minimal number of training iteration to get a cost which only has $\pm 1\%$ different to the final cost. The maximal value of each column is bold. From the result we can find MGD has the fastest speed of converging, RGD has minimal training result and AGD has the best overall performance.

An example of cost converging for each gradient descent methods on the same data-set is shown in the Table 2. The MGD has an increasing converge speed from 6939 to 7196 loops, but MGD is trapped in a local minimum point before 6939. The RGD has lots wave before reach the minimum value. This high volatility process helps RGD to find a minimal global point. The graph of AGD has both steps and wave. AGD can be treated as the mixture of MGD and RGD.

2.5 Default Balance

There is a class imbalance issues with the data, only $1-4\%$ default been observed, as shown in Table 4. If a statistic model is directly applied to the unbalance data, there is a high probability that the model predicts every Test observations to be non-default. It is because that the cost function takes an average of all losses of each training observation. Guessing every observation to be non-default has a lower cost than predicting some default. An example of unbalanced data is in Table 14. After the Model Scoring process, the result shows that LR is not much different from a random model.

One way of avoiding the unbalance default problem is to balance the percentage of default observations before the training process. This methods are the **Over-Sampling** and the **Under-Sampling**. The Over-Sampling is to copy the default observations until them achieves 50% of the whole data-set [18]. Then a wrong prediction of default observation will also appear on its replica and cause more substantial punishment to the cost. On the other hand, the Under-Sampling drop non-default observations randomly to achieve a balance default structure [19]. The Under-Sampling can reduce the difficulty of calculation but lose information from non-default data.

The third method is the **Weighted-Loss** which increases the punishment of wrong default observation as well. It multiplies a sample weight onto each loss directly, in which the sample weight should depend on the percentage of the labels. In other words, a smaller percentage label should have a larger sample weight, and a larger percentage label has less sample weight. This thesis sets the sample weight of label l to be

$$SW(l) = \sum_i \frac{n_i}{n_l},$$

where n_i is the number of label i in the data-set. The benefit of the Weighted-Loss is that it can apply non-integer sample weights which perfectly balance any types of data. However the Over-Sampling can only increase default samples integer times, it can never balance a 30% default rate to 50%. Therefore, this thesis mainly applies the Weighted-Loss to balance default.

After applying the Weighted-Loss, an improved result of LR is shown in Table 7, which gives a 200 times better result than unbalanced data, in Table 14.

3 Logistic Regression

3.1 Description

Logistic Regression (LR) is the most popular statistical model used in credit scoring. In finance, it has been used for over 50 years to classify real data. Lots of researches have shown its stability and accuracy applied to real-world data. In previous research mention in section 0.2, almost everyone gets a best or second best result from using LR, hence this thesis chooses LR to be the representative of traditional statistical models and treat its result as a benchmark.

According to Frank E. Harrell, Jr.'s book *Regression Modeling Strategies* [20], for a given observation vector

$$X = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_n^1 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^m & x_2^m & \dots & x_n^m \end{bmatrix},$$

the LR prediction is written as

$$P[Y=1|X] = \frac{1}{1+exp(-X\beta-\beta_0)}, \quad \beta = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_n \end{bmatrix}.$$

The LR training process is to find the best hyper-plane $\{X|X\beta+\beta_0=0\}$ which divides the whole space into the default half-space $\{X|P[Y=1|X]>0.5\}$ and the non-default half-space $\{X|P[Y=1|X]<0.5\}$. A common used definition of the "best" is that its parameters, β, maximise the likelihood,

$$L(\beta) = \prod_{Y_i=1} P(X_i) \prod_{Y_i=0} (1-P(X_i)),$$

where $P(X_i) = P[Y_i = 1|X_i]$ are predictions, Y_i are true labels of training observations. The production for a large number of probabilities is more likely to be a tiny number and might cause some unpredictable numerical problems. Some papers [20][21] prefer to maximise the log likelihood,

$$\begin{aligned} \log L(\beta) &= \sum_i Y_i \log(P(X_i)) + (1-Y_i)\log(1-P(X_i)) \\ &= \sum_i Y_i \log\left(\frac{P(X_i)}{1-P(X_i)}\right) + \log(1-P(X_i)) \\ &= \sum_i Y_i(X_i\beta) - \log(1+\exp(X_i\beta)) \end{aligned} \tag{5}$$

After taking the derivative of $\log L(\beta)$ to β_j, the new β_j can be calculated by

$$\beta_j = \beta_j + \eta \sum_i X_{i,j}(Y_i - P(X_i)),$$

where η is the training ratio. 0.01 is a commonly used training ratio. By repeating this process, the model training is completed until reaches a converging $\log L$. Besides, as the first derivative of log likelihood is well defined, it is able to apply four ways of gradient descent methods on $-\log L$, which is described in the Section 2.4. The AGD method can produce a faster and more accurate training process.

Year		2008	2009	2010	2011	2012	2013	2014	2015	Overall AR
1	AR	34.8%	27.3%	31.5%	30.6%	23.6%	28.9%	24.8%	21.4%	27.9%
	SD	0.1594	0.1003	0.0856	0.0700	0.0684	0.0729	0.0665	0.0609	
2	AR	36.7%	32.2%	35.7%	32.9%	27.8%	34.6%	27.7%	23.6%	31.4%
	SD	0.1101	0.0749	0.0490	0.0677	0.0663	0.0460	0.0658	0.0645	
3	AR	34.5%	26.9%	28.8%	30.7%	26.5%	27.8%	26.3%	19.2%	27.6%
	SD	0.1161	0.0923	0.0839	0.1053	0.0429	0.0706	0.0608	0.0779	
4	AR	35.8%	28.1%	32.2%	31.8%	23.1%	29.7%	24.0%	22.2%	28.4%
	SD	0.1950	0.0869	0.0877	0.0563	0.0761	0.0954	0.0515	0.0550	
5	AR	38.2%	28.4%	33.0%	30.6%	22.4%	30.6%	24.4%	21.7%	28.6%
	SD	0.1733	0.0906	0.0777	0.0662	0.0746	0.0735	0.0408	0.0538	
Average AR		36.0%	28.6%	32.3%	31.3%	24.7%	30.3%	25.4%	21.6%	28.8%
Average SD		0.1508	0.0890	0.0768	0.0731	0.0657	0.0717	0.0571	0.0624	

Table 7: LR results with winsorized and default-balanced data.
SD is the standard deviation of ARs in 10-Fold Cross-Validation. 1-5 means 5 times repeating test.
The Overall AR is the average annual ARs.

3.2 Results

The 10-Fold Cross-Validation splits a year data-set in 10 ways and produces 10 pairs of training and test observations, $\{(M_1, M_1'), \ldots, (M_{10}, M_{10}')\}$ (see Section 2.3). LR is applied to each pair of observations, so LR produces 10 ARs for each year. The AR in the tables is the annual AR, calculated by True results and all Predictions of 10-Fold Cross-Validation (see Section 2.3). The SD in tables means the standard deviation of 10 ARs from 10-Fold Cross-Validation. The overall AR is the average of five times repeat result. The standard deviation of ARs is used to measure the complexity of a model. For an over-complex model, the AR will be strongly affected by the training data and require a larger size of training data. On the other hand, a simple model is stable but more likely to lose information of data.

The results of LR is on unbalanced data shown in Table 14. The final AR is only 0.13%, which is a random guess and almost all predictions are non-default (0). After applying the Weighted-Loss method, see Section 2.5, the new result is shown in Table 7. The overall AR is 28.8% which is 200 times better than unbalanced data. The Average AR for each year shows a declining trend from 2008 to 2015.

However the average SD in Table 7 has a relatively high value of 0.1508 in 2008, and it fast decrease to the half, 0.0890, in the next year. Also, the average SD only have small change from 2009 to 2015 (from 0.0890 to 0.0624). In 2008 even though it has the highest AR but it varies significantly more than the other years, which means the predictions of 2008 is strongly affected by the split of K-Fold Cross-Validation. Therefore, LR has an unstable prediction for 2008 which only has a small size of default observation, see Table 4.

4 Multi-Layer Perceptron

4.1 Description

Multi-Layer Perceptron (MLP) is one of the most frequently used Neural Networks, all research in Section 0.2 used MLP. Similar to LR, MLP can find a surface which divides the observation space into two classes, but, instead of the hyper-plane, MLP can find a non-linear surface. The primary training process, the Backward Propagation method, first described by Rumelhart, Hinton and Williams [36] at 1986, let the large data-set training become possible. In addition, MLP has been proved it can approximate any function [23].

MLP contains three types of layers:

(i) an Input layer whose nodes representing observation variables,

(ii) an Output layer whose nodes representing the prediction,

(iii) one or two Hidden layers used to catch the non-linearity of the data, see Section 4.4,

and each layer contains several neurons. From the Input layer to the Output layer, neurons in each layer only pass information to the neurons in the next layer with the individual weight. The primary process of MLP training is the initialisation of the whole network, the **Forward Activation** and the **Backward Propagation**, see Section 4.2. After the initialisation of weights and biases randomly, the model uses the Forward Activation to produce prediction results and uses the Backward Propagation to adjust all weights during the training.

The Figure 3 is an example of MLP with one Input layer, one Hidden layer, and one Output layer. For each observation, this model takes three input variables and produces one dependence variable as output. Each link between neurons represents a weight, i.e., the weight between the Input layer and the Hidden layer is a 3×2 matrix, $W^{[1]}$, the weight between the Hidden layer and the Output layer is a 2×1 matrix, $W^{[2]}$. Besides, there is a bias, $b^{[1]}$, between the Input and the Hidden layer and a bias, $b^{[2]}$, between the Hidden layer and the Output layer.

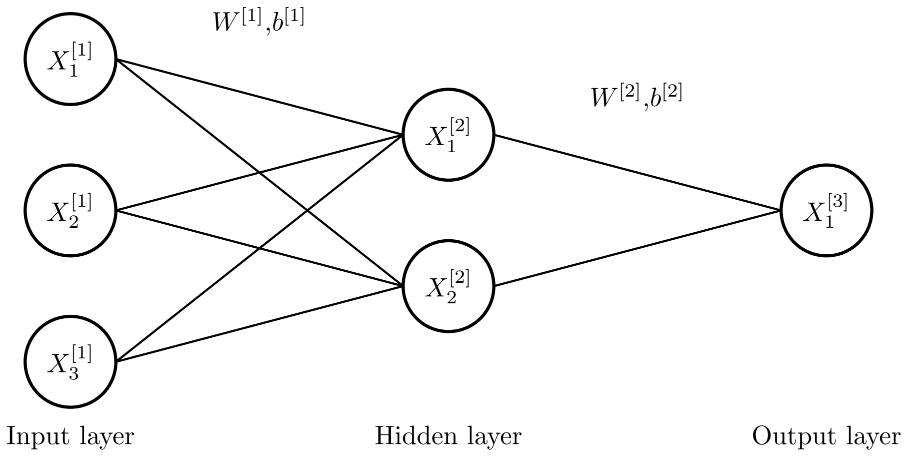

Figure 3: An example structure of MLP.

4.2 Main Processes

The prediction process of MLP is the **Forward Activation**, in which neurons of the Input layer take information from each variable of a observation. The model propagated all information forward through links to the Output layer. In each layer, two processes are applying the information. First, all information from the previous layer will be weighted and summed up with the bias of the layer as a linear function. Secondly the result is processed by the activation function and then passes to all linked neurons.

Assume a MLP model has same structure (3-2-1) as Figure 3, then the weights and biases of the model are

$$W^{[1]} = \begin{bmatrix} w^{[1]}_{1,1} & w^{[1]}_{1,2} \\ w^{[1]}_{2,1} & w^{[1]}_{2,2} \\ w^{[1]}_{3,1} & w^{[1]}_{3,2} \end{bmatrix}, \quad W^{[2]} = \begin{bmatrix} w^{[2]}_{1,1} \\ w^{[2]}_{2,1} \end{bmatrix}, \quad b^{[1]}, \quad b^{[2]}$$

To fit n three-dimensional observations, the MLP model will have three-dimensional variables in the Input layer, two-dimensional variables in the Hidden layer and one dimensional variables in the Output layer:

$$X^{[1]} = \begin{bmatrix} x^{[1]}_{1,1} & x^{[1]}_{1,2} & x^{[1]}_{1,3} \\ \vdots & \vdots & \vdots \\ x^{[1]}_{n,1} & x^{[1]}_{n,2} & x^{[1]}_{n,3} \end{bmatrix}, \quad X^{[2]} = \begin{bmatrix} x^{[2]}_{1,1} & x^{[2]}_{1,2} \\ \vdots & \vdots \\ x^{[2]}_{n,1} & x^{[2]}_{n,2} \end{bmatrix}, \quad X^{[3]} = \begin{bmatrix} x^{[3]}_{1,1} \\ \vdots \\ x^{[3]}_{n,1}. \end{bmatrix}$$

Then values of the Hidden layer is

$$x^{[2]}_{i,j} = \sigma\left(\sum_k w^{[1]}_{k,j} x^{[1]}_{i,k} + b^{[1]}\right)$$

where σ is the activation function. One common used activation function is the Sigmoid function, $\sigma(Z) = \frac{1}{1+exp(-Z)}$. $X^{[3]}$ is calculated by a similar function,

$$x^{[3]}_{i,j} = \sigma\left(\sum_k w^{[2]}_{k,j} x^{[2]}_{i,k} + b^{[2]}\right),$$

which is the prediction of $X^{[1]}$.

The training process of MLP is the **Backward propagation**. For each observation of the Training data-set, the model produces a prediction by the Forward Activation. There is a Loss function to measure how close from one prediction to its True label (1 or 0). The model takes the mean of all loss as the cost function. To achieve a minimal cost, the gradient descent method updates all parameters backward, see Section 2.4. One of the often used Loss function is the log loss function,

$$L(\hat{Y}, Y) = -Y \log \hat{Y} - (1 - Y)\log(1 - \hat{Y}),$$

where Y is the true label and $\hat{Y}$ is the prediction (See LR log likelihood, Equation (5)). If the Activation function is the Sigmoid function, then the first derivative of the cost respective to the second weights and the second bias are

$$dW^{[2]} = \frac{1}{n}\sum_i X^{[2]T}_i (X^{[3]}_i - Y_i), \quad db^{[2]} = \frac{1}{n}\sum_i (X^{[3]}_i - Y_i),$$

where n is the number of observations. This section only takes the GD method as a example (see Section 2.4 for more methods). The updated weights and bias are

$$\hat{W}^{[2]} = W^{[2]} + \eta dW^{[2]} \quad \text{and} \quad \hat{b}^{[2]} = b^{[2]} + \eta db^{[2]}$$

respectively, where η is the learning rate. Then the first derivative of the cost respective to first weights and first bias are

$$\begin{aligned} dW^{[1]} &= \frac{1}{n}\sum_i X_i^{[1]T}\left(\sigma'(X_i^{[1]}W^{[1]} + b^{[1]})W^{[2]}(X_i^{[3]} - Y_i)^T\right), \\ db^{[1]} &= \frac{1}{n}\sum_i \sigma'(X_i^{[1]}W^{[1]} + b^{[1]})W^{[2]}(X_i^{[3]} - Y_i)^T \end{aligned} \tag{6}$$

respectively, where σ' is the first derivative of the Sigmoid function, $\sigma'(x) = \frac{\exp(-x)}{(\exp(-x)+1)^2}$. It is worth to mansion that apply a function on a matrix means apply the function on each entry of the matrix. Similar as before, the updated weights are $\hat{W}^{[1]} = W^{[1]} + \eta dW^{[1]}$, and the updated bias is $\hat{b}^{[1]} = b^{[1]} + \eta db^{[1]}$. Repeat updating all weights and biases in next layer until reach the Input layer. The process of updating all weights and biases once is a training loop of Backward propagation. The MLP training process keeps running until the MLP get a minimal cost on the Validation data-set.

4.3 Activation Functions

There are five different Active functions introduced by *Tricks of the Trade* [22]: Sigmoid function, tanh function, ReLU function, Leaky ReLU function and a designed tanh function (shown in Figure 4).

The most typical activation function is the Sigmoid function $\sigma(Z) = \frac{1}{1+exp(-Z)}$, which can map any real numbers to the segment of (0,1). The result is often treated as a probability of default. However, it has been proved that centralised data (zero mean) has faster converge speed [22] for deep training. The results of the Sigmoid function can only be positive which is non-centralised. The tanh function is a better choice than the Sigmoid function because the tanh function maps all real numbers to (-1,1) and with the maximum gradient at zero. The values in the Output layer can be adapted to probabilities of default by function $g(x) = \frac{x+1}{2}$. Comparing with the Sigmoid function, the processed information from the tanh function is closer to zero. *Tricks of the Trade* [22] recommended way is only to use the Sigmoid function in the Output layer and use the tanh function during the perception.

Both of the Sigmoid function and the tanh function have two horizontal asymptotic lines. When the prediction is close to the target value (True label), the gradient of two activation function will tend to zero. Suppose the Loss function is $L(Y_i, \hat{Y}_i)$, where $\hat{Y}_i = \sigma(X_iW + B)$ is the probability of prediction, Y_i is the True label and σ is the activation function. The change of W in each training loop is

$$\Delta W = \frac{\alpha}{n}\sum_i^n \frac{\partial L(Y_i, \hat{Y}_i)}{\partial W} = \frac{\alpha}{n}\sum_i^n \frac{\partial L(Y_i, \hat{Y}_i)}{\partial \hat{Y}_i}\sigma'(X_iW + B)X_i,$$

where α is the learning rate, n is the number of observations and σ' is the first derivative of the activation function. If $XW + B$ are huge or negatively huge, $\sigma'(XW + B)$ will tend to zero and slow down the training speed. One of the solutions is setting target values to be in the range and let them be reachable. To avoid the activation function is too linear, the best target values are the maximal and minimal points of the second derivative. To keep same target values, $f(x) = 1.7159\tanh(\frac{2}{3}x)$ is a good choice of the activation function [22].

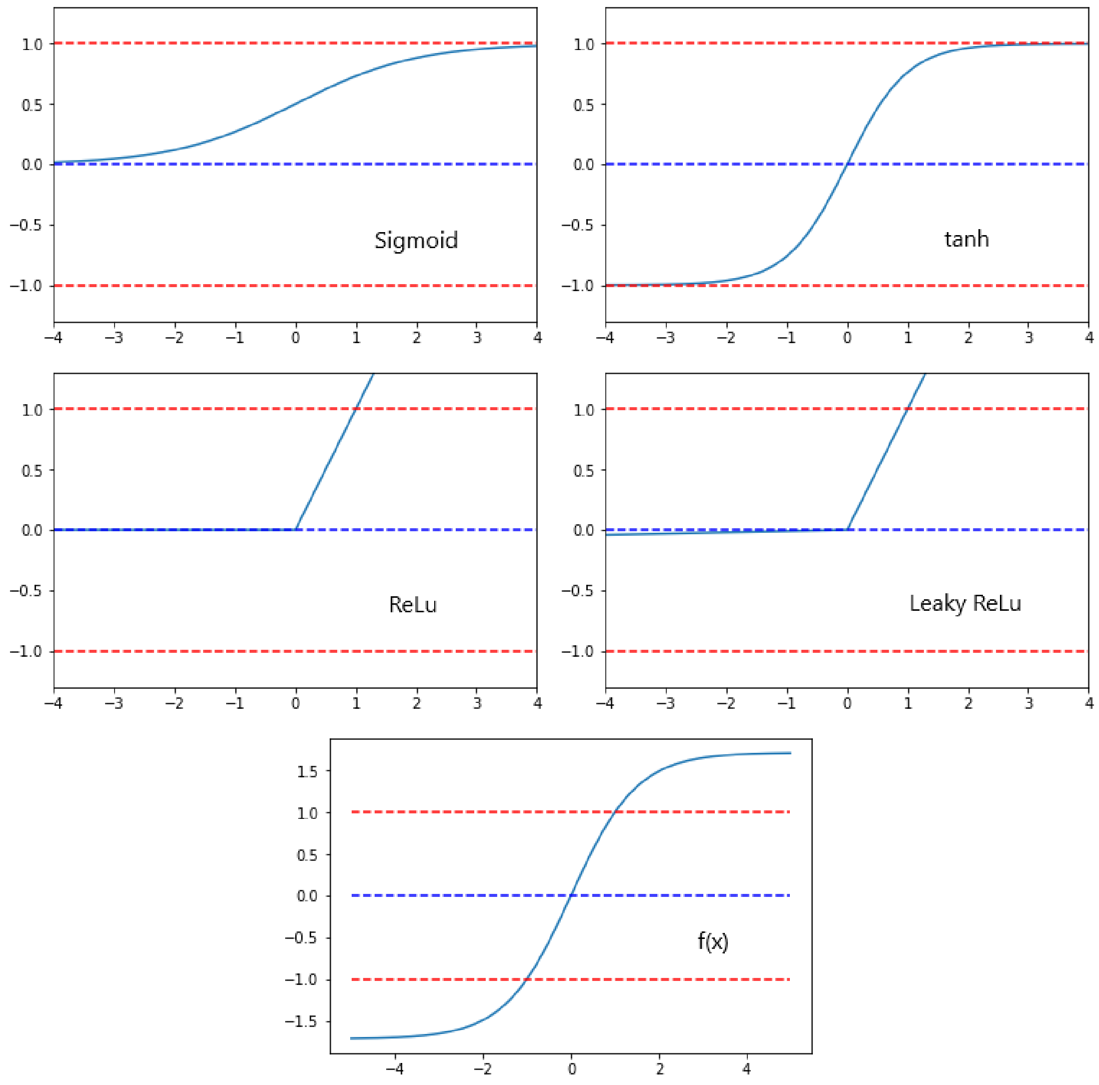

Figure 4: Five distinct Active Function. The red lines are $\{(x, y)|y = \pm1\}$.

Another choice of the activation function is the ReLU function, $\sigma(Z) = \max(0, Z)$. The ReLU function can keep the same gradient for all positive input. So the training speed will not decrease while the prediction is approaching the True label. However, the negative input will have zero gradients. Taking special initial values for weights and biases can fix this problem. Based on the ReLU function, the Leaky ReLU function, $\sigma(Z) = \max(0.01Z, Z)$ keeps the benefit of the ReLU function and still has a small gradient for negative input. The 0.01 can be altered to a larger number to procure quick converge for negative input, but it might cause more local minimal and mislead the training result.

To compare the performance of Activation functions, all five activation functions are applied on a single two-hidden-neurons hidden layer MLP with 2008 data, and its results are listed in Table 8. The Cost is the limit of cost after 50,000 training loops, and the Loop is the minimal loop

ID	Sigmoid		tanh		ReLu		Leaky ReLu		f(x)	
	Cost	Loop	Cost	Loop	Cost	Loop	Cost	Loop	Cost	Loop
1	0.5668	6927	0.5543	6848	0.5361	4990	0.5546	9521	0.5076	9544
2	0.5378	7363	0.5331	7609	0.5679	4934	0.4807	9877	0.5190	6913
3	0.4912	9639	0.5145	5876	0.4896	10046	0.6030	2112	0.5261	6742
4	0.5160	7396	0.4723	9943	0.5757	9032	0.5615	9856	0.5003	8176
5	0.5123	8015	0.5600	6211	0.5533	5684	0.5120	9458	0.5833	9088
6	0.5437	6857	0.5623	6603	0.5142	1349	0.5368	9985	0.5488	6175
7	0.5391	6284	0.5497	6233	0.5559	6778	0.5053	5481	0.5713	7986
8	0.5722	6642	0.5477	9511	0.5749	9996	0.5099	9999	0.5152	9036
9	0.5528	6748	0.5303	5266	0.5077	8207	0.5040	7560	0.5843	4523
10	0.5139	7280	0.5390	5241	0.4820	10000	0.4985	9985	0.5670	7803
Average	0.5346	7315.1	0.5363	6934.1	0.5357	7101.6	0.5266	8383.4	0.5423	7598.6

Table 8: Activation functions comparison. Cost is the final cost after $50,000$ training loops. Loop is the minimal training loop required to achieve a $\pm 1\%$ accurate cost.

required to reach the cost with only $\pm 1\%$ different with the limit of cost. The minimal average cost, 0.5266, is made by the Leaky ReLU function, which is only 0.0159% less than the average of all cost, 0.5351, hence the choice of activation function cannot significantly improve the accuracy of results. However, the minimal average loop, which is made by the tanh activation function, is 7.1% less than the average loop of all activation function, 7466.2. Therefore the tanh activation function is the most suitable function for this 14 dimensions simple data.

4.4 Hidden Layer Structure

During the building of an MLP structure, the number of neurons in the Input layer depends on the dimension of observations, and the number of neurons in the Output is one or two. The most significant questions are what the number of Hidden layer and numbers of neurons for each layer are. Many pieces of research talked about this question, but none of them have a precision enough conclusion.

"As many hidden nodes as dimensions needed to capture 70-90% of the variance of the input data-set." [27] is one of most acceptable opinions. The dimension of the hidden layer will not larger than input layer. Besides, after checking the value of weights, high correlative hidden neurons can be reduced. The left number of hidden neurons is the most suitable size. Except for this way, some researches suggests that the number of hidden neurons should be less than double size of input neurons [28], or be between the numbers of output neurons and input neurons [29].

In this thesis, only 14 fields are chosen as the input neurons. The hidden neuron should be less than the number of input neurons. The test results from 1 hidden neurons to 14 neurons are listed in Figure 5. The best result is "3" which means one Hidden layer with three hidden neurons. Some further two Hidden layer analysis results are also included in the Figure 5, but they do not improve accuracy. Therefore, one Hidden layer is enough for the data-set.

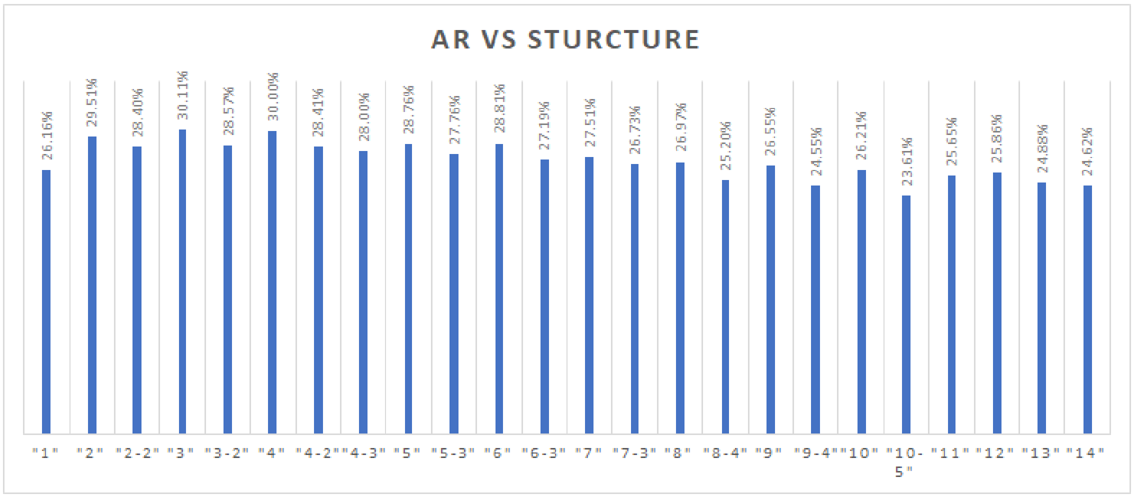

Figure 5: The final ARs of different MLP structure with the tanh Activation function and the AGD method on balanced data from 2008 to 2015.
"5-3" means 2 hidden layers with 5 neurons at first and 3 neurons at end.

4.5 Results

MLP is an advanced method of LR, each neuron of MLP is a LR model. As a balanced data can improve LR significantly (see Section 3.2), the MLP training uses Weighted-Loss on the Training data (see Section 2.5). The Section 4.3 proves the tanh function is the most suitable activation function. The Section 2.4 shows AGD is the best global minimal approaching method. After testing MLP with different Hidden layers in Section 4.4, the best structure is a 3-neurons Hidden layer. The Table 9 lists the optimal result of MLP. The model has near or over 30% ARs from 2009 to 2013. The standard deviation is low for each year. Although the average annual AR of 2008 (Table 9) is less than LR (Table 7), MLP has a better final AR and lower fluctuation in overall.

Year		2008	2009	2010	2011	2012	2013	2014	2015	Overall AR
1	AR	30.2%	34.9%	29.1%	31.6%	31.2%	31.7%	28.7%	29.9%	30.9%
	SD	0.1212	0.0778	0.0857	0.0691	0.0609	0.0543	0.0896	0.0516	
2	AR	29.1%	35.2%	29.4%	27.9%	32.1%	31.7%	27.7%	29.3%	30.3%
	SD	0.1216	0.0903	0.0678	0.0648	0.0710	0.0423	0.0510	0.0515	
3	AR	22.2%	31.1%	29.4%	31.3%	30.4%	30.5%	30.9%	27.5%	29.2%
	SD	0.0621	0.0694	0.0603	0.0735	0.1302	0.0602	0.0942	0.0529	
4	AR	29.3%	27.8%	30.6%	31.9%	29.2%	34.0%	27.2%	30.2%	30.0%
	SD	0.1504	0.0573	0.0578	0.0641	0.0828	0.0691	0.1132	0.0451	
5	AR	30.1%	31.1%	30.9%	31.4%	29.4%	33.5%	28.3%	26.4%	30.1%
	SD	0.0921	0.1009	0.0575	0.1037	0.0584	0.0598	0.0792	0.0771	
Average AR		28.2%	32.0%	29.9%	30.8%	30.4%	32.3%	28.5%	28.7%	30.1%
Average SD		0.1095	0.0791	0.0658	0.0750	0.0807	0.0571	0.0854	0.0556	

Table 9: 14-3-1 MLP with the tanh activation function and AGD.
SD is the standard deviation of ARs in 10-Fold Cross-Validation. 1-5 means 5 times repeating test.
The Overall AR is the average annual AR.

5 Probabilistic Neural Network

5.1 Description

Probabilistic Neural Network (PNN) is a nonlinear classifier statistic model which is based on the probability distribution function (PDF) and Bayesian theorem. Since Specht develops PNN at 1988 [30], PNN is a typical example of non-linear Neural Networks in credit scoring.

PNN has four layers:

(i) an Input layer whose nodes representing observation variables,

(ii) an Pattern layer which absorbs information from the distance between observations,

(iii) an Summation layer which produces the averages of groups in the Pattern layer,

(iv) an Output layer whose nodes representing the prediction.

All input information propagates from the Input layer to the Output layer following the network. Comparing with MLP, PNN has a more regular network structure. The Summation layer has the same number of neurons as the possible types of the label, and each neuron in the Pattern layer only connects one of the Summation layer's neuron.

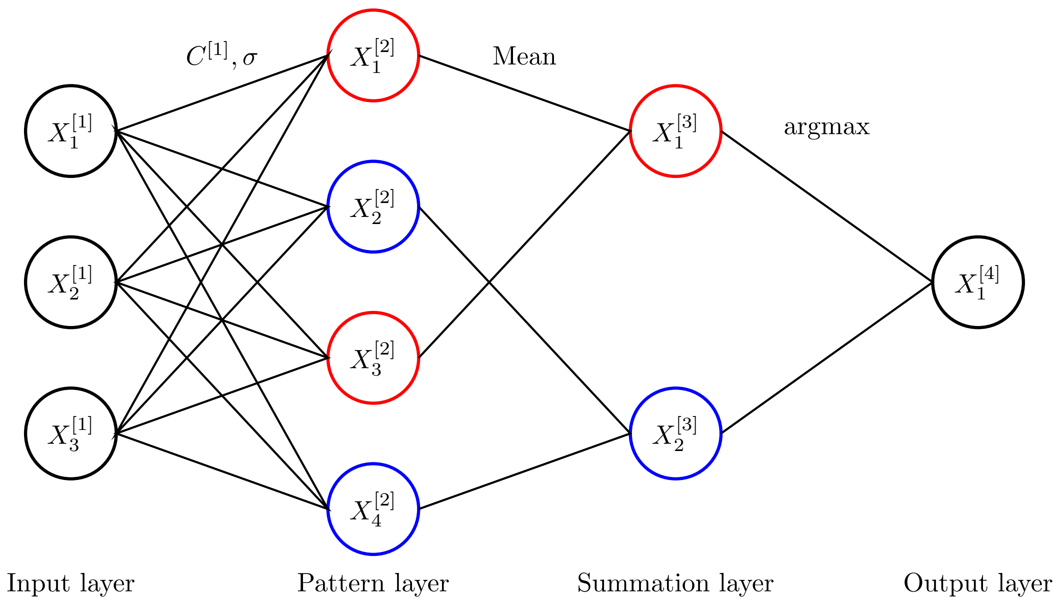

Figure 6: An example structure of PNN.

The Figure 6 is a simple PNN example. The Input layer has three neurons for three dimensions of observations, $X^{[1]}$, and the Input layer connect with the Pattern layer by a bias σ and centres, $C^{[1]}$, which is a 4×3 matrix. In the example, the Pattern layer contains four neurons which relate to four distinct centres which must have the same dimensions as observations. For a small data-set, it is common to use the Training data-set to be the centres. All neurons in the Pattern later are separated into certain groups. Any two neurons belong to a same group if and only if their related centres have same labels. Each group links with an individual neuron of the Summation layer, hence each neuron in the Summation layer has a distinct label. Finally, both of summation neurons pass their information, which is the mean of output from all linked pattern neurons, to the output neuron and the label from the highest-valued summation neuron is the output.

5.2 Main Processes

Instead of the linear function of observations and weights, the prediction process of PNN absorbs information from a distance between observations and centres. Suppose a PNN has the same structure as the example in Figure 6 and the Training data-set has n observations. Then the Input layer, centres and centres' labels are

$$X^{[1]} = \begin{bmatrix} x_1^1 & x_2^1 & x_3^1 \\ \vdots & \vdots & \vdots \\ x_1^n & x_2^n & x_3^n \end{bmatrix}, \quad C^{[1]} = \begin{bmatrix} c_1^1 & c_2^1 & c_3^1 \\ c_1^2 & c_2^2 & x_3^2 \\ c_1^3 & c_2^3 & x_3^3 \\ c_1^4 & c_2^4 & x_3^4 \end{bmatrix}, \quad Y = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

respectively. The first and third centres have label 1 and the rest centres have label 0, so $X_1^{[2]}$ and $X_3^{[2]}$ are in one group, $X_2^{[2]}$ and $X_4^{[2]}$ are in another group. The distance between ith observation, $X^i = [x_1^i, x_2^i, x_3^i]$, and jth centre, $C^j = [c_1^j, c_2^j, c_3^j]$, is

$$\|X^i - C^j\| = \sqrt{(x_1^i - c_1^j)^2 + (x_2^i - c_2^j)^2 + (x_3^i - c_3^j)^2}. \tag{7}$$

For the ith observation, the results of jth pattern neuron is

$$X_{ij}^{[2]} = \exp\left(-\frac{||X_i - C_j||^2}{\sigma^2}\right). \tag{8}$$

Hence, the first summation neuron is $X_{i1}^{[3]} = \frac{1}{2}X_{i1}^{[2]} + \frac{1}{2}X_{i3}^{[2]}$ with label 1. If $X_{i1}^{[3]}$ is larger than $X_{i2}^{[3]}$, then the prediction of ith observation is 1, otherwise its prediction is 0.

The training process of PNN is more complex than MLP as the Output layer of PNN contains the argmax function which is not differentiable, hence the cost of PNN is not differentiable as well. Gradient descent methods are not suitable for cost reduction, and the only ways are **Grid search** and **Random search**. **Grid search** lists all or most values for each parameter and tests on each of possible combination one by one to find the best parameters which optimise the result. **Random search** randomly sets each parameter in each training loop. After a certain number of loops, the parameters which get the best result is the optimiser. Therefore, Random search and Grid search are slow for continuous multi-dimension space and might miss the optimal value.

The first way of speed up the training process is to reduce parameter dimensions. According to Donald F. Specht's paper *Probabilistic Neural Networks* [30], PNN assumes all dimension of observations are individual, and each of default probability and non-default probability is following a mean of multiple Gaussian distributions, see example in Figure 7. For a given mean, μ, and standard deviation, σ, the Probability density function (PDF) of Gaussian distribution is

$$PDF(X_i) = \frac{1}{\sqrt{2\pi||\sigma||^2}} \exp\left(-\frac{||X_i - \mu||^2}{2\sigma^2}\right) = \alpha X_{ij}^{[2]},$$

where α is some constant which does not affect the output result of the network. From the Figure 7, the mean of PDF is likely to have high value at the means of each Gaussian distribution. To be able to achieve a low cost on training data, all training observations are picked as centres in the Pattern layer, so σ is the only parameter required for training. As the model assume PDFs of default and non-default are continuous, and σ should be a finite positive number, **Bisection** method can be used to find a best σ which minimise the cost of the Training data-set (see Appendix A for Bisection method).

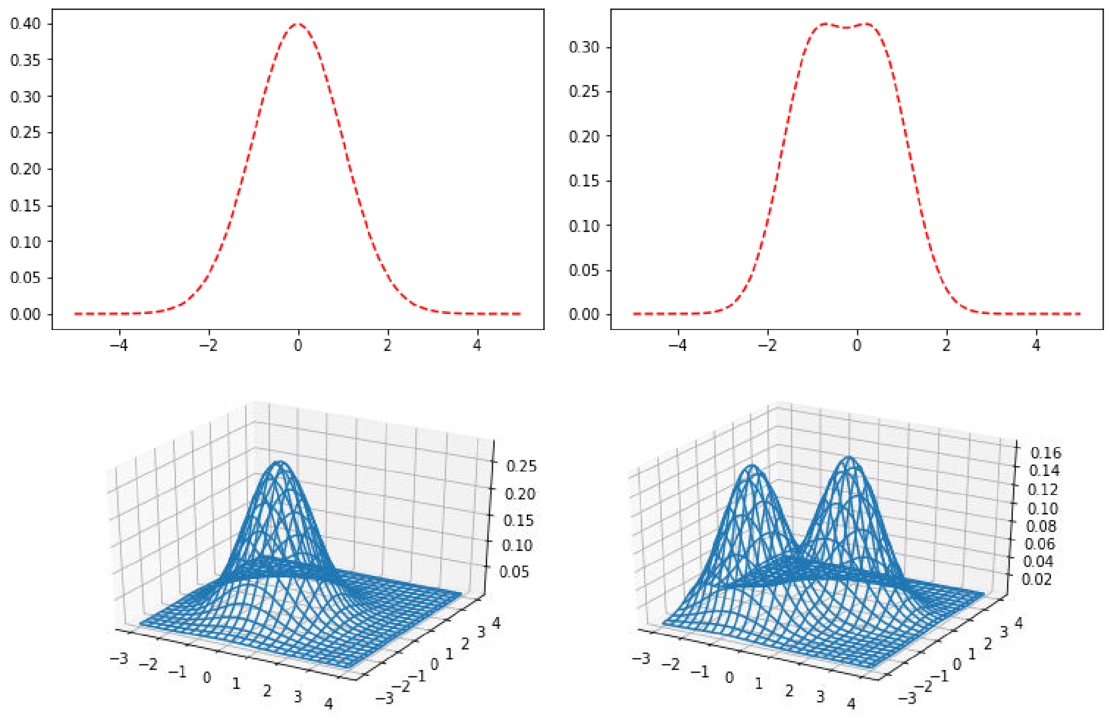

Figure 7: The top-left is the PDF of 1-dimensional Gaussian distribution with mean 0 and standard deviation 1, (0,1).
The top-right is the mean of PDFs of two 1-dimensional Gaussian distribution with parameters (0.5,0.7) and (-1,0.7) respectively.
The bottom-left is the PDF of 2-dimensional Gaussian distribution with mean [0,0], standard deviation [1,1]. Each number of [1,1] represents a standard deviation in one dimension.
The bottom-right is the mean of PDFs of two 2-dimensional Gaussian distribution with parameters ([1,1],[1,0.7]) and ([-1,-1],[1,0.7]) respectively.

5.3 Dimension of Variance

Different fields might have different variance, and values in the small variance field are more close to each other. A shorter distance will produce a more considerable value to the neurons of the Pattern layer. So those fields will be more important than a large variance field. To get a more accurate result, different dimensions of the data-set should have an individual σ in Equation (8). However, Bisection method is only usable on a function with 1-dimensional domain. The **boundary reduction** method can achieve similar work on a multi-dimensional variable, see Appendix A.

For a list of variables and their losses, the **boundary reduction** method assumes that the point with the minimal loss, X_0, and the point with the second minimal loss X_1 should be closest to the global minimum point. In every training loop, the boundary reduction method randomly picks a new variable from the circle which has centre X_0 and radius $||X_0 - X_1||$. The new random variable is joined into the list, and the whole process will repeat until its minimal loss reaches the convergence.

Table 15 (in Appendix D) shows the five times repeating test results with the global one dimensional σ. It has the best result 23.1% in 2009, the worst result 15.1% in 2012 and the final AR is 19.7%. Similar to the global σ, the local σ has maximal AR in 2009 and minimal AR in 2012 with values 29.3% and 20.5% respectively, shown in Table 10. The overall AR increase to the 23.8%, hence increase the dimension of σ can get definite improvement.

Note, every year's result gets some amount of improvement. Therefore, increase the dimension of σ only has positive effect to PNN model, and there is no reason to use the global σ.

5.4 Results

The best result of PNN is shown in Table 10. The best annual AR is 29.3% and 25.5% in 2009 and 2015 respectively, and the worst annual AR is in 2012 with 20.5%. From the table, there is no relation between the annual AR and data size. One of the reason is that PNN absorbs information from a distance between observations, so the AR is affected by the aggregation of observations more.

Year		2008	2009	2010	2011	2012	2013	2014	2015	Overall AR
1	AR	23.8%	27.9%	21.5%	21.0%	18.2%	22.0%	22.4%	24.9%	22.7%
	SD	0.1449	0.1092	0.0705	0.0482	0.0980	0.0626	0.0501	0.0721	
2	AR	19.7%	29.0%	22.8%	22.5%	18.2%	23.3%	24.4%	26.1%	23.2%
	SD	0.1321	0.0820	0.0619	0.0962	0.0710	0.0628	0.0614	0.0737	
3	AR	18.7%	27.9%	23.0%	24.6%	22.1%	25.7%	22.8%	26.5%	23.9%
	SD	0.0836	0.0785	0.0771	0.0785	0.0766	0.0695	0.0498	0.0500	
4	AR	22.2%	28.1%	22.8%	22.9%	22.4%	25.5%	20.0%	26.0%	23.7%
	SD	0.1001	0.1195	0.0713	0.0630	0.0927	0.0482	0.0741	0.0670	
5	AR	22.6%	33.6%	27.7%	25.0%	21.4%	23.7%	25.1%	24.0%	25.4%
	SD	0.1250	0.0845	0.0854	0.0799	0.0659	0.0687	0.0457	0.0661	
Average AR		21.4%	29.3%	23.6%	23.2%	20.5%	24.0%	22.9%	25.5%	23.8%
Average SD		0.1172	0.0947	0.0732	0.0809	0.0624	0.0571	0.0562	0.0658	

Table 10: PNN results with default-balanced data and local multi-σ.
SD is the standard deviation of ARs in 10-Fold Cross-Validation. 1-5 means 5 times repeating test.
The Overall AR is the average annual ARs.

Comparing with the LR best result in Table 7, it is worth mentioning that although the LR's final AR has 5% higher than the PNN's, the PNN has better AR than the LR in years 2009 and 2008. LR is the better statistic model but, in some condition, PNN is more accurate than LR.

6 Radial Basis Function Network

6.1 Description

Radial Basis Function Network (RBFN) is another commonly used artificial Neural Networks and is first formulated by Broomhead and Lowe [31] in 1988. Similar to PNN, RBFN absorbs information from the distance of observations, which is more convenient to find the non-linear boundary.

RBFN has one Input layer, one Hidden layer and one Output layer. Similar as PNN, all input information propagates from the Input layer to the Output layer following the network, and the number of neurons in the Hidden layer depended on the number of centres which must have same dimensions as observations. In the prediction process, all information from the Hidden layer will be processed through a linear function of weights and bias. Finally, a Sigmoid function changes the linear result to the probability of default.

A simple 4-centres RBFN example is shown in Figure 8. It has three neurons in the Input layer and one neuron in the Output layer. It means the RBFN model can predict three-dimensional observations and get one-dimensional predictions. Also four neurons in the Hidden layer means the RBFN has four three-dimensional centres, $C^{[1]}$, and a 4×1 matrix weight, $W^{[2]}$.

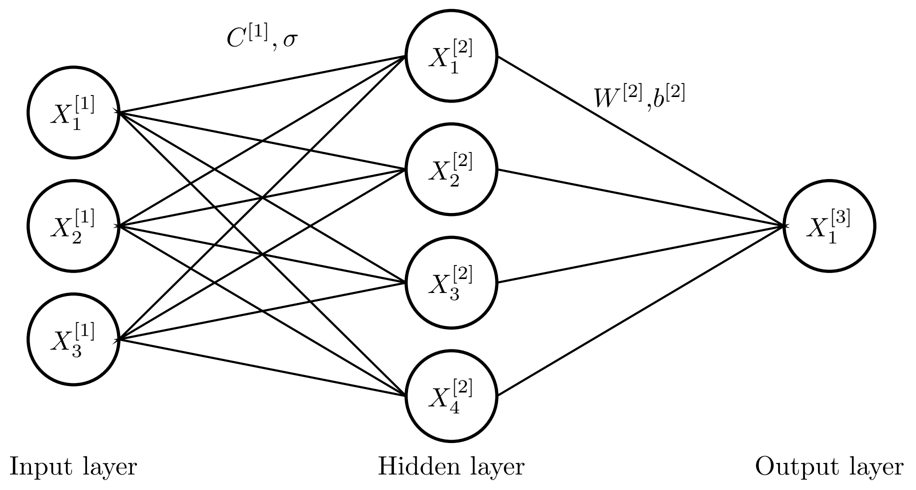

Figure 8: An example structure of RBFN.

6.2 Main Process

A simple 4-centres RBFN example are shown in Figure 8. Suppose the Training dataset and four centres are

$$X^{[1]} = \begin{bmatrix} x_1^1 & x_2^1 & x_3^1 \\ \vdots & \vdots & \vdots \\ x_1^n & x_2^n & x_3^n \end{bmatrix}, \quad C^{[1]} = \begin{bmatrix} C_1^{[1]} \\ C_2^{[1]} \\ C_3^{[1]} \\ C_4^{[1]} \end{bmatrix} = \begin{bmatrix} c_1^1 & c_2^1 & c_3^1 \\ c_1^2 & c_2^2 & x_3^2 \\ c_1^3 & c_2^3 & x_3^3 \\ c_1^4 & c_2^4 & x_3^4 \end{bmatrix}$$

respectively. Then the values of the Hidden layer, $X^{[2]}$, is a $n \times 4$ matrix. The jth value of ith row of the matrix is

$$X_{ij}^{[2]} = \Phi\left(\frac{||X_i^{[1]} - C_j^{[1]}||}{\sigma}\right),$$

where σ is a 1-dimensional parameter and Φ is the activation function of the Hidden layer which is common to use either Gaussian form $\Phi(x) = \exp(-x^2)$ or a multiquadric $\Phi(x) = \sqrt{c^2 + x^2}$, [31]. $\|\cdot\|$ is shown in Equation (7). The prediction of ith observation from the paper [31] is

$$X_i^{[3]} = X_i^{[2]} W^{[2]} + b^{[2]}. \tag{9}$$

where $W^{[2]}$ is the weight, $b^{[2]}$ is the bias and $X_i^{[2]}$ is the ith row of $X^{[2]}$. As True labels of data-set are only default (1) and non-default (0), this thesis uses

$$X_i^{[3]} = \phi(X_i^{[2]} W^{[2]} + b^{[2]}), \tag{10}$$

where ϕ is the Sigmoid function, as the prediction.

The original RBFN has a simple Training process [31]. Suppose there are n training observations, $X^{[1]} = X = [X_1, X_2, \ldots, X_n]$, and related labels, $Y = [Y_1, Y_2, \ldots, Y_n]$. The Loss function of RBFN, $L(Y_i, X_i^{[3]})$ can be the Mean Square Error function or the Log loss function. Same as PNN, setting centres $C^{[1]}$ to be training observations X can reduce the cost of training observations. According to the previous prediction function (see Equation (9)), the output of RBFN can be written as

$$\begin{bmatrix} X_1^{[3]} \\ \vdots \\ X_n^{[3]} \end{bmatrix} = \begin{bmatrix} \Phi\left(\frac{||X_1 - X_1||}{\sigma}\right) & \cdots & \Phi\left(\frac{||X_1 - X_n||}{\sigma}\right) \\ \vdots & \ddots & \vdots \\ \Phi\left(\frac{||X_n - X_1||}{\sigma}\right) & \cdots & \Phi\left(\frac{||X_n - X_n||}{\sigma}\right) \end{bmatrix} \begin{bmatrix} W_1^{[2]} \\ \vdots \\ W_n^{[2]} \end{bmatrix} + \begin{bmatrix} b^{[2]} \\ \vdots \\ b^{[2]} \end{bmatrix}. \tag{11}$$

By solving linear regression with Ordinary Least Square (OLS), the optimal weights and bias are

$$\begin{bmatrix} b^{[2]} \\ W^{[2]} \end{bmatrix} = \left(\begin{bmatrix} \underline{1} & \hat{\Sigma} \end{bmatrix}^T \begin{bmatrix} \underline{1} & \hat{\Sigma} \end{bmatrix}\right)^{-1} \begin{bmatrix} \underline{1} & \hat{\Sigma} \end{bmatrix}^T Y,$$

where $\hat{\Sigma}$ is the $n \times n$ matrix in the Equation (11) and $\underline{1}$ is $1 \times n$ matrix with 1 in each entries. σ is the only parameter which needs the gradient descent optimisation, see Section 2.4. The derivative of the cost function respective to σ is

$$\Delta\sigma = -\frac{1}{n} \sum_i^n \frac{\partial L(Y_i, X_i^{[3]})}{\partial X_i^{[3]}} \sum_j^n \Phi'\left(\frac{\|X_i - X_j\|}{\sigma}\right) W_j \frac{\|X_i - X_j\|}{\sigma^2}$$

where Φ' is the first derivative of Φ and $\frac{\partial L(Y_i, X_i^{[3]})}{\partial X_i^{[3]}}$ is the partial derivative of the Loss function respective to the prediction. This section only take the GD method as a example (see Section 2.4), the updated σ is $\hat{\sigma} = \sigma + \alpha\Delta\sigma$, where α is the learning rate.

However, this thesis uses the adapted RBFN model, i.e., the prediction is Equation (10), so the derivative of the cost function respective to σ changes to

$$\Delta\sigma = -\frac{1}{n} \sum_i^n \frac{\partial L(Y_i, X_i^{[3]})}{\partial X_i^{[3]}} \phi'(X_i^{[2]} W^{[2]} + b^{[2]}) \sum_j^n \Phi'\left(\frac{\|X_i - X_j\|}{\sigma}\right) W_j \frac{\|X_i - X_j\|}{\sigma^2}$$

where ϕ' is the first derivative of the Sigmoid function. Instead of the OLS, the weight $W^{[2]}$ have to be calculated by the gradient descent method. The derivative of the cost function respective to the $W^{[2]}$ is

$$\begin{aligned} \Delta W^{[2]} &= \frac{1}{n}\sum_i^n \frac{\partial L(Y_i, X_i^{[3]})}{\partial X_i^{[3]}}\phi'(X_i^{[2]}W^{[2]} + b^{[2]})X_i^{[2]T} \quad \text{and} \\ \Delta b^{[2]} &= \frac{1}{n}\sum_i^n \frac{\partial L(Y_i, X_i^{[3]})}{\partial X_i^{[3]}}\phi'(X_i^{[2]}W^{[2]} + b^{[2]}) \end{aligned} \tag{12}$$

Then weights, bias and σ are updated by the gradient descent method, see Section 2.4.

6.3 Relation with Gaussian Process Regression

In 1963, Gaussian Process Regression (GPR) was described for the first time by Georges Matheron [41]. For n given training observations, $\hat{X} = [\hat{X}_1, \dots, \hat{X}_n]^T$, and their related labels, $\hat{Y} = [\hat{Y}_1, \dots, \hat{Y}_n]^T$, GPR assumes the observation of data follows the standard linear regression model with a Gaussian noise ϵ, i.e.

$$\hat{Y} = f(\hat{X}) + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma_n^2)$$

where $\mathcal{N}(\mu, \nu)$ is a Normal distribution with mean μ and standard deviation ν, and $f(X) = XW + b$ is a linear function with weights W and bias b. The conditional PDF of Y is

$$P(\hat{Y}|\hat{X}, W) = \prod_{i=1}^{n} P(\hat{Y}_i|\hat{X}_i, W) = \frac{1}{(2\pi\sigma_n^2)^{n/2}} \exp\left(-\frac{\left\|\hat{Y} - \hat{X}W\right\|^2}{2\sigma_n^2}\right) \tag{13}$$

For a given inputs, X, the prediction is a Normal distribution [42]:

$$\begin{gathered} P(Y|X, \hat{X}, \hat{Y}) = \mathcal{N}(\Sigma^T\hat{\Sigma}^{-1}\hat{Y}, C(X, X) - \Sigma^T\hat{\Sigma}^{-1}\Sigma), \\ \Sigma = C(\hat{X}, X), \quad \hat{\Sigma} = C(\hat{X}, \hat{X}) \end{gathered}$$

where the jth value in ith row of $C(A, B)$ is

$$C(A, B)_{ij} = \left[\alpha \exp\left(-\frac{\|A_i - B_j\|^2}{2\sigma^2}\right) + a_0 + a_1 A_i^T B_j + \beta\delta(i, j)\right]_{ij}, \qquad \delta(i, j) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$$

where β is a small constant number to avoid non-invertible situation.

This prediction has high relation with the RBFN's prediction. Suppose the activation function $\Phi(\|A_i, B_j\|)$ of RBFN is $C(A, B)_{ij}$, then the $a_0 \sum_i^n W_i^{[2]}$ term is equivalent to the bias $b^{[2]}$. Let the centres to be training observations, the prediction becomes

$$X^{[3]} = \Sigma^T(\hat{\Sigma}^T\hat{\Sigma})^{-1}\hat{\Sigma}^T\hat{Y}.$$

If $\hat{\Sigma}$ is invertible, than $X^{[3]} = \Sigma^T\hat{\Sigma}^{-1}\hat{Y}$ is same as the mean of $P(Y|X, \hat{X}, \hat{Y})$.

The training process of GPR is to minimize the log likelihood [42],

$$l = -\frac{1}{2}\log \det\hat{\Sigma} - \frac{1}{2}\hat{Y}^T\hat{\Sigma}^{-1}\hat{Y} - \frac{n}{2}\log 2\pi$$

by gradient descent methods, see Section 2.4.

6.4 Recursive Orthogonal Least Squares Training and Centre Selectors

As the database has great size (over million data), it requires too much memory space (over 16 GB) to store every training observations as centres. Also, a significant number of neurons leads to a higher system complexity, which has a lower training speed. Gomm, J. Barry, and Ding Li Yu introduced Recursive Orthogonal Least Squares (ROLS) training and two ways of selecting centres [32] to reduce the number of centres.

Yu, D. L., J. B. Gomm, and D. Williams explained the ROLS method [34] based on QR decomposition. For a full rank $n \times m$ matrix Θ, there exists $n \times n$ orthogonal matrix Q such that

$$\Theta = Q \begin{bmatrix} R \\ 0 \end{bmatrix}, Q^T Q = QQ^T = I$$

and R is $n \times m$ upper triangle matrix with same rank as Φ.

Huang, De-Shuang, and Wen-Bo Zhao gave a more detail explanation in 2005 [35]. Let $Y(n) = [y_1, \dots, y_n]^T \in \mathbb{R}^{n\times l}$ be labels and $\Theta(n) = [\theta_1, \dots, \theta_n]^T \in \mathbb{R}^{n\times m}$ be the values of the Hidden layer. The final cost function can be written as

$$L(n) = \|Y(n) - \Theta(n)W(n)\|_F^2$$

where $\|X\|_F = \sqrt{\sum_i \sum_j X_{ij}^2}$ is the Frobenius norm, and the bias $b^{[2]}$ is transformed to be a constant term in θ. By using QR decomposition, it is able to get

$$\Theta = Q \begin{bmatrix} R \\ 0 \end{bmatrix}, Q^T Y = \begin{bmatrix} \hat{Y} \\ \bar{Y} \end{bmatrix} \tag{14}$$

Therefore, the cost function can be written as

$$\begin{aligned} L(n) &= \left\| Q(n) \begin{bmatrix} \hat{Y}(n) \\ \bar{Y}(n) \end{bmatrix} - Q(n) \begin{bmatrix} R(n) \\ 0 \end{bmatrix} W(n) \right\|_F^2 \\ &= \left\| \hat{Y}(n) - R(n)W(n) \right\|_F^2 + \left\| \bar{Y}(n) \right\|_F^2 \end{aligned} \tag{15}$$

where $\left\|\bar{Y}(n)\right\|_F^2$ represents the residual of the error cost function $L(n)$. If a new training observation is added into the data-set,

$$\begin{aligned} L(n+1) &= \|Y(n+1) - \Theta(n+1)W(n+1)\|_F^2 \\ &= \left\| \begin{bmatrix} Y(n) \\ y(n+1) \end{bmatrix} - \begin{bmatrix} \Theta(n) \\ \theta(n+1) \end{bmatrix} W(n+1) \right\|_F^2 \\ &= \left\| \begin{bmatrix} Q(n) \begin{bmatrix} \hat{Y}(n) \\ \bar{Y}(n) \end{bmatrix} \\ y(n+1) \end{bmatrix} - \begin{bmatrix} Q(n) \begin{bmatrix} R(n) \\ 0 \end{bmatrix} \\ \Theta(n+1) \end{bmatrix} W(n+1) \right\|_F^2 \\ &= \left\| \begin{bmatrix} \hat{Y}(n) \\ y(n+1) \\ \bar{Y}(n) \end{bmatrix} - \begin{bmatrix} R(n) \\ \theta(n+1) \\ 0 \end{bmatrix} W(n+1) \right\|_F^2 \\ &= \left\| \hat{Y}(n+1) - R(n+1)W(n+1) \right\|_F^2 + \|\bar{y}(n+1)\|_F^2 + \left\|\bar{Y}(n)\right\|_F^2 \end{aligned} \tag{16}$$

So R, Q, $\bar{Y}$, $\hat{Y}$ are updated by the following equations:

$$\begin{aligned} \begin{bmatrix} R(n) \\ \theta^T(n+1) \end{bmatrix} &= Q(n+1) \begin{bmatrix} R(n+1) \\ 0 \end{bmatrix} \\ \begin{bmatrix} \hat{Y}(n+1) \\ \bar{y}^T(n+1) \end{bmatrix} &= Q^T(n+1) \begin{bmatrix} \hat{Y}(n) \\ y^T(n+1) \end{bmatrix}, \end{aligned} \tag{17}$$

$W(n+1)$ can be calculated by $R(n+1)W(n+1) = \hat{Y}(n+1)$, and the new residual of error can be updated by $\left\|\bar{Y}(n+1)\right\|^2 = \left\|\bar{Y}(n)\right\|^2 + \|\bar{y}(n+1)\|^2 = L(n+1)$.

The size of R, $\hat{Y}$ are only related to the number of centres. The training complexity will linearly increase while the growing of the training data size.

One way of selecting network centres is **Backward selection** algorithm [32]. Suppose the jth centre is deleted, then the jth column of Θ is removed and similarly jth column of R, jth row of W are removed as well, $R_{!j} = [r_1, \ldots, r_{j-1}, r_{j+1}, \ldots, r_n]$, $W_{!j} = [w_1, \ldots, w_{j-1}, w_{j+1}, \ldots, w_n]^T$. The cost function becomes

$$L(!j) = \left\| \begin{bmatrix} \hat{Y} - R_{!j}W_{!j} \\ \bar{Y} \end{bmatrix} \right\|_F^2 = \left\| \begin{bmatrix} r_j w_j^T \\ \bar{Y} \end{bmatrix} \right\|_F^T = \left\|r_j w_j^T\right\|_F^2 + \left\|\bar{Y}\right\|_F^2. \tag{18}$$

The new optimal weight W_j can be calculated by

$$R_{!j} = Q_{!j} \begin{bmatrix} R_j \\ 0 \end{bmatrix}, \begin{bmatrix} \hat{Y}_j \\ \bar{y}_j^T \end{bmatrix} = Q_{!j}^T \hat{Y}, R_j W_j = \hat{Y}_j. \tag{19}$$

Then the new cost function will be

$$L(j) = \left\| \begin{bmatrix} Q_{!j} \begin{bmatrix} \hat{Y}_j \\ \bar{y}_j^T \end{bmatrix} - Q_{!j} \begin{bmatrix} R_j \\ 0 \end{bmatrix} W_j \end{bmatrix} \right\|_F^2 = \left\|\hat{Y}_j - R_j W_j\right\|_F^2 + \left\|\bar{y}_j^T\right\|_F^2 + \left\|\bar{Y}\right\|_F^2. \tag{20}$$

Therefore, the $\|\bar{y}_j\|_F^2$ is the residual of error caused by the deleting jth centre. The new residual of error is $\left\|\bar{Y}_j\right\|_F^2 = \|\bar{y}_j\|_F^2 + \left\|\bar{Y}\right\|_F^2 = L(j)$.

Gomm, J. Barry, and Ding Li Yu use Akaike's final prediction error (FPE) [33] as a stopping signal of the Backward selection algorithm [32]:

$$FPE = \frac{1 + \beta(n_p/N)}{1 - \beta(n_p/N)} V \tag{21}$$

where $V = \left\|\bar{Y}(N)\right\|_F^2 / (N)$, n_p is the dimension of network output, and β is a weighting factor. The paper suggest to use $\beta = 2$. The whole algorithm process is to find minimal point of FPE by deleting centres.

The other way of selecting centres is **Forward selection** algorithm [32]. Suppose the model already have the first $k-1$ centres and is looking for the kth centre. The upper triangle matrix is $R = [r_1, \ldots, r_k, \ldots, r_n]$, and the measure of information is $\hat{Y} = [\hat{y}_1, \ldots, \hat{y}_{n_p}^T]$ from $(k-1)$th centre search. For all $k < j \leq n$, $R_j = [r_1, \ldots, r_{k-1}, r_j, r_{k+1}, \ldots, r_{j-1}, r_{j+1}, \ldots, r_n]$. Then the new $\hat{Y}_j$ and W_j can be calculated by

$$R_j^* = Q_j R_j, \hat{Y}_j = \begin{bmatrix} \hat{Y}_j^1 \\ \hat{y}_{j,k}^T \\ \hat{Y}_j^2 \end{bmatrix} = Q_j^T \hat{Y}, R_{j(k)}^* W_k = \begin{bmatrix} \hat{Y}_j^1 \\ \hat{y}_{j,k}^T \end{bmatrix} \tag{22}$$

where R_j^* is upper triangle matrix has same size as R_j, $R_{j(k)}^*$ is the top $k \times k$ matrix of R_j^*, $\hat{Y}_j^1 = [\hat{y}_{j,1}, \ldots, \hat{y}_{j,k-1}]^T$ and $\hat{Y}_j^2 = [\hat{y}_{j,k+1}, \ldots, \hat{Y}_{j,n_p}]$. Therefore, the new cost function will be $V_j = (\left\|\hat{Y}_j^2\right\|_F^2 + \|\bar{Y}\|_F^2)/N$.

The Equation (14) gives $\|Y\|_F^2 = \left\|\hat{Y}\right\|_F^2 + \|\bar{Y}\|_F^2$, hence the maximal $\left\|\hat{y}_j\hat{y}_j^T\right\|_F^2$ can decide which centre will be the kth. The whole process keeps running until the FPE (see Equation (21)) reaches the minimum point.

6.5 K-Means Clustering Centre Selector

Another centre selector method is K-Means Clustering method which is famous in data analysis. K-Means Clustering can separate observations into k clusters, and each observation point belongs to the cluster whose centre is closest. Each centre is the mean of all observation points in its cluster, for example, three clusters are shown in Figure 9.

K-means clustering has four steps:

1. Initialise k centres by randomly picking k observations, $\{C_1, \ldots, C_k\}$.
2. Classify all observations into k clusters by finding the closed centre,

$$\text{group}(X) = \underset{i \in \{1,\ldots,k\}}{\arg\min} \|X - C_i\|.$$

3. Recalculate means for each cluster,

$$C_y = \underset{\text{group}(X)=y}{\text{mean}}(X), y \in \{1, \ldots, k\}.$$

Set the means to be new centres.

4. Repeat step 3 and step 4, until reaching a given number of iterations, no exchange of points between clusters or reaching a threshold value.

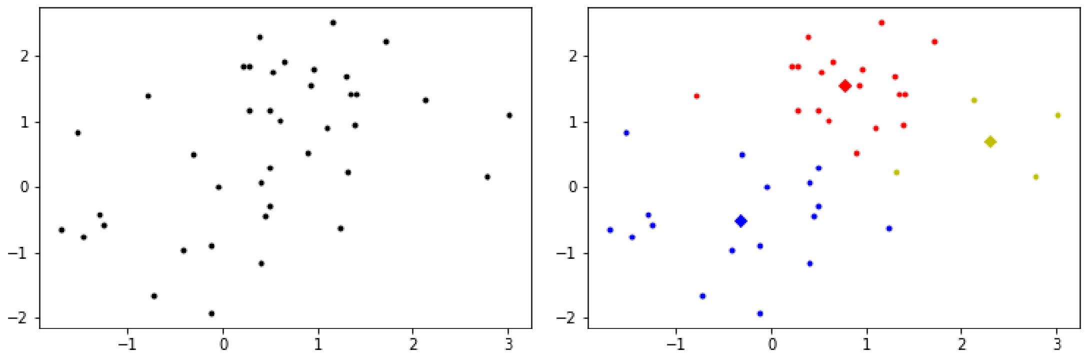

Figure 9: The left graph is some random points in a two dimensional space. The right graph is three clusters after applying K-means clustering on the points from the left graph. The diamond marks in the right graph are the centres of each cluster.

K-Means Clustering can reduce the number of non-default centres to k centres. As this thesis takes the activation function for the Hidden layer to be the Gaussian form, $\phi(x) = \exp(-\frac{x^2}{\sigma})$, the mean of few non-default centres, which are close to each other, is likely to be a local maximal point in the probability surface. Figure 10 is an example of probability surfaces. The left graph is

Clusters	2008	2009	2010	2011	2012	2013	2014	2015	Average AR
100	16.07%	15.29%	9.78%	15.83%	15.75%	24.15%	21.39%	17.54%	16.98%
200	**21.85%**	**21.33%**	**13.84%**	**18.32%**	**18.80%**	**26.29%**	23.23%	**20.12%**	**20.47%**
250	**21.47%**	**21.22%**	13.33%	16.93%	**17.21%**	25.70%	**23.51%**	**20.39%**	**19.97%**
300	19.78%	20.52%	12.49%	**19.48%**	16.65%	**26.14%**	**23.50%**	19.16%	19.71%
350	19.37%	18.59%	13.55%	18.15%	16.23%	25.22%	22.63%	19.79%	19.19%
400	19.42%	18.63%	**13.96%**	18.17%	16.82%	25.34%	22.43%	19.66%	19.30%
500	15.04%	17.68%	13.28%	17.47%	15.55%	23.80%	20.74%	19.57%	17.89%

Table 11: The AR of RBFN for different number of clusters. The best two ARs in each column are in bold.

the probability surface of three Gaussian distribution's mean. The means of those three Gaussian distributions are $[1, 0]$, $[\frac{\sqrt{3}}{2}, -\frac{1}{2}]$ and $[-\frac{\sqrt{3}}{2}, -\frac{1}{2}]$ whose average is $[0, 0]$. The mean of right side standard Gaussian distribution is $[0, 0]$. It is obvious the left probability surface has a maximal point at $[0, 0]$ and has a similar shape as the right probability surface (standard Gaussian distribution). The only difference between the two surfaces is the sharpness, hence it is possible to use the centre of a cluster representing all RBFN's centres in this cluster without losing most information.

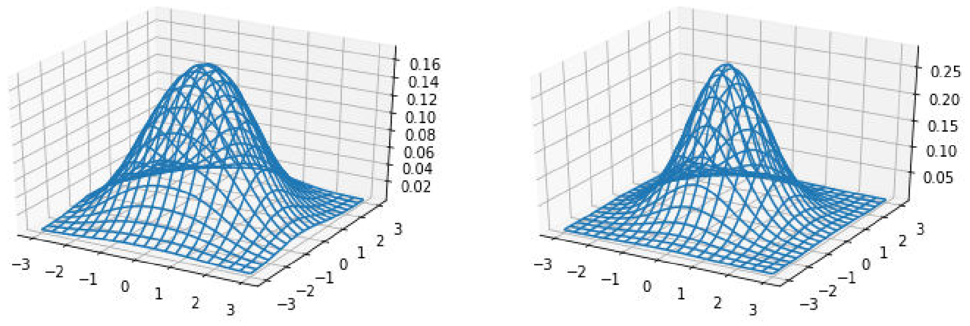

Figure 10: The left probability surface is the mean of three Gaussian distribution with same standard deviation $[1, 1]$, but different means $[1, 0]$, $[\frac{\sqrt{3}}{2}, -\frac{1}{2}]$, $[-\frac{\sqrt{3}}{2}, -\frac{1}{2}]$.
The right probability surface is standard Gaussian distribution with mean $[0, 0]$ and standard deviation $[1, 1]$.

Figure 11 lists overall ARs for different numbers of clusters, the best two ARs in each column are in bold. It is evident that the PNN with 200 clusters has the best result in almost every year. Therefore, 200 is the most suitable number for RBFN's clusters.

6.6 Results

As an old model, GPR requires matrix inverse during both of the training process and the prediction process. When applying GPR on more than 200 observations, the matrix inverse becomes computationally very expensive which is limited by the CPU performance and memory space. A way of avoiding the large matrix inverse in the training process is only to take part of observations (about 300) randomly in each training loop, which gives a locally trained σ. After repeating a certain amount of loops, the locally trained σ is getting closer to the global trained σ. According to the relation between RBFN and GPR, the weights optimisation can replace the matrix inverse in the prediction process. However, the AR of the adjusted GPR model is about 0%. It is hard to say whether the adjustment makes the regression failed, but it is clear that the GPR model is not suitable for our data.

Year		2008	2009	2010	2011	2012	2013	2014	2015	Overall AR
1	AR	21.9%	19.8%	14.5%	19.1%	18.8%	27.6%	22.2%	19.5%	20.4%
	SD	0.1290	0.0757	0.0808	0.0509	0.0595	0.0537	0.0512	0.0346	
2	AR	19.5%	21.9%	13.9%	16.6%	19.0%	24.5%	23.0%	21.0%	19.9%
	SD	0.1268	0.0928	0.0946	0.0942	0.0830	0.0545	0.0571	0.0448	
3	AR	24.9%	18.8%	14.4%	19.8%	20.7%	29.0%	24.4%	20.0%	21.5%
	SD	0.0965	0.1068	0.0656	0.0901	0.0671	0.0543	0.0447	0.0628	
4	AR	20.9%	25.0%	11.9%	20.1%	17.5%	25.7%	22.8%	20.6%	20.6%
	SD	0.1337	0.0904	0.0663	0.0754	0.0760	0.0427	0.0709	0.0679	
5	AR	22.0%	21.2%	14.4%	16.0%	18.0%	24.7%	23.7%	19.5%	19.9%
	SD	0.1334	0.1043	0.0542	0.0711	0.0752	0.0522	0.0545	0.0519	
Average AR		21.8%	21.3%	13.8%	18.3%	18.8%	26.3%	23.2%	20.1%	20.5%
Average SD		0.1239	0.0940	0.0723	0.0763	0.0732	0.0515	0.0557	0.0524	

Table 12: The result of 200 clusters RBFN.
SD is the standard deviation of ARs in 10-Fold Cross-Validation. 1-5 means 5 times repeating test. The Overall AR is the average annual ARs.

During the application of the RBFN on the real data, the main problem is the training speed and required memory space. When the number of centres is over $20,000$, the program will run over all the available memory space (10GB) and terminate the process. The only way to avoid this problem is by reducing the number of centres to $5,000$. The first way is the Under-Sampling which is shown in Section 2.5, but the final AR is about 0%. The second way is ROLS centre selector, however, both of the Backward centre selector and the Forward centre selector need to run QR decomposition on all centres individually. It's very difficult to apply ROLS centre selector to the problem due to the limitation of the machine. The third method is the K-Means Clustering centre selector (see Section 6.5) and the optimal cluster size is 200. Therefore, the final model is to use K-Means Clustering centre selector on non-default centres with 200 clusters.

Table 12 is the best results which are RBFN with 200 clusters. The best annual AR is 26.3% in 2013's data, and the worst annual AR is 13.8% in 2010's data. There is no clear relation between AR and data size. Comparing with LR, RBFN does not have any advantage on training speed, prediction accuracy, and memory space.

7 Restricted Boltzmann Machine

7.1 Description

Restricted Boltzmann Machine (RBM) is the component layer of Deep Belief Network which is the typical representative of Deep Learning. Different from previous models, RBM is built on the Gibbs sampling (see Appendix B) and the conditional probability. In 1986, Paul Smolensky developed RBM which is used to be called as Harmonium [37].

The original type of RBM has a binary-valued **Hidden layer**, a binary-valued **Visible layer**, and an **Energy function**. The Energy function, $E(H, V)$, maps values of the Hidden layer, H, and the values of the Visible layer, V, to a surface which is proportional to the probability surface of V and H, $P(H, V)$. Therefore, RBM can calculate a conditional probability of any fields for any other given fields. It is worth to mention that RBM treats observations and their labels together as the Visible layer. The prediction is the conditional probability of the label given the observation.

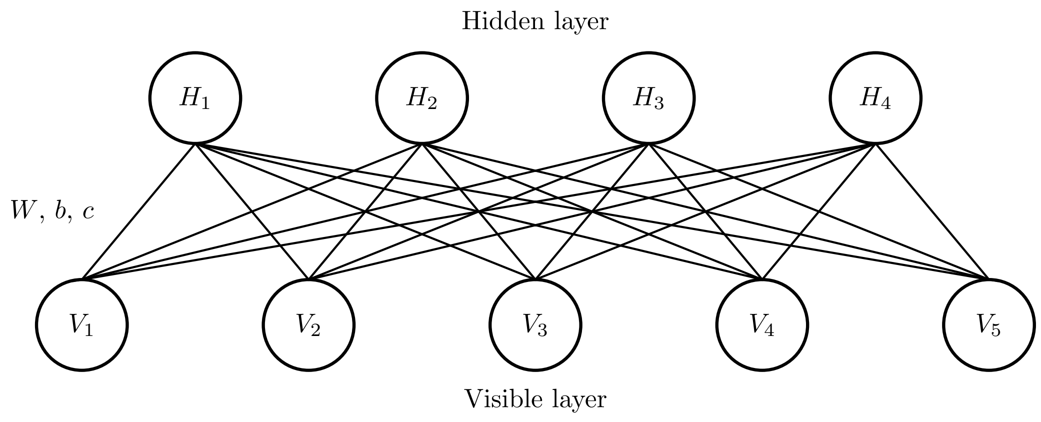

Figure 11: An example structure of RBM.

Figure 11 is an example structure of RBM which has four hidden neurons, five visible neurons and an Energy function,

$$E(V, H) = -H^T W V - b^T V - c^T H,$$

where the weights, W, and the biases b, c, are initialised randomly. The core of the RBM is assuming that it is able to find a probability function of the Visible layer, V, the Hidden layer, H, such that

$$P(V, H) = \frac{E(V, H)}{Z},$$

where Z is a normalisation constant which normally uses $\sum_{V,h} E(V, h)$. All observations, X, and their labels, Y, are input values of the Visible layer, V. For a observation, the values of the Hidden layer, H are sampled by the conditional probability, $P(H_j|V) = \phi(c_j + \sum_i W_{ji} V_i)$, where ϕ is the Sigmoid function. Then the positive phase is $E_0 = E(V, \phi(c + WV))$. Once the whole Hidden layer is generated following the conditional probabilities, the RBM model transforms information backward to reproduce the Visual layer $\hat{V}$, through the conditional probability function $P(\hat{V}_j|H) = \phi(b_j + W_j^T H_j)$. Similar as the previous, the new generated Hidden layer, $\hat{H}$, is following the conditional probability, $P(\hat{H}|\hat{V}) = \phi(c_j + \sum_i W_{ji} \hat{V}_i)$, and the negative phase is $E_1 = E(\hat{V}, \phi(c+$

$W\hat{V})$). Finally, the difference between the positive phase and the negative phase is the Loss function of the observation. This section only take the GD method as a example (see Section 2.4), all parameters $\Theta = (W, U, b, c, d)$ will be updated by $\Theta = \Theta - \lambda(\frac{\partial}{\partial\Theta}E_0 - \frac{\partial}{\partial\Theta}E_1)$. After the training process, the labels Y can be predicted by the conditional probability function [38]

$$\begin{aligned} P(Y|X) &= \frac{\sum_{H_1\in\{0,1\}}\sum_{H_2\in\{0,1\}}\cdots\sum_{H_n\in\{0,1\}}\exp(-E(X,Y,h))}{\sum_{Y^\star\in\{1,\dots\}}\sum_{H_1\in\{0,1\}}\cdots\sum_{H_n\in\{0,1\}}\exp(-E(X,Y^*,h))} \\ &= \frac{\exp(d_Y + \sum_j \mathrm{Softplus}(c_j + U_{jY} + \sum_i W_{ji}X_i))}{\sum_{Y^\star\in\{1,\dots\}}\exp(d_{Y^\star} + \sum_j \mathrm{Softplus}(c_j + U_{jY^\star} + \sum_i W_{ji}X_i))} \end{aligned} \tag{23}$$

where $\mathrm{Softplus}(x) = \log(1+\exp(x))$.

Similar to the Gibbs sampling (see Appendix B), the newly generated pair $(\hat{V},\hat{H})$ is equivalent to a random variable following the probability, $P(V,H)$. If the original pair, (V,H), has low probability, $P(V,H)$, then the new generate pair $(\hat{V},\hat{H})$ is likely to have higher value on the probability surface, i.e., $P(\hat{V},\hat{H}) > P(V,H)$, which is same as $E_1 > E_0$, hence reducing the difference between E_0 and E_1 is equivalent to maximising the likelihood of V.

7.2 Gaussian RBM and Further Improvements

The original RBM is used to analysis binary-valued data, not suitable for the database, but a combination with Gaussian probability allow RBM to have a non-binary Visible layer. The first Gaussian RBM (GRBM) is based on RBM and added the Gaussian noise into the model [39].

$$E(V,h) = -h^TWV - b^TV - c^Th + \frac{1}{2\sigma^2}\|V\|^2 \tag{24}$$

Honglak Lee, Chaitanya Ekanadham and Andrew Y.Ng did the further improvement [40]. They scaled the energy function by the variance, which reduced the mean square error between inputs and outputs. The Energy function and conditional probability functions are adapted to

$$\begin{aligned} E(V,h) &= -\frac{1}{\sigma^2}(h^TWV + b^TV + c^Th) + \frac{1}{2\sigma^2}\|V\|^2 \\ P(V_i|h) &= \mathcal{N}(b_i + h^TW_i, \sigma^2) \\ P(h_j|V) &= \phi((c_j + W_jV)/\sigma^2) \end{aligned} \tag{25}$$

where $\mathcal{N}(\mu,\nu)$ means a normal distribution with the mean μ and the standard deviation ν.

Gaussian Bernoulli RBM (GBRBM) is published by Krizhevsky, Alex, and Geoffrey Hinton [43]. It adapts the constant global σ to the individual local σ_i for each visible neurons.

$$\begin{aligned} E(V,h) &= \sum_i \frac{(V_i - a_i)^2}{2\sigma_i^2} - (\sum_j b_jh_j + \sum_{i,j}\frac{V_i}{\sigma_i}h_jW_{ij}) \\ P(V_i|h) &= \mathcal{N}(a_i + \sum_j \sigma_i h_j^T W_{ij}, \sigma_i^2) \\ P(h_j|V) &= \phi(\sum_i \frac{V_i}{\sigma_i}W_{ij} + b_j) \end{aligned} \tag{26}$$

σ_i can be adapted by Gibbs sampling, same as other parameters.

Based on GBRBM, Cho, KyungHyun, Alexander Ilin, and Tapani Raiko improved the Gaussian probability function, limited the affect of σ [44].

$$\begin{aligned} E(V,h) &= \sum_i \frac{(V_i - a_i)^2}{2\sigma_i^2} - (\sum_j b_j h_j + \sum_{i,j} \frac{V_i}{\sigma_i^2} h_j W_{ij}) \\ P(V_i|h) &= \mathcal{N}(a_i + \sum_j h_j^T W_{ij}, \sigma_i^2) \\ P(h_j|V) &= \phi(\sum_i \frac{V_i}{\sigma_i^2} W_{ij} + b_j) \end{aligned} \tag{27}$$

7.3 Hybrid Model

The data-set is a combination of non-binary observations and binary labels. It means the inputs of the Visible layer is mixed of binary and non-binary values. None of the previous RBM models fit this data type. According to Hinton, Geoffrey E.'s work [45], setting Visual layer and Hidden layer both to be Gaussian is not a good choice. Therefore this section introduces a Hybrid model from the original binary RBM and the improved GBRBM. The Hybrid RBM model contains a non-binary **Observations layer**, X, a binary **Label layer**, Y and a binary **Hidden layer**, H.

By adding their energy functions together, the new Energy function for the Hybrid RBM is

$$E(X,Y,h) = \sum_i \frac{(X_i - a_i)^2}{2\sigma_i^2} - (\sum_j b_j H_j + \sum_{i,j} \frac{X_i}{\sigma_i^2} h_j W_{ij} + \sum_k c_k Y_k + \sum_{k,j} Y_k U_{kj} H_j). \tag{28}$$

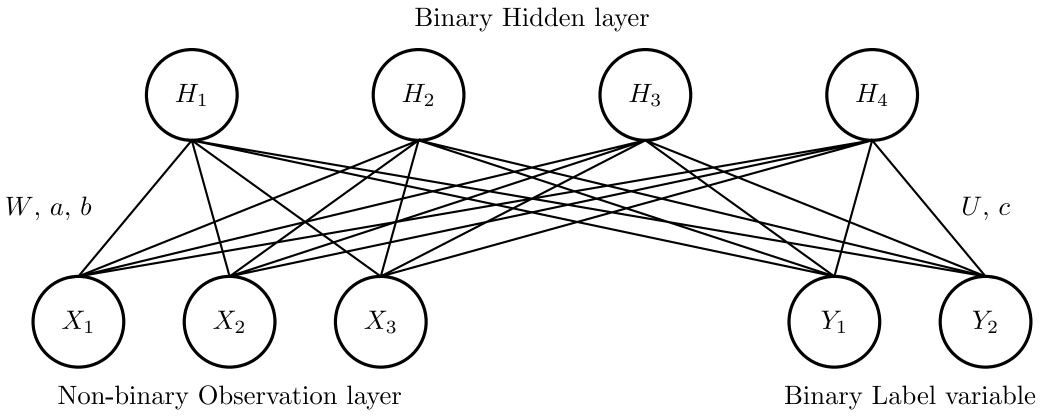

Figure 12: An example structure of Hybird RBM.

Once the Energy function is decided, the conditional probability functions are proved by Bayesian inference (Detail provement is in Appendix C). The conditional probability function for the Hidden layer is

$$\begin{aligned} P(\hat{H}_m|X,Y) &= \frac{\sum_{\hat{H}_{k\neq m}} p(X,Y,\hat{H}_m,\hat{H}_{k\neq m})}{\sum_H p(X,Y,H)} \\ &= \begin{cases} \phi\left(b_m + \sum_i \frac{X_i}{\sigma_i^2} W_{im} + \sum_k Y_k U_{km}\right) & \hat{H}_m = 1 \\ 1 - \phi\left(b_m + \sum_i \frac{X_i}{\sigma_i^2} W_{im} + \sum_k Y_k U_{km}\right) & \hat{H}_m = 0 \end{cases} \end{aligned} \tag{29}$$

where ϕ is the Sigmoid function, $\hat{H}_{k\neq m}$ means the Hidden layer without the mth neuron i.e. $\hat{H}_{k\neq m} = \left[\hat{H}_1, \ldots, \hat{H}_{m-1}, \hat{H}_{m+1}, \ldots, \hat{H}_n\right]$.

Similarly, the conditional probability of the Label layer is

$$\begin{aligned} p(\hat{Y}_m|X,H) &= \frac{\sum_{\hat{Y}_{k\neq m}} P(X,\hat{Y}_m,\hat{Y}_{k\neq m},H)}{\sum_Y P(X,Y,H)} \\ &= \begin{cases} \phi\left(c_m + \sum_j H_j U_{mj}\right) & \hat{Y}_m = 1 \\ 1-\phi\left(c_m + \sum_j H_j U_{mj}\right) & \hat{Y}_m = 0 \end{cases} \end{aligned} \tag{30}$$

and the conditional probability of the Observation layer is

$$\begin{aligned} P(\hat{X}_m|Y,H) &= \frac{\sum_{\hat{X}_{k\neq m}} P(\hat{X}_m,\hat{X}_{k\neq m},Y,H)}{\sum_X P(X,Y,H)} \\ &= \frac{1}{\sqrt{2\pi}\sigma_m} \exp\left(-\frac{(\hat{X}_m - a_m - \sum_j H_j W_{mj})^2}{\sigma_m^2}\right) \end{aligned} \tag{31}$$

The result of $P(\hat{X}_m|Y,h)$ is similar to the PDF of the normal distribution with the mean $a_m + \sum_j H_j W_{mj}$ and the standard deviation σ_m. Hence, for given Y and H, the new X_m can be generated by $\mathcal{N}(a_m + \sum_j H_j W_{mj}, \sigma_m^2)$, where $\mathcal{N}(\mu,\nu)$ means a normal distribution with the mean μ and the standard deviation ν.

As each Y_m represent a label class, each observation can only belong to one class. During the calculation of $P(\hat{Y}_m|X)$, it is able to assume that $\hat{Y}_k = 0$ for any $k \neq m$. Hence, the prediction function is

$$\begin{aligned} P(\hat{Y}_m|X) &= \frac{\sum_{\hat{Y}_{k\neq m},h} P(X,\hat{Y}_m,\hat{Y}_{k\neq m},H)}{\sum_{Y,H} P(X,Y,H)} \\ &= \frac{\exp(c_m\hat{Y}_m)\prod_j\left(\exp(\hat{Y}_m U_{mj} + \sum_i \frac{X_i}{\sigma_i^2} W_{ij} + b_j) + 1\right)}{\prod_j(\exp(\sum_i \frac{X_i}{\sigma_i^2} W_{ij} + b_j) + 1) + \exp(c_m)\prod_j\left(\exp(U_{mj} + \sum_i \frac{X_i}{\sigma_i^2} W_{ij} + b_j) + 1\right)} \end{aligned} \tag{32}$$

Take the example of Hybrid RBM in Figure 12, the training process is the same as the binary RBM. Suppose the training observations and labels are

$$X = \begin{bmatrix} x_1^1 & x_2^1 & x_3^1 \\ \vdots & \vdots & \vdots \\ x_1^n & x_2^n & x_3^n \end{bmatrix}, \quad Y = \begin{bmatrix} y^1 \\ \vdots \\ y^n \end{bmatrix}$$

respectively. As the labels are belong to two class: default (1) and non-default (0). Instead of 1-dimensional Y, RBM takes 2-dimensional

$$Y_0 = \begin{bmatrix} 1-y^1 & y^1 \\ \vdots & \vdots \\ 1-y^n & y^n \end{bmatrix}$$

as the Label layer. The first column of ith row is 1 if the ith observation is non-default, and it is 0 for a default observation. In other words, the first column is opposite of the second column, and there must be one "1" and one "0" in any row. After the initialisation the weights, the Hidden layer H_0 is calculated by the Observation layer $X_0 = X$, the Label layer Y_0 and the Equation (29), $P(H_0|X_0,Y_0)$. Then, the model backwards generate a new Observation layer X_1 and a new Label layer Y_1 by the Equations (30), $P(X_1|Y_0,H_0)$, and the Equation (31), $P(Y_1|X_1,H_0)$, respectively.

Similarly, their relative new Hidden layer H_1 is computed by the Equation (29), $P(H_1|X_1, Y_1)$. Finally, the gradient descent method (see Section 2.4) update all parameter $\theta = (W, U, a, b, c, \sigma)$ and the cost function

$$L(X_0, Y_0) = E(X_0, Y_0, H_0) - E(X_1, Y_1, H_1).$$

Repeat those processes until a converged cost on the training data-set or reaching a minimal cost on the Validation data-set.

After the training process, the default and non-default probability, Y_1, is calculated by the Equation (32), $P(Y_1|X_0)$, in which Y_1 is a $2 \times n$ matrix and n is the number of observations in the data-set. The first column of Y_1 is the probability of non-default and the second column is the probability of default. The label with higher probability will be the prediction of the Hybrid RBM.

7.4 Missing Data

Missing data is an inevitable problem when applying a statistic model to the real data. Normally, the observation with the missing field will be dropped during the data cleaning, or the missing field will be filled up by a guessing number. However, RBM can use known fields to find the highest possible of the missing field by $P(\hat{X}_m|X_{k \neq m})$, similar as predicting the class of labels by $P(\hat{Y}_m|X)$. Then the highest possible $\hat{X}_m$ is the value of missing data $X_{k \neq m}$, which chooses the mean of the normal distribution for non-binary X_m and the one from $\{0, 1\}$ with the higher probability for binary X_m.

The ability to generate missing data has more benefit in prediction. A training-completed RBM can predict a company without taking all fields. Although missing fields will reduce the accuracy of RBM, it is useful in practice.

7.5 Problems and Result

This thesis test RBM with 10, 100 and 200 hidden neurons. All of them produce about a 0% AR. The main reason for it is that RBM considers every dimension of observations and their labels equally. An extra large variable in observation will cause a larger effect than a wrong prediction. For example, some variable in observations is larger than $1,000$. The mean of each variable is under 10. So a_i in Energy function (see Equation (28)) would not exceed 20. The σ is less than 20 because 99% variables are under 10, also, the Hidden layer h is variable between 0 and 1. Hence the energy for observation containing extra large variable will be over $100,000$. The common observation (no extra large variable) has energy under 1000. The whole training process actually is trying to reduce the energy of those unusual observations.

One way of avoiding it is dropping all large observation values (over 10). It process increases the AR to 3% which is still a terrible result. Therefore, RBM is not a suitable classify model for our database.

8 Further Directions

8.1 RBM Dimensional Reduction

GRBM maps the Visible layer to the binary Hidden layer, and this process has a highly accurate inverse, hence, instead of 14 fields, the binary Hidden layer can be used for prediction. This GRBM transform can reduce the dimension of observations from 14 non-binary fields to finite binary values without losing most information from them. This process is called RBM dimensional reduction, [46].

For each training data, the GRBM dimensional reduction transforms the observation to binary values by the conditional probability, $P(H|V)$. Its Hidden layer becomes a new observation of other models. An example of the combination of RBM dimension reduction and Multi-Layer Perceptron is in Figure 13.

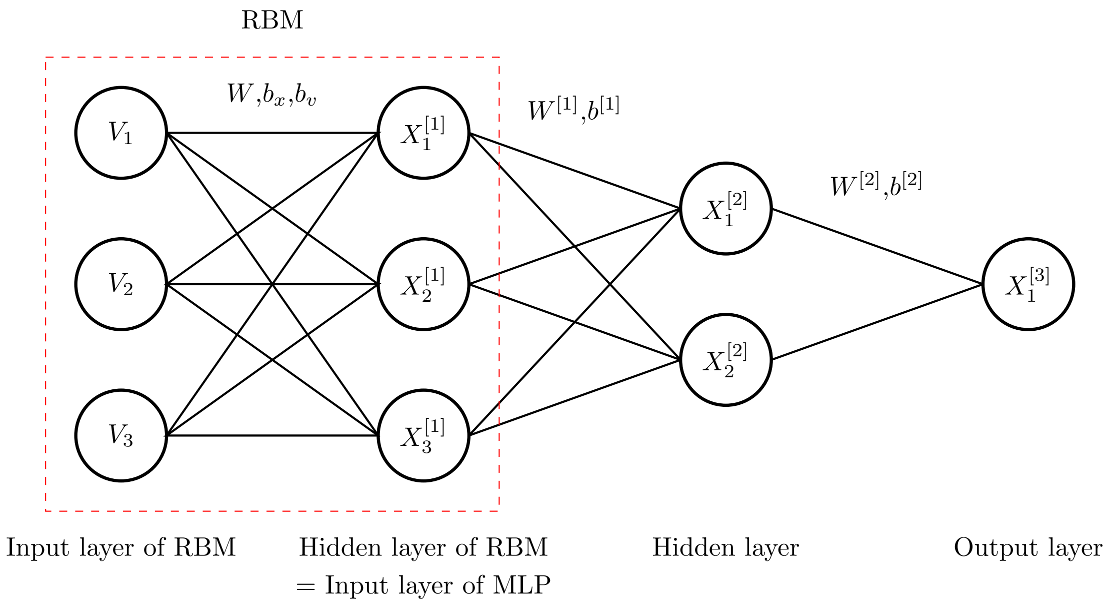

Figure 13: An example structure of RBM dimension reduction with MLP.

8.2 Voting Model

As this thesis has more than one usable model, the voting model can decide which one to use. The probabilistic voting model is based on the voting theory and Majority rule [47].

After applying all trained models for a observation and the majority label is the prediction of the Voting model.

9 Conclusion

This thesis applies five different models to the real companies data from 2008 to 2015. RBM has some issues during the processing (see Section 7.5). For the rest four models, their hyper-parameters are adapted in each Chapter. The comparison of the four best models is shown in Figure 14. The detail data are listed in Table 7, 9, 10 and 12

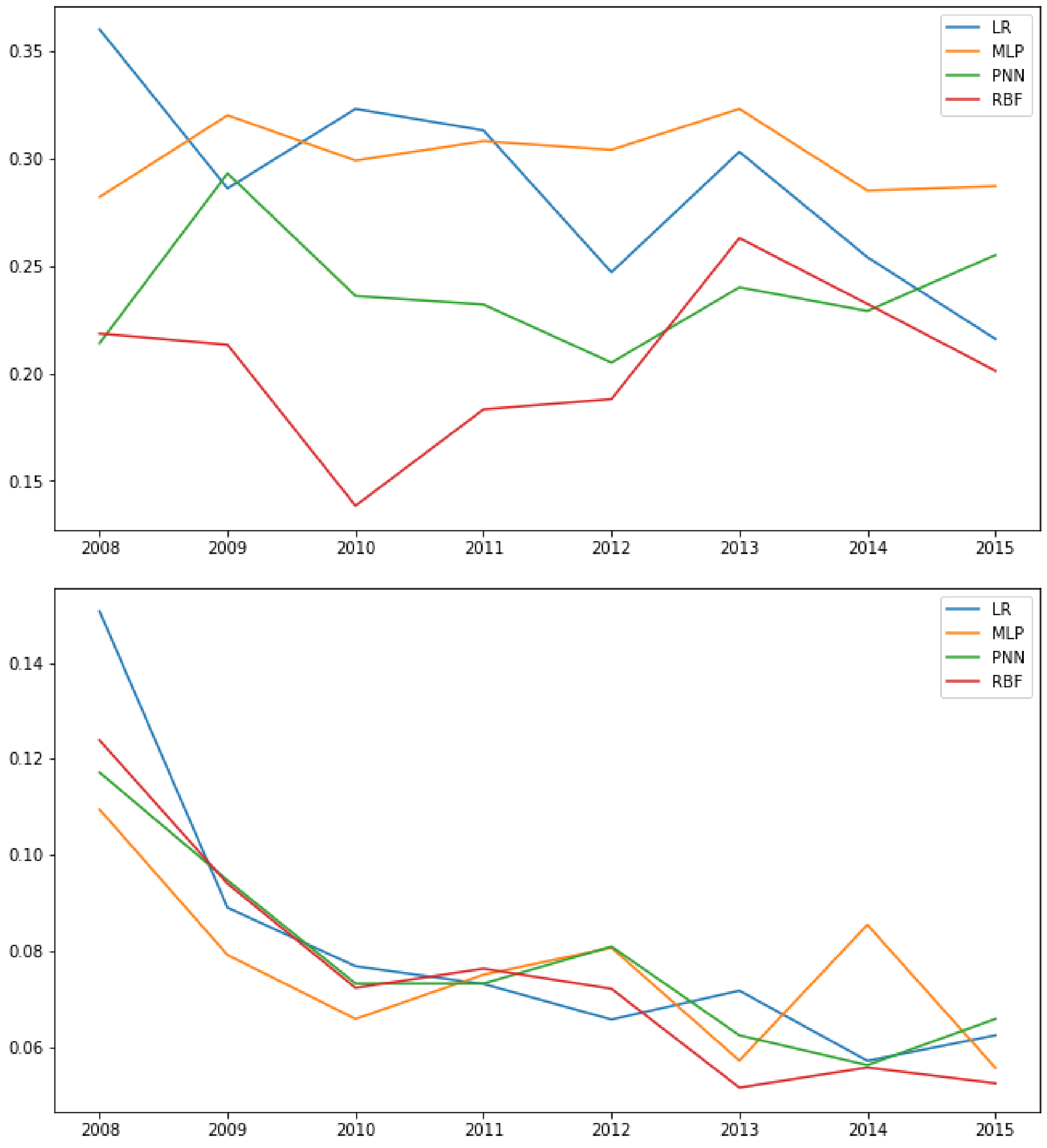

Figure 14: The above graph is the average annual AR of four best model. The below graph is the average standard deviation of four best model.

The SD in the second graph of Figure 14 represents the standard deviation of k-fold cross-validation results' AR. The SD tends to decrease while the increasing of data size. It means the increase of data size can improve the stability of four models' predictions.

The first graph of Figure 14 lists all AR for each year. It is obvious that a three-hidden-neurons MLP and LR have best overall AR. Although LR has better AR on 2008's, 2010's and

2011's observations, LR has a significant drop in the 2012's and the 2015's data. LR might have great overall AR, but it is too fluctuate. Therefore, MLP is the most suitable model to predict the bankrupt.

LR has the highest overall AR, 36%, on 2008's data, but decrease to minimise 21.6% on 2015's data. It shows a negative correlation between AR and data size. Also, Lr has a good AR before 2011. According to the Table 4, LR is more suitable for under $20,000$ data size.

A three-hidden-neurons MLP has higher overall AR between 2009 and 2013. According to the Table 4, the three-hidden-neurons MLP is more suitable for a $10,000$ to $40,000$ data size. However, a more complex database will be difficult to find a most suitable structure for MLP. Also, when the additional data is added to the database, the most suitable structure might changes.

PNN and RBFN do not have a regular pattern of AR in Figure 14. The reason for it is that the size of the training data will affect their structure, i.e., a more extensive training data size will lead to more hidden neurons. Their model complexity is self-adjust by data, which MLP and LR do not have. Therefore, for a much more complex data size, PNN and RBFN can be treated as a fast analysis.

LR has the best result for under $20,000$ data size. MLP is perfectly suitable for any size of data, but it requires more time to find the best structure. PNN's and RBFN's results are less affected by data size. They are more suitable for a growing database.

RBM is not suitable for prediction, but it can do a combination with other models.

A Appendix: Bisection method and Boundary Reduction method

Bisection method is a recursive method to find one solution or minimum of a single variable function. Different with gradient descent methods, it does not need to calculate the gradient of the function. For non-smooth continuous function, Bisection method is the only way to find the solution and minimal point.

The left graph in Figure A is a example of Bisection solution detector. Assume a solution of function, f, is in the period $[a, b]$ and $f(a) < 0 < f(b)$. The main process of Bisection method contains three steps:

1. Let $x_n = \frac{a+b}{2}$, and calculate $f(x_n)$.
2. If $f(x_n) > 0$ then $b \leftarrow x_n$. Otherwise, $a \leftarrow x_n$.
3. Repeat step 1 and 2 until get $f(x_n) = 0$, or the period $[a, b]$ is small enough.

($a \leftarrow x$ means giving the value of x to a.)

Following the steps of the Bisection method, the Bisection method finds x_1 in the first loop. Then the boundary reduces to the $[x_1, b]$. Repeatedly, it finds x_2, x_3, …, until the boundary is small enough, and the x_n is the root of function.

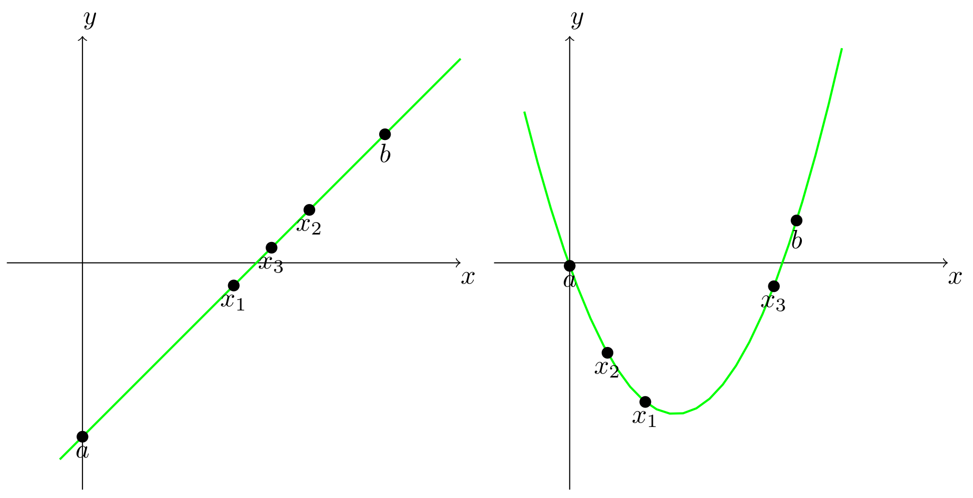

Figure 15: Bisection method finding solution (Left) and minimal point (Right).

Bisection minimal/maximal detector has an example in the right graph of the Figure 15. The main assumption of Bisection minimal detector is that the minimal or maximal point of the function f in the period $[a, b]$ is unique. Hence for any two points x, y in the period, if $f(x) > f(y)$ then the minimal point m is greater than x. Otherwise, m is less than y.

Suppose there is a minimal point in period $[a, b]$. Bisection minimal detector has more complex steps, as shown in Figure 16. Following those steps, it firstly finds x_1 and let x be the value of x_1. Then the detector finds x_2, and let a be the value of x_2. Repeat the process until it finds an accurate enough result or reach certain loops.

Based on the Bisection method, I develop the Boundary Reduction method. It contains similar

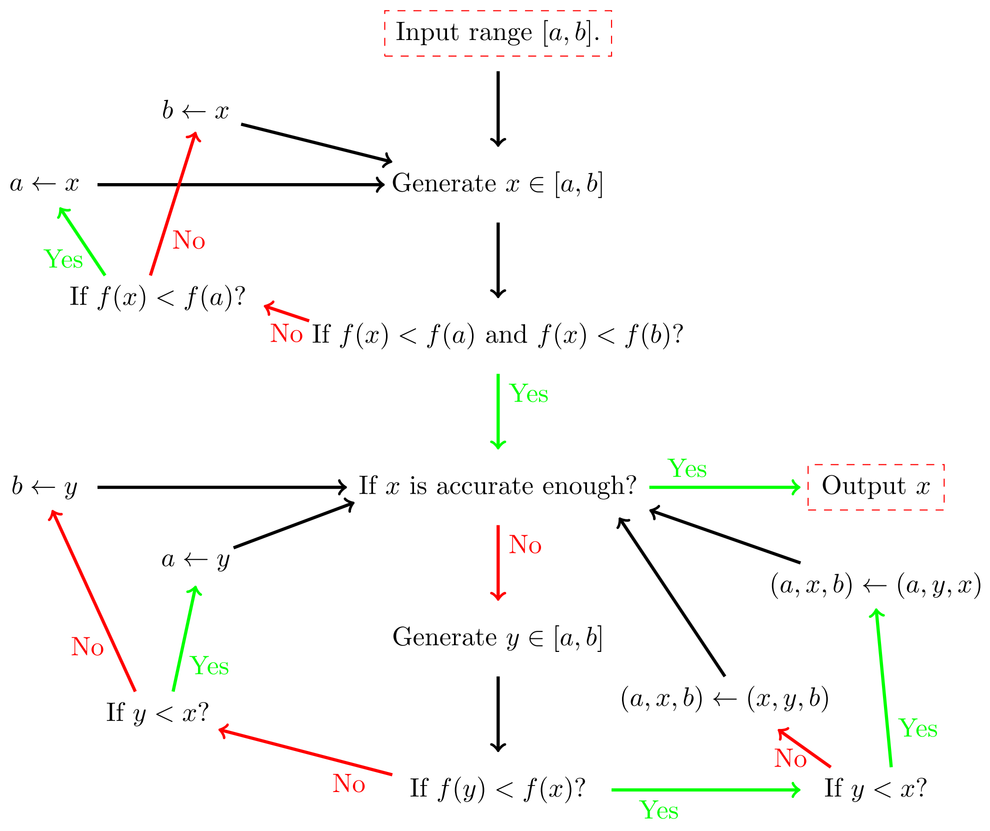

Figure 16: Bisection minimal finder process illustration.

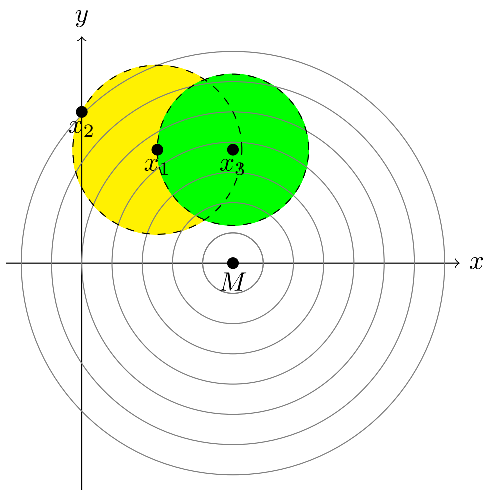

Figure 17: M is the minimal point of $f(x, y)$. Each concentric circle is a set $C_k = \{(x, y) \in \mathbb{R}^2 | f(x, y) = k\}$. Smaller concentric circle has smaller k.

four steps:

1. Choose two points in the space, a and b, such that $f(a) < f(b)$.
2. Randomly generate a point, x, in $\{x \in \mathbb{R}^2 | \, \|x - a\| < \|b - a\|\}$.
3. If $f(x) < f(a)$ then $b \leftarrow a$ and $a \leftarrow x$. If $f(a) < f(x) < f(b)$, then $b \leftarrow x$.
4. Repeat step 2,3, until get close enough a and b.

The Figure 17 shows an example of Boundary Reduction method, which starts with two points x_1 and x_2, and the next point, x_3, is generated from the yellow circle. The fourth point will be randomly picked from the green circle. Repeat the process until getting a small enough circle.

B Appendix: Gibbs sampling

Suppose ϕ_1, $\phi_2 \sim p(\phi_1, \phi_2)$ need to be sampled by the conditional probabilities $p(\phi_1|\phi_2)$ and $p(\phi_2|\phi_1)$. Then Gibbs sampling is a good method for this question.

Gibbs sampling has three steps:

1. Initial a starting value X_0, Y_0 in range of ϕ_1, ϕ_2.
2. Simulate X_i by $p(\phi_1|\phi_2 = Y_{i-1})$.
3. Simulate Y_i by $p(\phi_2|\phi_1 = X_i)$.
4. Repeat steps 2, 3 until get a converge enough result, reach certain amount of loop or achieve some conditions.

The final ϕ_1 and ϕ_2 can be treated as the random variables generated by $p(\phi_1|\phi_2)$ and $p(\phi_2|\phi_1)$.

C Appendix: Hybrid RBM Formula Provement

$$
\begin{aligned}
P(\hat{h}_m|X,Y) &= \frac{\sum_{\hat{h}_{k\neq m}} p(X,Y,\hat{h}_m,\hat{h}_{k\neq m})}{\sum_h p(X,Y,h)} \\
&= \frac{\sum_{\hat{h}_{k\neq m}} \exp\left(-E(X,Y,\hat{h}_m,\hat{h}_{k\neq m})\right)}{\sum_h \exp\left(-E(X,Y,h)\right)} \\
&= \frac{\sum_{\hat{h}_{k\neq m}} \exp\left(-\sum_i \frac{(X_i-a_i)^2}{2\sigma_i^2} + \sum_j b_j\hat{h}_j + \sum_{i,j}\frac{X_i}{\sigma_i^2}\hat{h}_j W_{ij} + \sum_k c_k Y_k + \sum_{k,j} Y_k U_{kj}\hat{h}_j\right)}{\sum_h \exp\left(-\sum_i \frac{(X_i-a_i)^2}{2\sigma_i^2} + \sum_j b_j h_j + \sum_{i,j}\frac{X_i}{\sigma_i^2} h_j W_{ij} + \sum_k c_k Y_k + \sum_{k,j} Y_k U_{kj} h_j\right)} \\
&= \frac{\exp\left(b_m\hat{h}_m + \sum_i \frac{X_i}{\sigma_i^2}\hat{h}_m W_{im} + \sum_k Y_k \hat{h}_m U_{km}\right)}{\sum_{h_m\in\{0,1\}} \exp\left(b_m h_m + \sum_i \frac{X_i}{\sigma_i^2} h_m W_{im} + \sum_k Y_k h_m U_{km}\right)} \\
&= \frac{\exp\left(b_m\hat{h}_m + \sum_i \frac{X_i}{\sigma_i^2}\hat{h}_m W_{im} + \sum_k Y_k \hat{h}_m U_{km}\right)}{1 + \exp\left(b_m + \sum_i \frac{X_i}{\sigma_i^2} W_{im} + \sum_k Y_k U_{km}\right)} \\
&= \begin{cases} \phi\left(b_m + \sum_i \frac{X_i}{\sigma_i^2} W_{im} + \sum_k Y_k U_{km}\right) & \hat{h}_m = 1 \\ 1 - \phi\left(b_m + \sum_i \frac{X_i}{\sigma_i^2} W_{im} + \sum_k Y_k U_{km}\right) & \hat{h}_m = 0 \end{cases}
\end{aligned}
\tag{33}
$$

$$
\begin{aligned}
p(\hat{Y}_m|X,h) &= \frac{\sum_{\hat{Y}_{k\neq m}} p(X,\hat{Y}_m,\hat{Y}_{k\neq m},h)}{\sum_Y p(X,Y,h)} \\
&= \frac{\exp\left(c_m\hat{Y}_m + \sum_j \hat{Y}_m h_j U_{mj}\right)}{\sum_{Y_m\in\{0,1\}} \exp\left(c_m Y_m + \sum_j Y_m h_j U_{mj}\right)} \\
&= \frac{\exp\left(c_m\hat{Y}_m + \sum_j \hat{Y}_m h_j U_{mj}\right)}{1 + \exp\left(c_m + \sum_j h_j U_{mj}\right)} \\
&= \begin{cases} \phi\left(c_m + \sum_j h_j U_{mj}\right) & \hat{Y}_m = 1 \\ 1 - \phi\left(c_m + \sum_j h_j U_{mj}\right) & \hat{Y}_m = 0 \end{cases}
\end{aligned}
\tag{34}
$$

$$
\begin{aligned}
P(\hat{X}_m|Y,h) &= \frac{\sum_{\hat{X}_{k\neq m}} p(\hat{X}_m, \hat{X}_{k\neq m}, Y, h)}{\sum_X p(X,Y,h)} \\
&= \frac{\sum_{\hat{X}_{k\neq m}} \exp\left(-E(\hat{X}_m, \hat{X}_{k\neq m}, Y, h)\right)}{\sum_h \exp\left(-E(X,Y,h)\right)} \\
&= \frac{\sum_{\hat{X}_{k\neq m}} \exp\left(-\sum_i \frac{(\hat{X}_i-a_i)^2}{2\sigma_i^2} + \sum_j b_j h_j + \sum_{i,j} \frac{\hat{X}_i}{\sigma_i^2} h_j W_{ij} + \sum_k c_k Y_k + \sum_{k,j} Y_k U_{kj} h_j\right)}{\sum_X \exp\left(-\sum_i \frac{(X_i-a_i)^2}{2\sigma_i^2} + \sum_j b_j h_j + \sum_{i,j} \frac{X_i}{\sigma_i^2} h_j W_{ij} + \sum_k c_k Y_k + \sum_{k,j} Y_k U_{kj} h_j\right)} \\
&= \frac{\exp\left(-\frac{(\hat{X}_m-a_m)^2}{2\sigma_m^2} + \sum_j \frac{\hat{X}_m}{\sigma_m^2} h_j W_{mj}\right)}{\int_{-\infty}^{\infty} \exp\left(-\frac{(X_m-a_m)^2}{2\sigma_m^2} + \sum_j \frac{X_m}{\sigma_m^2} h_j W_{mj}\right) dX_m} \\
&= \frac{\exp\left(-\frac{(\hat{X}_m-a_m)^2}{2\sigma_m^2} + \sum_j \frac{\hat{X}_m}{\sigma_m^2} h_j W_{mj}\right)}{\sqrt{2\pi}\sigma_m \exp(-\frac{a_m^2}{2\sigma_m^2} + \frac{1}{\sigma_m^2}(a_m + \sum_j h_j W_{mj})^2)} \\
&= \frac{1}{\sqrt{2\pi}\sigma_m} \exp\left(-\frac{(\hat{X}_m - a_m - \sum_j h_j W_{mj})^2}{\sigma_m^2}\right)
\end{aligned}
\tag{35}
$$

$$
\begin{aligned}
P(\hat{Y}_m|X) &= \frac{\sum_{\hat{Y}_{k\neq m},h} p(X, \hat{Y}_m, \hat{Y}_{k\neq m}, h)}{\sum_{Y,h} p(X,Y,h)} \\
&= \frac{\sum_h \exp\left(c_m \hat{Y}_m + \sum_j \hat{Y}_m U_{mj} h_j + \sum_{i,j} \frac{X_i}{\sigma_i^2} h_j W_{ij} + \sum_j b_j h_j\right)}{\sum_{Y_m,h} \exp\left(c_m Y_m + \sum_j Y_m U_{mj} h_j + \sum_{i,j} \frac{X_i}{\sigma_i^2} h_j W_{ij} + \sum_j b_j h_j\right)} \\
&= \frac{\exp(c_m \hat{Y}_m) \sum_h \exp\left(\sum_j \hat{Y}_m U_{mj} h_j + \sum_{i,j} \frac{X_i}{\sigma_i^2} h_j W_{ij} + \sum_j b_j h_j\right)}{\sum_{Y_m \in \{0,1\}} \exp\left(c_m Y_m\right) \sum_h \exp\left(\sum_j Y_m U_{mj} h_j + \sum_{i,j} \frac{X_i}{\sigma_i^2} h_j W_{ij} + \sum_j b_j h_j\right)} \\
&= \frac{\exp(c_m \hat{Y}_m) \prod_j \left(\exp(\hat{Y}_m U_{mj} + \sum_i \frac{X_i}{\sigma_i^2} W_{ij} + b_j) + 1\right)}{\sum_{Y_m \in \{0,1\}} \exp\left(c_m Y_m\right) \prod_j \left(\exp(Y_m U_{mj} + \sum_i \frac{X_i}{\sigma_i^2} W_{ij} + b_j) + 1\right)} \\
&= \frac{\exp(c_m \hat{Y}_m) \prod_j \left(\exp(\hat{Y}_m U_{mj} + \sum_i \frac{X_i}{\sigma_i^2} W_{ij} + b_j) + 1\right)}{\prod_j (\exp(\sum_i \frac{X_i}{\sigma_i^2} W_{ij} + b_j) + 1) + \exp\left(c_m\right) \prod_j \left(\exp(U_{mj} + \sum_i \frac{X_i}{\sigma_i^2} W_{ij} + b_j) + 1\right)}
\end{aligned}
\tag{36}
$$

D Appendix: Tables

Year		2008	2009	2010	2011	2012	2013	2014	2015	Overall AR
1	AR	-0.5%	-17.2%	-0.4%	1.6%	-5.3%	5.6%	18.5%	23.0%	3.2%
	SD	0.019	0.062	0.015	0.061	0.061	0.055	0.043	0.085	
2	AR	-0.5%	-16.9%	-0.4%	1.7%	-4.5%	5.7%	19.6%	22.9%	3.5%
	SD	0.018	0.057	0.017	0.038	0.083	0.062	0.072	0.049	
3	AR	-0.5%	-16.9%	-0.3%	1.2%	-3.7%	6.2%	17.5%	22.1%	3.2%
	SD	0.023	0.068	0.016	0.050	0.098	0.050	0.058	0.043	
4	AR	-0.5%	-19.1%	-0.4%	1.9%	-2.7%	5.6%	17.4%	22.2%	3.0%
	SD	0.045	0.063	0.019	0.036	0.090	0.059	0.057	0.066	
5	AR	-0.5%	-17.3%	-0.1%	1.2%	-3.5%	6.5%	17.5%	22.8%	3.3%
	SD	0.027	0.071	0.014	0.053	0.086	0.069	0.052	0.040	
Average AR		-0.5%	-17.5%	-0.3%	1.5%	-3.9%	5.9%	18.1%	22.6%	3.2%

Table 13: Logistic Regression results with default-balanced but non-winsorized data.

SD is the standard deviation of ARs in 10-Fold Cross-Validation. 1-5 means 5 times repeating test.

The Overall AR is the average annual ARs.

Year		2008	2009	2010	2011	2012	2013	2014	2015	Overall AR
1	AR	0.67%	0.27%	0.13%	-0.11%	0.80%	0.07%	-0.01%	-0.01%	0.23%
	SD	2.20E-02	1.00E-02	2.00E-02	4.20E-03	1.90E-02	1.40E-02	2.30E-04	1.20E-04	
2	AR	0.56%	-0.10%	0.35%	0.04%	-0.58%	-0.40%	0.06%	0.00%	-0.01%
	SD	1.60E-02	2.60E-03	1.50E-02	8.60E-03	1.90E-02	4.80E-03	4.90E-03	7.80E-05	
3	AR	0.59%	-0.14%	0.65%	0.20%	-0.56%	-0.39%	-0.01%	0.00%	0.04%
	SD	2.40E-02	2.70E-03	2.90E-02	6.70E-03	1.30E-02	9.60E-03	2.30E-04	7.80E-05	
4	AR	0.56%	-0.05%	0.25%	0.53%	0.09%	-0.10%	0.15%	-0.01%	0.18%
	SD	4.30E-02	8.20E-04	1.80E-02	1.20E-02	2.60E-02	8.00E-03	5.40E-03	9.00E-05	
5	AR	0.56%	0.16%	-0.23%	0.10%	1.35%	-0.12%	-0.01%	-0.01%	0.23%
	SD	2.40E-02	1.00E-02	8.60E-03	8.60E-03	3.20E-02	1.40E-02	2.30E-04	8.90E-05	
Average AR		0.59%	0.03%	0.23%	0.15%	0.22%	-0.19%	0.03%	-0.01%	0.13%

Table 14: Logistic Regression results of winsorized but non-default-balanced data.

SD is the standard deviation of ARs in 10-Fold Cross-Validation. 1-5 means 5 times repeating test.

The Overall AR is the average annual ARs.

Year		2008	2009	2010	2011	2012	2013	2014	2015	Overall AR
1	AR	23.3%	24.9%	19.1%	23.2%	14.3%	22.1%	19.0%	20.7%	20.8%
	SD	0.0978	0.0937	0.1060	0.0909	0.0826	0.0581	0.0504	0.0457	
2	AR	21.0%	25.8%	19.0%	19.4%	15.2%	22.0%	18.8%	21.3%	20.3%
	SD	0.1331	0.0929	0.0784	0.0930	0.0856	0.0397	0.0731	0.0555	
3	AR	14.2%	25.1%	18.5%	17.8%	16.4%	20.3%	19.4%	21.2%	19.1%
	SD	0.0667	0.1032	0.1076	0.0548	0.0691	0.0620	0.0752	0.0136	
4	AR	21.9%	20.2%	17.0%	18.9%	13.4%	19.1%	21.8%	23.0%	19.4%
	SD	0.1515	0.1037	0.0756	0.0973	0.0641	0.0766	0.0657	0.0619	
5	AR	18.4%	19.7%	19.8%	18.5%	16.0%	21.8%	16.8%	20.9%	19.0%
	SD	0.1727	0.0716	0.0718	0.0804	0.0699	0.0442	0.0379	0.0528	
Average AR		19.7%	23.1%	18.7%	19.6%	15.1%	21.1%	19.2%	21.4%	19.7%

Table 15: PNN results with default-balanced data and single global σ.

SD is the standard deviation of ARs in 10-Fold Cross-Validation. 1-5 means 5 times repeating test.

The Overall AR is the average annual ARs.

References

[1] Beaver, William H. *Financial ratios as predictors of failure.* Journal of accounting research (1966): 71-111.

[2] Aguilera, Ana M., Manuel Escabias, and Mariano J. Valderrama. *Using principal components for estimating logistic regression with high-dimensional multicollinear data.* Computational Statistics & Data Analysis 50.8 (2006): 1905-1924.

[3] Byford, Sam. *Googles AlphaGo AI beats Lee Se-dol again to win Go series 4-1.* (2017).

[4] Gibbs, Samuel. *Alphazero AI beats champion chess program after teaching itself in four hours.* Guardian, December 7 (2017).

[5] Desai, Vijay S., et al. *Credit-scoring models in the credit-union environment using neural networks and genetic algorithms.* IMA Journal of Management Mathematics 8.4 (1997): 323-346.

[6] Piramuthu, Selwyn. *Financial credit-risk evaluation with neural and neurofuzzy systems.* European Journal of Operational Research 112.2 (1999): 310-321.

[7] West, David. *Neural network credit scoring models* Computers and Operations Research 27 (2000) 1131-1152

[8] Lee, Tian-Shyug, et al. *Credit scoring using the hybrid neural discriminant technique.* Expert Systems with applications23.3 (2002): 245-254.

[9] Mirta Bensic, Natasa Sarlija and Marijana Zekic-Susac. *Modelling Small-Business Credit Scoring by using Logistic Regression, Neural Networks and Decision Tree* Intelligent systems in Accounting, Finance and Management, 13, 133-150 (2005)

[10] Ong, Chorng-Shyong, Jih-Jeng Huang, and Gwo-Hshiung Tzeng. *Building credit scoring models using genetic programming.* Expert Systems with Applications 29.1 (2005): 41-47.

[11] De Keyser, Robain. *Engineering Applications in Artificial Intelligence.* (2005).

[12] Beaver, William H. *Market prices, financial ratios, and the prediction of failure.* Journal of accounting research (1968): 179-192.

[13] Edmister, Robert O. *An empirical test of financial ratio analysis for small business failure prediction.* Journal of Financial and Quantitative analysis 7.2 (1972): 1477-1493.

[14] Altman, Edward I. *Financial ratios, discriminant analysis and the prediction of corporate bankruptcy.* The journal of finance 23.4 (1968): 589-609.

[15] Ohlson, James A. *Financial ratios and the probabilistic prediction of bankruptcy.* Journal of accounting research (1980): 109-131.

[16] Rakotomamonjy, Alain. *Optimizing area under ROC curves with SVMs.* (2004).

[17] Bengio, Yoshua, and Yves Grandvalet. *No unbiased estimator of the variance of k-fold cross-validation.* Journal of machine learning research 5.Sep (2004): 1089-1105.

[18] Chawla, Nitesh V., et al. *SMOTE: synthetic minority over-sampling technique.* Journal of artificial intelligence research 16 (2002): 321-357.

[19] Liu, Xu-Ying, Jianxin Wu, and Zhi-Hua Zhou. *Exploratory undersampling for class-imbalance learning.* IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 39.2 (2009): 539-550.

[20] Frank E. Harrell, Jr.: *Regression Modeling Strategies.* With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis

[21] Dreiseitl, Stephan, and Lucila Ohno-Machado. *Logistic regression and artificial neural network classification models: a methodology review.* Journal of biomedical informatics 35.5-6 (2002): 352-359.

[22] Montavon, Grgoire, Genevive B. Orr, and Klaus-Robert Müller. *Tricks of the Trade.* (1998).

[23] Leshno, Moshe, et al. *Multilayer feedforward networks with a non-polynomial activation function can approximate any function.* (1992).

[24] Ruder, Sebastian. *An overview of gradient descent optimization algorithms.* arXiv preprint arXiv:1609.04747 (2016).

[25] Kingma, Diederik P., and Jimmy Ba. *Adam: A method for stochastic optimization.* arXiv preprint arXiv:1412.6980 (2014).

[26] Karsoliya, Saurabh. *Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture.* International Journal of Engineering Trends and Technology 3.6 (2012): 714-717.

[27] Boger, Zvi, and Hugo Guterman. *Knowledge extraction from artificial neural networks models.* IEEE International Conference On Systems Man And Cybernetics. Vol. 4. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE), 1997.

[28] Berry, Michael J., and Gordon Linoff. *Data mining techniques: for marketing, sales, and customer support.* John Wiley & Sons, Inc., 1997.

[29] Blum, Adam. *Neural networks in C++: an object-oriented framework for building connectionist systems.* Vol. 1. New York: Wiley, 1992.

[30] Donald F. Specht:*Probabilistic Neural Networks*Neural Networks. Vol. 3. pp. 109-118. 1990

[31] D S Broomhead and D Lowe. *Radial Basis Functions, Multi-variable functional interpolation and adaptive networks* ROYAL SIGNALS AND RADAR ESTABLISHMENT MALVERN 1988

[32] Gomm, J. Barry, and Ding Li Yu. *Selecting radial basis function network centres with recursive orthogonal least squares training.* IEEE Transactions on Neural networks 11.2 (2000): 306-314.

[33] Akaike, Hirotugu. *Fitting autoregressive models for prediction.* Annals of the institute of Statistical Mathematics 21.1 (1969): 243-247.

[34] Yu, D. L., J. B. Gomm, and D. Williams. *A recursive orthogonal least squares algorithm for training RBF networks.* Neural Processing Letters 5.3 (1997): 167-176.

[35] Huang, De-Shuang, and Wen-Bo Zhao. *Determining the centres of radial basis probabilistic neural networks by recursive orthogonal least square algorithms.* Applied Mathematics and Computation 162.1 (2005): 461-473.

[36] David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams: *Learning representations by back-propagating errors* Nature VOL 323 9 OCTOBER 1986

[37] Smolensky, Paul (1986).*Chapter 6: Information Processing in Dynamical Systems: Foundations of Harmony Theory* In Rumelhart, David E.; McLelland, James L. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations. MIT Press. pp. 194281. ISBN 0-262-68053-X.

[38] Hugo Larochelle, Michael Mandel, Razvan Pascanu and Yoshua Bengio *Learning Algorithms for classification Restricted Boltzmann Machine* Journal of Machine Learning Research 13 (2012) 643-669

[39] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle *Greedy Layer-Wise Training of Deep Networks* In Proc.Annu.NIPS.conf. vol. 19 Advances in Neural Information Processing Systems, Vancouver, Dec.2006, pp. 163-160.

[40] Honglak Lee, chaitanya Ekanadham and Andrew Y.Ng*Sparse deep belief net model for visual area V2* U.G.thesis, Symbolic systems program, Stanford University, Jun.2007.

[41] Noel Cressie:*The Origins of Kriging* Mathematical Geology, Vol. 22. No. 3, 1990

[42] Christopher K. I. Williams and Carl Edward Rasmussen *Gaussian Processes for Regression* Advances in neural information processing systems. P 514–520. 1996.

[43] Krizhevsky, Alex, and Geoffrey Hinton. *Learning multiple layers of features from tiny images.* Vol. 1. No. 4. Technical report, University of Toronto, 2009.

[44] Cho, KyungHyun, Alexander Ilin, and Tapani Raiko. *Improved learning of Gaussian-Bernoulli restricted Boltzmann machines.* International conference on artificial neural networks. Springer, Berlin, Heidelberg, 2011.

[45] Hinton, Geoffrey E. *A practical guide to training restricted Boltzmann machines.* Neural networks: Tricks of the trade. Springer, Berlin, Heidelberg, 2012. 599-619.

[46] Hinton G E, Salakhutdinov R R. *Reducing the dimensionality of data with neural networks[J].* science, 2006, 313(5786): 504-507.

[47] Coughlin, Peter J. *Probabilistic voting theory.* Cambridge University Press, 1992.