

Lappeenranta University of Technology
School of Business and Management
Degree Program in Computer Science

Shaghayegh Royae

**An Evaluation of Statistical Models for Programmatic TV Bid Clearance
Predictions**

Supervisor: Professor Ahmed Seffah

Co-supervisors: David Andersson

Adam Wojciechowski

ABSTRACT

Lappeenranta University of Technology
School of Business and Management
Degree Program in Computer Science

Shaghayegh Royae

An Evaluation of Statistical Models for Programmatic TV Bid Clearance Predictions

Master's Thesis

96 pages, 45 figures, 4 tables, 5 appendixes

Supervisor: Professor Ahmed Seffah

Keywords: Machine Learning, Neural Networks, Generalized Linear Models, Optimization, Support Vector Machines.

This thesis presents three different classes of statistical models which are applied to predict bid clearance probabilities in first price auctions. The data set used comprise in two years' worth of data from programmatic TV auctions where each bid has assigned class labels. Generalized Linear Models, Neural Networks and Support Vector Machines are described in detail together with methods for data pre-processing. Special focus is on Neural Networks and in particular on the comparison of the performance of different optimization methods which are evaluated in order to find the method most suitable. Findings indicate that Neural Networks perform on par, or better than the other methods mainly due to their high accuracy when predicting losing bids.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my mentor, David Andersson for his excellent supervision, suggestions and guidance during this work.

Many thanks to my supervisor, Professor Ahmed Seffah at Lappeenranta University of Technology for his great support and guidance during this master thesis and throughout my educational process.

TABLE OF CONTENTS

1	INTRODUCTION	7
1.1	BACKGROUND.....	7
1.2	ADVERTISING IN TV INDUSTRY.....	7
1.3	RESEARCH OBJECTIVES AND QUESTIONS AND METHODOLOGY.....	8
1.4	STRUCTURE OF THE THESIS	9
1.5	DATA COLLECTION AND ANALYSIS PROTOCOL.....	10
1.5.1	<i>Data preprocessing</i>	10
1.5.2	<i>One hot encoding</i>	10
1.5.3	<i>Data Analysis</i>	12
1.5.4	<i>Covariance Matrix</i>	12
2	STATISTICAL MODELS APPLIED TO THE PROBLEM AND METHOD OF WORK	16
2.1	DESCRIPTION OF PROBLEM.....	16
2.2	STATISTICAL MODELS	16
2.2.1	<i>Generalized Linear Models</i>	16
2.2.2	<i>Neural Networks</i>	20
2.3	SUPPORT VECTOR MACHINES	31
2.4	METHOD OF WORK.....	33
3	DATA ANALYSIS AND RECOMMENDATION.....	37
3.1	INITIAL ASSESSMENT.....	37
3.2	DETAILED DATA ANALYSIS	37
3.2.1	<i>Neural Networks</i>	37
3.2.2	<i>Generalized Linear Models</i>	56
3.2.3	<i>Support Vector Machines</i>	56
3.3	MAJOR FINDINGS AND RECOMMENDATIONS.....	56
4	DISCUSSION AND CONCLUSIONS	57
4.1	SUMMARY.....	57
4.2	MAJOR RESULTS	57

4.3	FUTURE WORK	58
5	REFERENCES	59
	APPENDIX 1. MATLAB CODE FOR NEURAL NETWORKS	63
	APPENDIX 2. MATLAB CODE FOR GENERALIZED LINEAR MODELS.....	76
	APPENDIX 3. MATLAB CODE FOR SUPPORT VECTOR MACHINES.....	82
	APPENDIX 4. MATLAB CODE FOR PREPROCESSING.....	89
	APPENDIX 5. MATLAB CODE FOR PCA.....	92

LIST OF TABLES

Table 1. Sample inputs of categorical data	11
Table 2. One hot encoded samples	11
Table 3. Data Preprocessing	34
Table 4. Evaluation process	35

LIST OF FIGURES

Figure 1. Principal component analysis	13
Figure 2. Rotation of the data	14
Figure 3. A zoom in of a region	14
Figure 4. Logistic Regression	19
Figure 5. Architecture of Neural Networks	21
Figure 6. Basic Structure of Backpropagation	23
Figure 7. linear binary support vector machine classifiers	31
Figure 8. Illustration of method	34
Figure 9. Illustration of validation	36
Figure 10. Mean Positive and Negative for conjugate gradient backpropagation with Polak-Ribière updates	38
Figure 11. Mean Positive for conjugate gradient backpropagation with Polak-Ribière updates	39
Figure 12. Mean Negative for conjugate gradient backpropagation with Polak-Ribière updates	39
Figure 13. Mean Positive and Negative for Fletcher-Reeves	40
Figure 14. Mean Positive for Fletcher-Reeves	40
Figure 15. Mean Negative for Fletcher-Reeves	40
Figure 16. Mean Positive and Negative for Levenberg-Marquardt backpropagation	41
Figure 17. Mean Positive for Levenberg-Marquardt backpropagation	42
Figure 18. Mean Negative for Levenberg-Marquardt backpropagation	42
Figure 19. Mean Positive and Negative for conjugate gradient backpropagation with Powell-Beale restarts	43
Figure 20. Mean Positive for conjugate gradient backpropagation with Powell-Beale restarts	43
Figure 21. Mean Negative for conjugate gradient backpropagation with Powell-Beale restarts	44
Figure 22. Mean Positive and Negative for one-step secant backpropagation	44
Figure 23. Mean Positive for one-step secant backpropagation	45
Figure 24. Mean Negative for one-step secant backpropagation	45
Figure 25. Mean Positive and Negative for Broyden Fletcher Goldfarb Shanno Quasi-Newton backpropagation	46

Figure 26. Mean Positive for Broyden Fletcher Goldfarb Shanno Quasi-Newton backpropagation.....	46
Figure 27. Mean Negative for Broyden Fletcher Goldfarb Shanno Quasi-Newton backpropagation.....	47
Figure 28. Mean Positive and Negative for bayesian regularization backpropagation	47
Figure 29. Mean Positive for bayesian regularization backpropagation	48
Figure 30. Mean Negative for bayesian regularization backpropagation.....	48
Figure 31. Mean Negative and Positive for scaled conjugate gradient	49
Figure 32. Mean Negative and Positive for scaled conjugate gradient	49
Figure 33. Mean Negative for scaled conjugate gradient	49
Figure 34. Mean Negative and Positive for gradient descent with momentum and adaptive learning rate backpropagation.....	50
Figure 35. Mean Positive for gradient descent with momentum and adaptive learning rate backpropagation.....	51
Figure 36. Mean Negative for gradient descent with momentum and adaptive learning rate backpropagation.....	51
Figure 37. Mean Negative and Positive for gradient descent with momentum backpropagation.....	52
Figure 38. Mean Negative and Positive for gradient descent with momentum backpropagation.....	52
Figure 39. Mean Negative and Positive for gradient descent with momentum backpropagation.....	53
Figure 40. Mean Negative and Positive for resilient backpropagation.....	53
Figure 41. Mean Positive for resilient backpropagation.....	54
Figure 42. Mean Negative for resilient backpropagation	54
Figure 43. Mean Negative and Positive for gradient descent	55
Figure 44. Mean Positive for gradient descent	55
Figure 45. Mean Negative for gradient descent.....	55

LIST OF SYMBOLS AND ABBREVIATIONS

ANN	Artificial Neural Network
BPA	Back Propagation Algorithm
BR	Bayesian Regulation
BiCePS	Bid Clearance Prediction Service
BFGS	Broyden Fletcher Goldfarb Shanno
CG	Conjugate Gradient
CGB	Conjugate Gradient with Powell/Beale Restarts
CGF	Fletcher-Powell Conjugate Gradient
CGP	Polak-Ribière Conjugate Gradient
CE	Cross Entropy
FFNN	Feedforward Neural Networks
GLM	Generalized Linear Models
GD	Gradient Descent
GDM	Gradient Descent with Momentum
GDX	Variable Learning Rate Gradient Descent
GDM	Gradient Descent with Momentum
LM	Levenberg-Marquardt
MSE	Mean Square Error
RP	Resilient Backpropagation
OSS	One Step Secant
QN	Quasi Newton
SVM	Support Vector Machine
SCG	Scaled Conjugate Gradient
Stdev	Standard Deviation
SVD	Singular Vector Decomposition
WO	WideOrbit
PTV	programmatic TV

1 INTRODUCTION

1.1 Background

WideOrbit (WO) company provides an advertising management software for media corporations like radio stations, TV stations and cable networks which are around 6,000 stations overall. It was launched in 1999 and the business is growing since then. The corporation offers a variety of services such as; the WO media sales, a sale management solution which aimed to reduce the time spent on making proposals and focusing on marketing which will lead to further sales opportunities and great enhance in customer satisfaction. It offers WO Traffic, an advertising sales and operations software solution, which aims to simplify management and operation of ad sales over different media platforms. WO Programmatic was introduced to highlight its offer for both linear and digital advertising, which is estimated to be the answer for growing business of broadcasting and networks. WideOrbit has focused on both linear and digital advertising, linear advertising refers to advertising that is presented in straightforward and fixed linear way on traditional media such as TV or radio, while digital advertising is when Internet technologies are used for delivering the ads instead of traditional media [1]. WO Programmatic TV (PTV) is an algorithmic method that will speed up the process in comparison to traditional advertising, which will potentially lead to more profit for a TV station. The focus of this thesis is on improving the available bid clearance predictor for PTV so that it can produce better results based on which advertisers manage their schedule and budget. At the moment, WideOrbit is headquartered in San Francisco and has offices in United States, and Europe [2], [3].

1.2 Advertising in TV industry

Advertising in traditional media has changed over the years. In the past, in order to place a commercial on TV, the advertiser had to go through a prolonged process, which could have been not even effective for their business. Programmatic buying categorizes and alleviates the burden of the entire process of advertising including placing orders, payments and optimizing available inventories [4].

In this method, spots will be sold in an auction-based, where the advertiser will place a bid for a specific spot, which has been chosen based on lots of features to address a particular group of audiences [5]. The auction is based on a first price auction scheme which means that all buyers can place their sealed bids and at the end of the auction, the highest presented price will win the auction [6]. Although the results are not published until the end of the auction, it is very important for the advertisers to know how probable it is for them to win the spot. This thesis aims to improve the available online TV auction's outcome predictor, in order to provide better feedbacks for the buyers.

1.3 Research objectives and questions and methodology

The key objective of this thesis is to analyze the performance of three different statistical models in conjunction with various optimization techniques in order to assess the suitability of either one as a prediction engine for online TV auctions. Hence, the key research question can be specified as: "which statistical model has the best performance for programmatic TV bid clearance predictions?"

Which could be decomposed into the following sub-questions:

1. How each individual model would perform based on same training dataset?
2. What approaches each model use in order to calculate the result?
3. What preprocessing algorithm can be used to normalize the data?

Computer simulation and literature review could be named as two primary research methods in this thesis. In order to answer the research questions, several simulations have been performed and different accredited pieces of literature have been reviewed.

The Finna Search Services of Lappeenranta University of Technology and Google search engine was used in order to access various databases and sources such as IEEE Xplore, ScienceDirect and SpringerLink. In this thesis, a wide investigation on other researchers' work has been done through literature review and the list of references is provided.

Computer simulations are growing to be an important method in scientific research which is widely used in different areas. Theories in regard to systems' performance can be assessed by conducting and modifying simulations [33].

In this study computer simulations have been used to explore the performance of different statistical models in order to find the most accurate model for predicting bid clearance probabilities in first price auctions. The simulation was directed in three phases. In phase one, the preprocessing was conducted based on patterns in the dataset. In phase two, the process of evaluation was elected using different theories. In Phase three, the simulations were performed and their generated results were compared in order to find the best performance [34]. A detailed description of the method of work is presented in section two.

1.4 Structure of the thesis

This thesis is consist of five main chapters. The first chapter presents an introduction about WideOrbit company, their business, background on advertising TV industry, as well as a description on data acquisition and data preprocessing algorithm applied on the dataset. In addition, research objectives, questions and methodologies are presented in this chapter.

The second chapter includes a description of the problem, and detail explanation of three different statistical models used to predict the outcomes for online TV auctions so that in next chapters these results can be compared and most accurate model will be selected. The models are classified as 1) Generalized Linear Models 2) Neural Networks 3) Support Vector Machines. In addition, it explains in detail, the several steps that are taken in this thesis in order to access the three statistical models and process the data in order to find the outcome of each prediction models.

The third chapter illustrates the outcome of each statistical model separately so that the results can be compared and the best-performed method which produces the most accurate prediction of the probability of the bid for winning the spot can be chosen.

In chapter four, a summary of the work and the conclusion is presented.

1.5 Data collection and analysis protocol

The data used in this study has been provided by the WideOrbit to be used for the master thesis. The provided data is for a period of approximately two years from 2015-11-01 up until 2017-01-30.

1.5.1 Data preprocessing

The data used in this thesis is consists of two parts, the first part is the categorical data on which one hot encoding has been performed and on the second part which is the continuous data; the following preprocessing formula has been used. The following formula will be applied to the values in each column with continues values:

$$\hat{x} = \frac{x - \text{mean}(x)}{\text{stdev}(x)} \quad (1)$$

For each value x in the column, the value will be subtracted by mean of the column and divided by its standard deviation. Standard deviation and mean will be calculated for each column separately [7].

1.5.2 One hot encoding

Machine learning algorithms used in classification problems performs much better when the categorical variables are nominal, which means when there is no order in the data [8]. There are 15 columns with categorical data in the database which all have been encoded to nominal values as the first step. Here week days have been chosen as an example:

Table 1. Sample inputs of categorical data

Sample	Weekday	Numerical
1	Thursday	1
2	Thursday	1
3	Friday	2
4	Monday	3
5	Saturday	4
6	Wednesday	5
7	Friday	2
8	Thursday	1

Eight input samples of categorical data have been shown here which are transformed into five distinct categories. First, the categorical values are encoded to these nominal values but this is not clear for machine learning algorithms since it gives the impression that category of Friday is greater than the category of Thursday. In order to avoid this confusion, one column for every unique category is created. All cells in the column are supposed to get zero values except one cell that contains the sample name on top of it [8].

Table 2. One hot encoded samples

Sample	Thursday	Friday	Monday	Saturday	Wednesday
1	1	0	0	0	0
2	1	0	0	0	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	0	1	0
6	0	0	0	0	1
7	0	1	0	0	0
8	1	0	0	0	0

One hot encoding works very well in this dataset and despite the massive increase in dimensionality, it generates better results in compare to not applying the one hot encoding on the categorical part of data.

1.5.3 Data Analysis

Data analysis can turn into a very challenging task when it comes to higher dimension of data, since further variables make it more difficult to process the data. It is quite common that in datasets with several variables, there are redundancies from an information theoretic point of view. Another common finding is that all variables do not affect the outcome greatly, thus some groups of variables can be replaced by an individual new variable. The Principal component analysis (PCA) is one of the methods that is used for dimension reduction which achieves the above mentioned, wanted result. The process projects data into eigenspace which makes the data linearly independent, capturing salient feature and information, in fewer variables. Every single one of them is a linear mixture of the primary variables and there is no redundancy between them [9].

1.5.4 Covariance Matrix

Performing the Eigen decomposition on covariance matrix is the basic step in PCA, the computation of the covariance matrix is done via the subsequent equation:

$$\Sigma = \frac{1}{n-1} ((X - \bar{x})^T (X - \bar{x})) \quad (2)$$

In this equation, \bar{x} is the mean vector and can be shown by the following formula, in which every value illustrates the mean of a collection of data in the database [10]:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (3)$$

Although Eigen decomposition of the covariance matrix is quite popular, performing a Singular Vector Decomposition (SVD) has shown better results in terms of advancing the

computational effectiveness. As it has been said before PCA aims to lower dimensionality of variables by introducing new variables, eigenvectors specify the direction of the line, and eigenvalue is a value, indicating the variance in data in that direction [11]. Eigenvalue will be used to select eigenvectors containing least data which can be omitted since they do not lead to losing valuable information. The general way for this technique is to rate eigenvalues in order to choose the most important ones [10]. In this study two eigenvectors were used when forming the basis for transformation and decorrelation, thus dimensionally reducing the 20 continuous columns to a 3d dataset suitable for plotting.

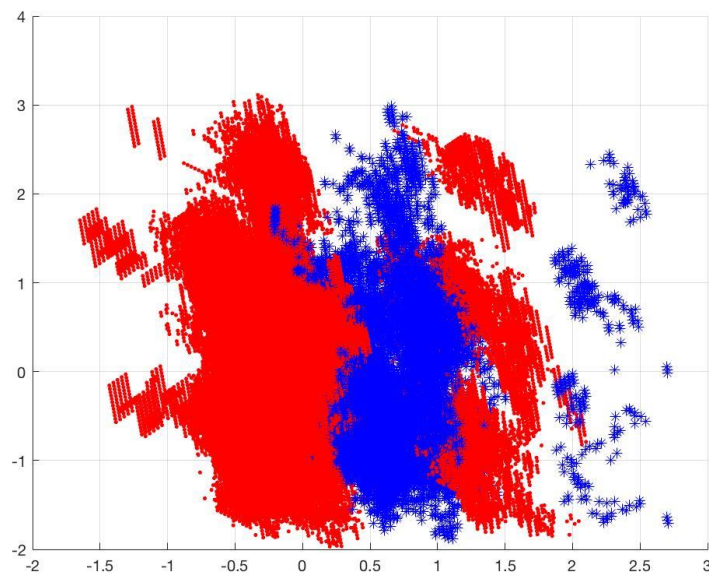


Figure 1. Principal component analysis

An initial view of the dimensionally reduced data indicates that the classification problem may be solved using a linear technique.



Figure 2. Rotation of the data

Rotation of the data indicates overlap between auction winners and loser when projected down to three dimensions thus rendering evaluations of non-linear techniques interesting.

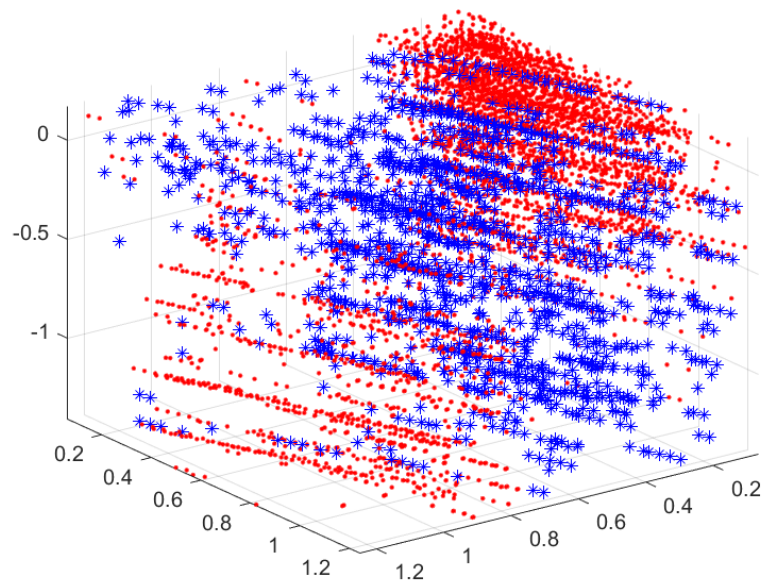


Figure 3. A zoom in of a region

A zoom in of a region where the density of winners seems high still shows overlap between the two classes of data.

As can be seen in the figure, structural information containing class conditional coordinates have been retained in the process, indicating valuable information in the continuous part of the dataset. In this study, MATLAB built-in function called PCA has been used to discover the principal components [10].

2 STATISTICAL MODELS APPLIED TO THE PROBLEM AND METHOD OF WORK

2.1 Description of problem

In WideOrbit Programmatic TV, spots for advertising on TV are sold in an auction based marketplace. All clients should submit their bids for specific spots that they want, but the result of winning or losing the auction is not available until the end of the day when the marketplace is closed.

However, it is very important for the advertisers to know how probable it is for them to win that specific spot they have bid on, so they can manage their budget in advance. At WideOrbit this valuable feedback (Probability of winning or the bid) is produced through a Bid Clearance Prediction Service (BiCePS). BiCePS is currently based on a Generalized Linear Models, but there are other statistical models that can be used in order to predict the probability of winning or losing a submitted bid.

The research problem is generated from the need of WideOrbit company to find the model that produces the most accurate prediction of winning or losing a bid in the Programmatic TV marketplace. Therefore, different commonly used statistical models have been evaluated and their generated results are compared in order to find the best method. Each statistical model has a unique approach for outcoming the probability of losing or winning, a detail description of three different statistical models assessed in this thesis is represented below:

2.2 Statistical models

2.2.1 Generalized Linear Models

Linear models can be used to forecast or examine experimental data. In classical linear models, it is assumed that elements of Y are normally distributed. The following form is a representation of conventional linear regression models [12].

$$E(Y) = X\theta \quad (4)$$

Where $E(Y)$ is the expected value of dependent variable, X is explanatory variable and θ is the parameter that is wished to be estimated.

Generalized Linear Models (GLM), are developed based on conventional linear models which allows for output distributions that are not Normal [13]. In generalized linear models instead of explanatory variable, there is a whole variety of them which vary together, hence they are called covariates. The GLM are defined by the following three components:

1. A conditional distribution which is a member of the exponential family.
2. A linear predictor, $\eta = \theta_0 + \theta_1x_1 + \theta_2x_2 + \dots + \theta_nx_n$
3. A link function g that convert expectation of response variable $E(Y) = \mu$ to the linear predictor: $g(\mu) = \eta$ [14].

2.2.1.1 Likelihood function for generalized linear models

In GLM it has been supposed that elements of Y come from the exponential family of distribution, which can be written as follows:

$$f_y(y; \theta, \phi) = \exp\{(y\theta - b(\theta))/a(\phi) + c(y, \phi)\} \quad (5)$$

In this formula $a(\phi)$, $b(\theta)$ and $c(y, \phi)$ are known functions and the canonical parameter is represented by θ . Overall the following form can be presented:

$$E(Y) = b'(\theta) \text{ and } \text{var}(Y) = b''(\theta)a(\phi) \quad (6)$$

Hence, the variance is dependent on two features, the first one is $b''(\theta)$ which is specified by mean and the second one is the variance function $V(\mu)$ that can be represented by the following form:

$$a(\phi) = \phi / \omega \sigma^2 = / \omega \quad (7)$$

Where φ is the dispersion parameter and is constant in the process, but ω is representation of the weight which changes in each iteration [15], [12].

2.2.1.2 Link functions

The purpose of link function is to relate the mean of the distribution function μ and the linear predictor η . In order to choose the suitable link function among available ones, several factors will be considered. The mean and linear predictor are alike in conventional linear models, in which η and μ can have any content. Nevertheless, for Poission function, μ can only have positive values. When it comes to binomial distribution μ can get values between zero and one. Here are three principal functions [12]:

1. Logit link

$$\eta = \log \mu / (1 - \mu) \quad (8)$$

2. Probit link

$$\eta = \Phi^{-1}(\mu) \quad (9)$$

3. Complementary log-log link

$$\eta = \log(-\log(1 - \mu)) \quad (10)$$

2.2.1.3 Logistic regression

Logistic regression is a form of GLM method and is based on the same formula, but is suitable when it comes to modeling binary target variables. These kinds of problems are considered as a classification problem. Logistic regression is based on the logistic function which can be shown as follows [28]:

$$F(x) = \frac{1}{1 + e^{-x}} \quad (11)$$

In order to make an estimation of an occurrence, it is good to know the probability that it fits a specific category. The logistic function ranges from zero to one, so it can be applied for calculating category probability. When there are two categories like zero and one, the logistic function can be altered by presenting parameterized predictor variables the probability of response variable being one can be shown as follows [17]:

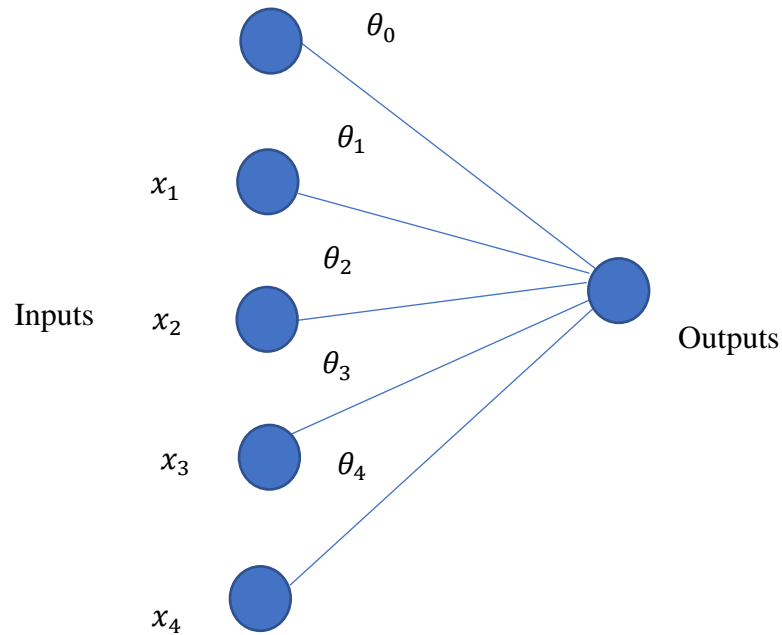


Figure 4. Logistic Regression

$$X = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 \quad (12)$$

$$F(Y = 1|X) = \frac{1}{1 + e^{-X}} \quad (13)$$

Here weights are shown by θ and inputs are represented by x . θ_0 is the bias which is an independent constant value which changes the decision boundary [16]. As it can be seen from the formula inputs are multiplied by corresponding weights and their values are added in order to estimates the value of y [17].

2.2.2 Neural Networks

2.2.2.1 Biology of Neural Networks

Information in human brain is processed through neurons, each neuron consists of various parts such as soma, axon and dendrites. Dendrite is in charge of receiving pulses and axon is responsible for transmitting pulses. There is a basic structure between two neurons known as synapses, which via chemical reactions improve or restrain the pulses between neurons. Synapses are capable of improving and learning through activities they participate in and also pulses that pass through them. There are about 10^{11} neurons in cerebral cortex which complexly are connected to each other. The neurons talk via pulses with a frequency range of not more than hundred hertz. Parallel processing is done in the brain but still, data transfer between neurons is disseminated into interconnections instead of being done in massive amounts [19].

2.2.2.2 Artificial Neural Networks

Artificial neural networks (ANNs) are computational models which have been developed based on human brains. In an ANN, inputs are multiplied by weights and then passed through a mathematical function that controls the triggering of the neuron. Computations depend on the weights, so by modifying the weights, the optimal output can be reached [20].

2.2.2.3 Architecture of Artificial Neural Network

The architecture of the network in this thesis contains three layers; the input layer, the hidden layer, and the output layer.

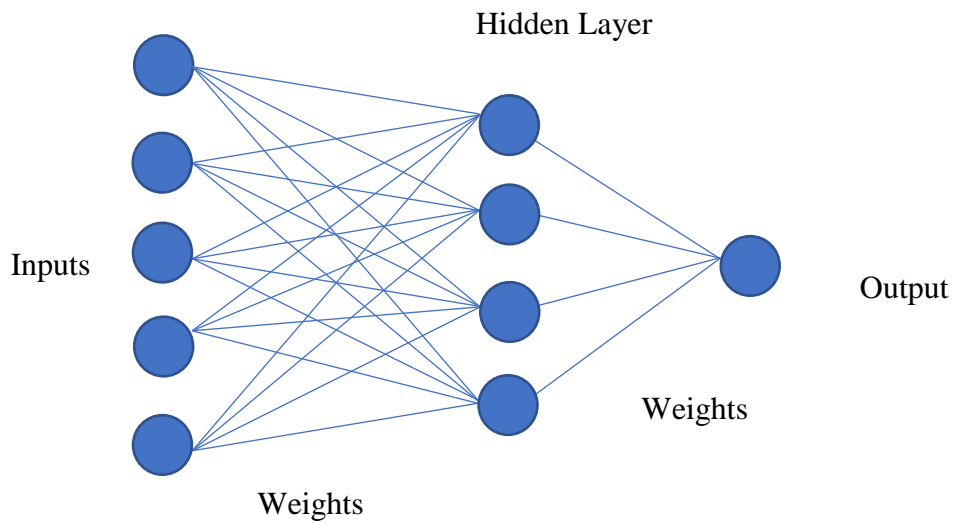


Figure 5. Architecture of Neural Networks

Diverse structures have been tested in order to find the most optimal result. The best structure has been consisting of 733 input neurons, 9 hidden layer neurons and one output neuron which shows the winning or losing probability. Test and validation data percentage has also been tested with various ratios and using no data for testing and validation has shown the best result. The number of iterations has been set to 150 for second order networks since the usage of Hessian matrix for more accurate estimation in second order algorithms actually lead to much faster convergence than first-order algorithms and 10000 for first order networks. The network was created based on patternet which is a feedforward network. The default hidden layer for patternet network is tan-sigmoid and for the output layer is softmax [18].

2.2.2.4 The Error function (Performance function) in training

The training starts by initializing the weights and biases and continues by adjusting them in order to improve network performance, the most common performance function in feedforward networks is mean square error (MSE), which is computed based on outputs produced by the network and the actual results. The formula is defined as follows [21]:

$$F = mse = \frac{1}{N} \sum_{i=1}^N (e_i)^2 = \frac{1}{N} \sum_{i=1}^N (t_i - a_i)^2 \quad (14)$$

common mean square error functions are not always the best choice. Especially in classification problems the maximum likelihood or cross entropy error function (CE) has shown greater results in case of number of epochs required for training and also generalization ability of the network. The CE function is defined as follows [22]:

$$e = md \log \frac{md}{my} + (1 - md) \log \frac{1 - md}{1 - my} \quad (15)$$

Since the anticipated and actual outputs should be scaled from -1 to +1 range to 0 to +1 range so that they can be considered as probabilities, my and md can be specified as $my = \frac{y+1}{2}$ and $md = \frac{d+1}{2}$. [22]

There are various optimization algorithms for training neural networks, in this study all MATLAB built-in optimization algorithms have been tested and the results have been presented. These algorithms use the gradient of the error with respect to the weights which is completed via the backpropagation formula [21].

2.2.2.5 Backpropagation

Backpropagation is a widely used algorithm in training Neural Networks. The primary step in training the network is providing inputs and let the network produce an output for each layer, then computing the error based on the network's outputs and actual results. As it has been mentioned before, the error can be calculated using different error functions. Here, one of the widely used error functions is the CE function. A commonly used activation function is the logistic function:

$$y = \frac{1}{1 + e^{-net}}$$

The activation function is applied to the weighted sum of the neurons inputs:

$$net = \sum_{i=1} x_i w_i + w_0$$

Where w_i is the i^{th} parameter of the model. For consistency with respect to current literature, the parameters of a neural network will henceforth be notated by W_i instead of θ_i . Where w_0 is the bias. Basically, the backpropagation algorithm is using the chain rule for the neurons in every layer [23].

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial net} \frac{\partial net}{\partial w_i}$$

$$net = \sum x_i w_i + w_0$$

$$y = \text{activation}(net)$$

$$E = \text{error}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial net} \frac{\partial net}{\partial w_i}$$

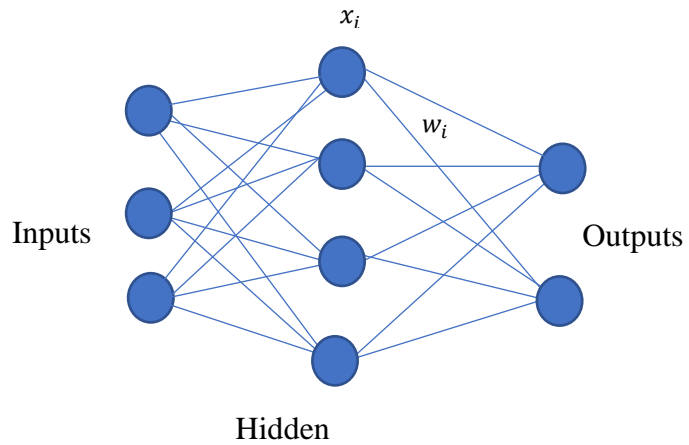


Figure 6. Basic Structure of Backpropagation

The objective with backpropagation is to alter the weights w_i up until the point that the output of the network is closer to the expected output, which means to minimize the error of the network outputs. One of the popular optimization algorithm used along with backpropagation is gradient descent. Calculating gradients and reducing the error until the point where the model has extracted patterns from the training data is the next step in the process [35]:

$$\nabla E = \left(\frac{\partial E}{\partial \omega_1}, \frac{\partial E}{\partial \omega_2}, \dots, \frac{\partial E}{\partial \omega_n} \right) \quad (16)$$

The training starts by random weights, then they will be changed up until the point that error is minimal.

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} \quad (17)$$

Here w_i is the representation of input weights, error is shown by E and η is the learning rate [35].

2.2.2.6 Training Algorithms

Selecting the algorithm for training the network is a key step since it will significantly affect the prediction result. In this thesis, various backpropagation algorithms have been analyzed and compared which have been briefly explained below [21].

2.2.2.7 Gradient descent backpropagation

The training function that uses gradient descent backpropagation algorithm is called 'traingd'. Gradient descent is the most common optimization method which adjusts the network weights and biases in the way that error function declines the fastest. It iterates in the following format until the point that network convergence:

$$w_{k+1} = w_k - \eta g_k \quad (18)$$

In this equation, the learning rate is shown by η , the vector of weights and biases are represented by w_k the gradient is specified by g_k [21].

2.2.2.8 Gradient descent with momentum backpropagation

The training function that uses gradient descent with momentum backpropagation algorithm is called 'traingdm'. In this algorithm alongside the error which is calculated in backpropagation, variation tendency of the error curve is also considered [32]. In each

iteration, an additive value will be added to the value of previous weights and thresholds and they will be updated based on the following formula:

$$\Delta X(k + 1) = mc \times \Delta X(k) + \eta \times mc \times \frac{\partial E}{\partial X} \quad (19)$$

Here mc is the momentum factor which is usually 0.95, the training time is shown by k, the error function is represented by E and η is a constant and is representation of the learning rate [32].

2.2.2.9 Gradient descent with momentum and adaptive learning rate backpropagation

The training function that uses gradient descent with momentum and adaptive learning rate backpropagation is called ‘traingdx’. Learning rate is a principal factor for convergence in backpropagation algorithm and an unsuitable choice of learning rate will lead to slow speed of convergence. The advantage of self-adaptive learning rate algorithms is that the learning time is reduced since the learning rate here is updated automatically. The training function used in momentum is also used here, the only difference is that η (learning rate) is not fixed and is a variable here [32]:

$$\eta(k + 1) = \begin{cases} 1.05\eta(k) \\ 0.07\eta(k) \\ \eta(k) \end{cases} \quad (20)$$

2.2.2.10 Resilient Backpropagation

The training function that uses Resilient backpropagation algorithm is called ‘trainrp’. Resilient Backpropagation (RPROP) updates weights directly based on local gradient which is basically different from algorithms that work based on learning rate [32], the advantage of the algorithm is that gradient does not have a negative impact on the updating process. In order to accomplish that purpose for every weight, an individual update-value Δ_{ij} has been introduced [24]:

$$\Delta w_{ij}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)}, & \text{if } \frac{\partial E^{(t)}}{\partial w_{ij}} > 0 \\ +\Delta_{ij}^{(t)}, & \text{if } \frac{\partial E^{(t)}}{\partial w_{ij}} < 0 \\ 0, & \text{else} \end{cases} \quad (21)$$

The basic idea behind update-value is that when the derivative of the performance function is positive, weights are reduced by update-value and if the derivative of the performance function is negative, and the weights are increased by the update-value [24], [32].

2.2.2.11 Conjugate gradient backpropagation with Fletcher-Reeves updates

The training function that uses conjugate gradient algorithm with Fletcher-Reeves updates is called ‘traincgf’. The idea behind backpropagation algorithm is altering weights and biases in negative direction of gradient which is the direction that performance function declines most rapidly. It has been observed that decreasing most rapidly along negative of the gradient does not certainly lead to fastest convergence. The way conjugate gradient works is searching along a direction that generates faster convergence than gradient descent, the direction is named conjugate direction. In order to specify the step size, searching will be made along the conjugate gradient direction which will lead to quicker convergence [20].

$$p_0 = -g_0 \quad (22)$$

Here the step size is estimated through a line search, avoiding computing Hessian matrix for specifying the optimum distance and move along the present search direction:

$$w_{k+1} = w_k + a_k p_k \quad (23)$$

The way to define the new search direction is to merge the new steepest descent direction with the former search direction:

$$p_k = -g_k + \beta_k p_{k-1} \quad (24)$$

The way constant β_k is calculated determines the version of gradient algorithm. The Fletcher-Reeves is updated as follows [20]:

$$\beta_k = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}} \quad (25)$$

2.2.2.12 Conjugate gradient backpropagation with Polak-Ribière updates

The training function that uses conjugate gradient algorithm is called ‘traincgp’. This algorithm has been developed by Polak and Ribière and the search direction is the same as Fletcher-Reeves algorithm:

$$p_k = -g_k + \beta_k p_{k-1} \quad (26)$$

In this version of gradient decent constant β_k is calculated as follows [20], [37]:

$$\beta_k = \frac{\Delta g_{k-1}^T g_k}{g_{k-1}^T g_{k-1}} \quad (27)$$

2.2.2.12.1 Line Search

A line search algorithm is a part of a larger optimization approach that aims to find the minimum of a specified function. At each iteration, the search direction d_k is computed through:

$$x_{k+1} = x_k + \alpha_k d_k \quad (28)$$

Here α_k is the step length, and x_k is the current iteration. The line search algorithm chooses the step size that after a fair amount of iterations, generate a value, close to the global minimum of the function. There are several line search algorithms available to choose from. Srchcha is a common search function that searches in a specified direction to find the minimum of the performance function. For that purpose, it uses a technique developed by

Charalambous. Srchcha is the default search function for majority of conjugate gradient algorithms since it has shown great outcomes for various problems [25].

2.2.2.13 Conjugate gradient backpropagation with Powell-Beale restarts

The training function that uses conjugate gradient backpropagation with Powell-Beale restarts is called ‘traincgb’. The common approach in conjugate gradient methods is to iteratively reset the search direction to the negative of the gradient. Originally the reset point happens when number of the iterations and network parameters are equivalent to each other. Powell introduces a novel approach, in which the reset point is reached when the perpendicularity among the current gradient and the former gradient is adequately small. This will be verified as follows:

$$|g_{k-1}^T - g_k| \geq 0.2 \|g_k\|^2 \quad (29)$$

In this method, the search direction will be rest to negative of the gradient when the above inequality is satisfied [26], [27].

2.2.2.14 Scaled conjugate gradient backpropagation algorithm

The training function that uses scaled conjugate gradient algorithm is called ‘traincsg’. All the above explained conjugate gradient algorithms require line search which can take a long time to perform. In this algorithm usage of a step size scaling process will lead to avoiding the computationally expensive line search in each iteration [29].

2.2.2.15 One-step secant backpropagation

The training function that uses one-step secant backpropagation algorithm is called ‘trainoss’. In this algorithm, the amount of computation and storage usage is quite fewer than the BFGS algorithm, but in compare to conjugate gradient methods, it is fairly higher. In this method, the computation of matrix inverse is not needed for calculation of the new search direction, since the complete Hessian matrix is not stored at each step. Every variable is updated based on the subsequent formula:

$$X = X + a * dX \quad (30)$$

Here a is chosen to make the performance function minimized along the search direction which is represented by dX . Negative of the gradient initializes the search direction and in later steps, it is computed based on the former gradients and steps and the new gradient as follows:

$$dX = -gX + Ac * X_{step} + Bc * dgX \quad (31)$$

Here gradient is shown by gX , weight changes is represented by X_{step} and gradient change is shown by dgX [29].

2.2.2.16 Quasi-Newton(QN)

The training functions that use Quasi-Newton algorithms is called ‘trainbfg’. The Newton algorithm has shown better results than conjugate gradient backpropagation because of its faster convergence. A basic step in Newton method is shown below:

$$w_{k+1} = w_k - H_k^{-1} g_k \quad (32)$$

Where H_k^{-1} is the Hessian matrix of the performance index at the current iteration. It is regarded as quite costly and complex for feedforward neural networks to calculate the Hessian matrix. To overcome this issue a class of algorithms called Quasi-Newton methods has been developed that are based on Newton’s method, this algorithm approximates the inverse of the Hessian matrix [29].

2.2.2.17 Levenberg-Marquardt algorithm

The training function that uses Levenberg-Marquardt learning rate algorithm is called ‘trainlm’. The need for an algorithm that approaches second order training speed without calculating Hessian matrix led to the development of Levenberg-Marquardt. In cases that the

performance function is based on sum of squares, Hessian and gradient can be estimated as follows:

$$H = J^T J \quad (33)$$

$$g = J^T e \quad (34)$$

The networks errors are represented by e here, the Jacobian matrix shown by J , includes the derivatives of the network errors with respect to weights and biases. A suitable approach is to calculate Jacobian matrix via backpropagation method. The calculation below is applied by LM method to the Hessian matrix through the subsequent updating method:

$$w_{k+1} = w_k - [J^T J + \mu I]^{-1} J^T e \quad (35)$$

Here w represents connection weights. When μ has zero value, the formula simply turn into Newton's algorithm and in case μ has a large value, the equation will turn into gradient descent. The perfect approach here is to move towards Newton's algorithm since it is quicker and more precise around a minimum error, therefore, after every successful step, μ is reduced [20].

2.2.2.18 Bayesian regulation backpropagation

The training function that uses Bayesian regularization algorithm is called 'trainbr'. Bayesian regularization basically works based on LM algorithm. This algorithm tends to make combination of squared errors and weights minimum [32]:

$$jj = jw * jw \quad (36)$$

$$je = jw * E \quad (37)$$

$$dw = -(jj + I * mu) \setminus je \quad (38)$$

Here the Jacobian jw is computed using backpropagation with respect to the weight and bias variables w . The identity matrix is represented by I and E is the Error [36]. This approach also incorporates information about the functional form of hyper priors and priors of the weights. When transformed by the likelihood estimate for data given weights, this gives a regularized point estimate in the posterior, thus alleviating the risk of overfitting.

2.3 Support vector machines

The Support Vector Machines (SVM) is a classification algorithm and initially was launched for binary classification. The objective of the model is to discover and draw the best decision surface (optimal hyperplane) between the classes which mean the surface that has the biggest margin (distance) from nearest samples of each category.

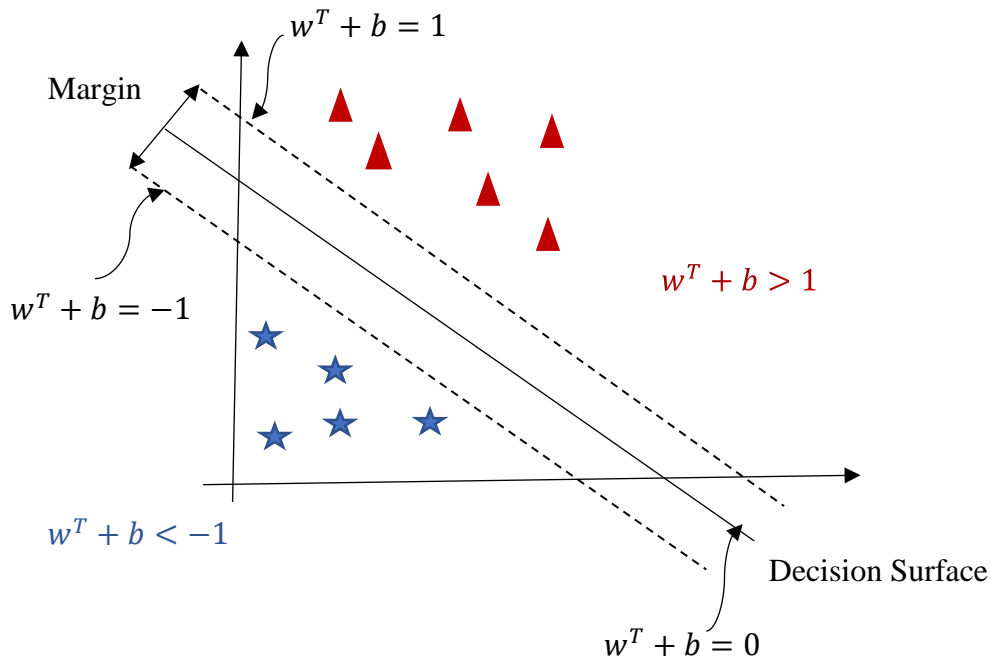


Figure 7. linear binary support vector machine classifiers

Suppose there are a certain number of objects $x_i \in R^d$ belonging to two groups, $y_i \in \{1, -1\}$. The hyperplane between two classes is drawn in a way that all samples of the first class are on one side and all samples of the second class are on the other side of the line. In an n -dimensional space hyperplane is defined by:

$$w^T x + b = 0 \quad (39)$$

Where x is a point on the boundary decision, b is bias and w is the weight vector. It is assumed that components of the dataset satisfy the subsequent relations:

$$\begin{cases} \text{if } y_i = 1 & \Rightarrow w^T x_i + b \geq 1 \\ \text{if } y_i = -1 & \Rightarrow w^T x_i + b \leq -1 \end{cases} \quad (40)$$

which can then be combined into:

$$y_i (w^T x_i + b) \geq 1 \quad (41)$$

By replacing $\|w\|$ with $\frac{\|w\|}{2}$ and using Lagrange Multipliers α_i , the above formula can be rewritten as:

$$\min_{w,b} \max_{\alpha \geq 0} \left\{ \frac{\|w\|^2}{2} - \sum_{i=1}^n \alpha_i ((w \cdot x_i - b)y_i - 1) \right\}, \alpha_i \geq 0 \quad (42)$$

Since the data is not always spread in a linear form, it is needed to have a substitute model with a soft margin that tolerates a few misclassified samples. In order to implement this, slack variable ξ_i is introduced, which allows x_i to not meet the margin requirements at the cost of adding the value of ξ_i to the function that needed to be minimized. The following form is the most common way to write it:

$$\frac{\|w\|^2}{2} + C \sum_{i=1}^n \xi_i \quad (43)$$

The subsequent revised constraint should also be applied to this minimization:

$$y_i (w^T x_i + b) \geq 1 - \xi_i \quad (44)$$

Applying the Lagrange Multipliers again will, the problem can be rewritten as the following form:

$$\min_{w,b,\xi} \max_{\alpha,\beta} \left\{ \frac{\|w\|^2}{2} + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n (\alpha_i (y_i (w^T \cdot x_i - b) - 1 + \xi_i) - \beta_i \xi_i) \right\} \quad (45)$$

When $\alpha_i, \beta_i \geq 0$. In order to eliminate slack variables, this relation can be rewritten in its dual form:

$$\max_{\alpha} \left\{ \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \right\} \quad (46)$$

considering the following constraints: $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^n \alpha_i y_i = 0$ [30] [31].

2.4 Method of work

The initial phase of the project comprised of data analysis. Based on the patterns in the dataset it was decided to divide the dataset into two categories of continuous data and categorical data. One hot encoding was applied on the categorical data in order to convert the categorical data to nominal data which works significantly better with classification algorithms and on the later part the normalization formula was performed in order to bring the dataset to a common scale. The preprocessed dataset was fed into Generalized Linear models, Neural Networks and Support Vector Machines and their generated results were evaluated.

In order to evaluate the performance of the models, data from the first day of the database up until forty days later was fed into the model, so that the model could predict results for the next day of that period. This model fitting and validation algorithm iterates on a simulated daily basis until end of dataset. The mean values for both losers and winners have been computed separately and conclusive results for each model has been presented so that the best model can be selected.

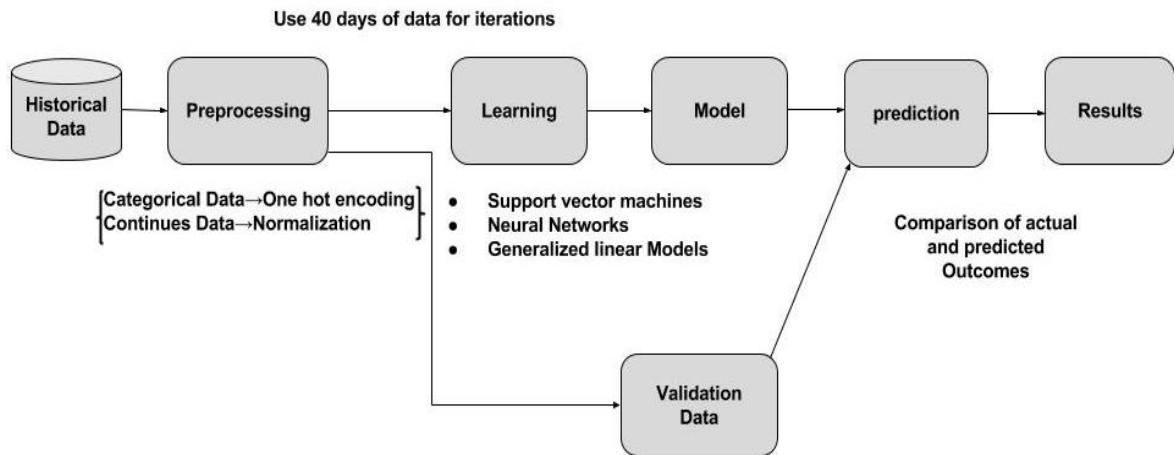


Figure 8. Illustration of method

Table 3. Data Preprocessing

Algorithm 1. Data Preprocessing
1. Input: Dataset X
2. Output: Preprocessed dataset \hat{X}
3. For each column $x_i \in X$
4. If x_i is continuous
$\hat{x}_i = \frac{x - \text{mean}(x)}{\text{stdev}(x)}$
5. End if
6. If x_i is categorical
$\hat{x}_i = \text{One_Hot_Encode}(x_i)$
7. End if
8. Return $\hat{X} = (\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n)$

The whole dataset is consist of two classes of categorical data and continues data. One hot encoding algorithm is applied to categorical data in order to transfer the categorical part of data to nominal data since nominal data works notably better with machine learning

algorithms. On columns of data that comprise continuous data, a normalization formula is applied to transport the data to a common scale.

Table 4. Evaluation process

Algorithm 2. Evaluation process
<ol style="list-style-type: none"> 1. Input: Preprocessed dataset X 2. Output: winning or losing probability 3. $R \leftarrow 40$ 4. For $i=1$: all days-R 5. Fit model on all days in range i : $R+i-1$ 6. Generate the model 7. Use the model to predict the result for day $R+i$ 8. End for 9. Compute the average of the winning and losing probabilities for the whole dataset separately. <p>* i correspond to an iteration step for a single day.</p>

After preprocessing, data is ready for training. The objective is to feed the data set to different algorithms and use the generated models to predict the result. The way that validation process works is by choosing 40 days of data from the beginning of dataset and feeding data of those days to the algorithms and using the generated model to predict the results for the next day, then it will move one day forward, and again train the model on the next 40 days of data and predict the result for the next day. This process will go on iteratively till the end of the data (Figure 9). Since we already have the results for those next days in our dataset we can see how well the model has learned and how promising are the results of the model. The average of probabilities for both losers and winners will be computed separately, obviously the satisfactory results for winners should be close to one and the probability of winning for losers should be close to zero.

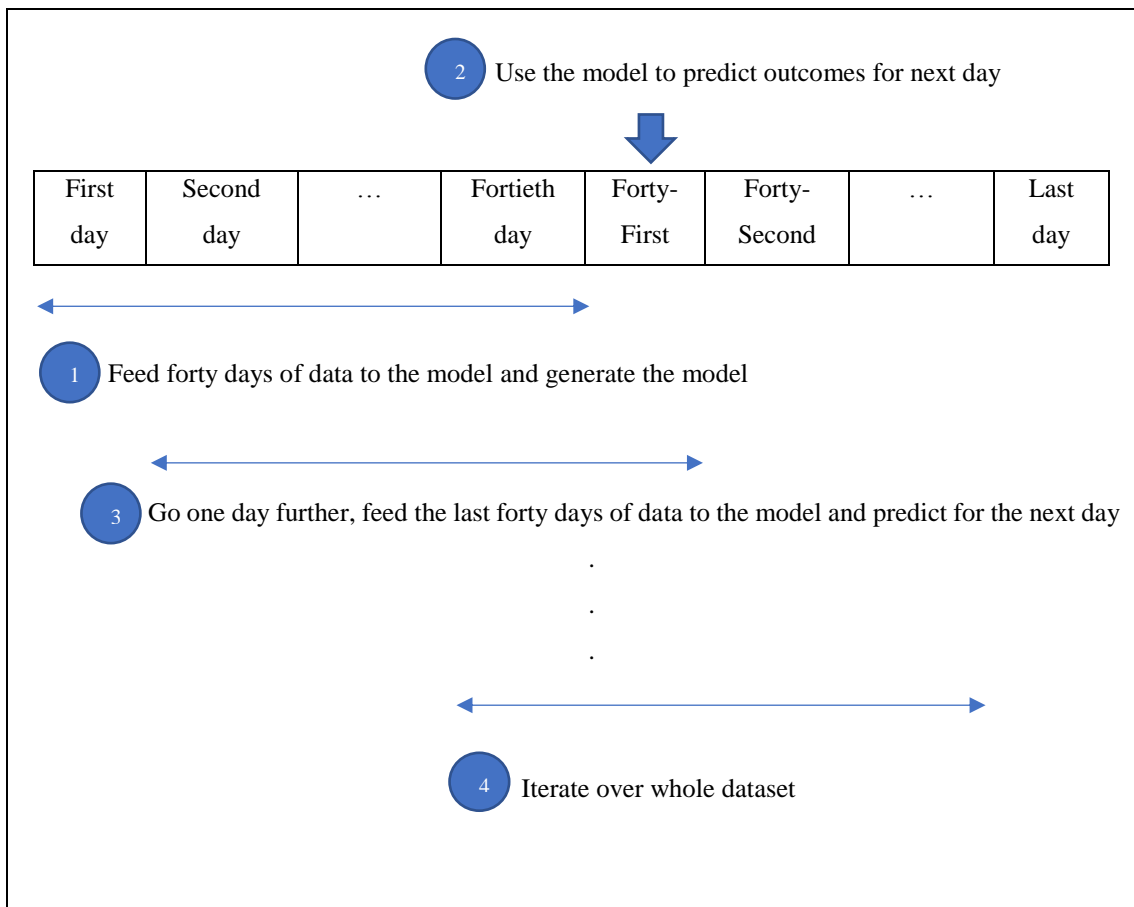


Figure 9. Illustration of validation

3 DATA ANALYSIS AND RECOMMENDATION

3.1 Initial assessment

The dataset used in this study is historical data for past two years from WideOrbit company. The early stage of the study included data analysis. The patterns in the dataset indicated two categories of continuous data and categorical data. In order to scale the data, two preprocessing algorithms have been applied to the dataset. One hot encoding has been used for the categorical data with the intention of converting the categorical data to nominal data that works remarkably better with classification algorithms and on the continues part of data the normalization formula has been applied with the purpose of conveying the dataset to a common scale. After preprocessing, statistical models are applied to the data and their generated outcomes are compared in order to find the best method. In the following section, a detailed analysis is represented:

3.2 Detailed data analysis

In this section, various training algorithms applied in this thesis and their predicted results based on different structures will be presented. Prediction of winning or losing in percentage has been shown in graphs below for different algorithms and using different structures of the method. Typically, hidden layer size is a principal factor in shaping the structure of the Neural Network. The tools that were utilized are Microsoft excel and MATLAB exclusively. The average of results for both winners (mean positive) and loser (mean negative) have been presented separately and also in one graph in order to demonstrate a better perspective of the outcomes. It is clear that a perfect model would outcome one for winners and zero for losers.

3.2.1 Neural Networks

Several training algorithms that have been applied in order to find the best performance and their generated outcomes have been presented in the following section.

3.2.1.1 Conjugate Gradient Backpropagation with Polak-Ribière Updates

The `traincgp` training function is based on conjugate gradient backpropagation with Polak-Ribière updates. As it can be seen from the graphs, `traincgp` has performed well overall for majority of tested structures. The means of predicted values for both positives(winners) and negatives(losers) is represented in figure 10, this graph is suitable to see the general patterns of how the predictions increase or decrease based on different hidden layer size for both winners and losers. But the different range of results make it difficult to see the precise trends in the graphs, therefore two separate diagrams have represented in order to provide a deep understanding of the outcomes.

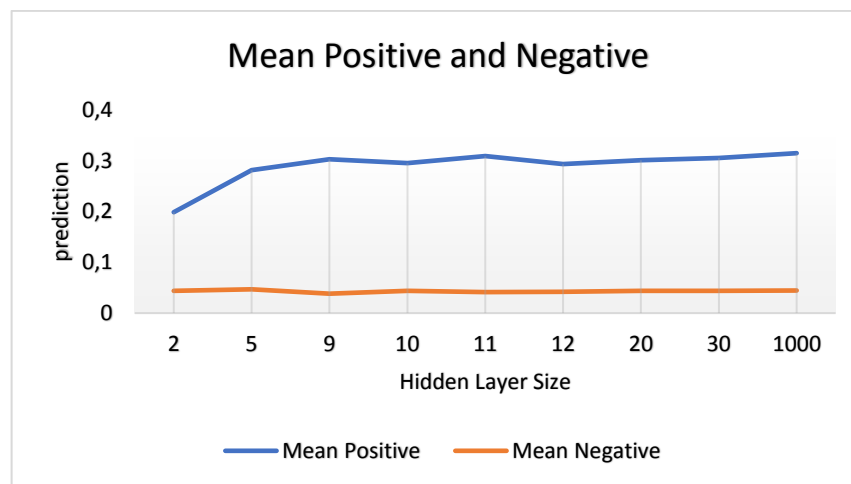


Figure 10. Mean Positive and Negative for conjugate gradient backpropagation with Polak-Ribière updates

Exploring the results in more detail will disclose that the best performance of all algorithms is obtained from the `traincgp` method with 9 hidden layers size structure; with the value of 0,3027 for winners and the value of 0,0382 for losers, which is a superior result in compare to other outputs for losers.

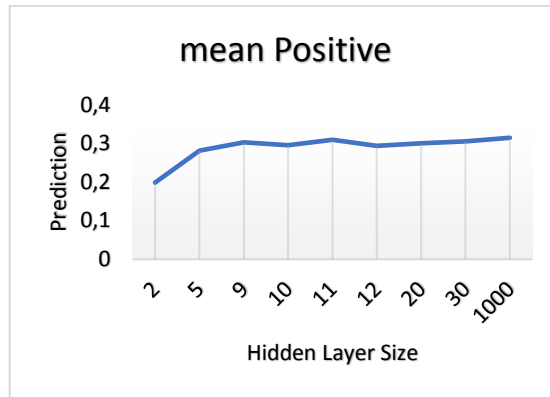


Figure 11. Mean Positive for conjugate gradient backpropagation with Polak-Ribière updates

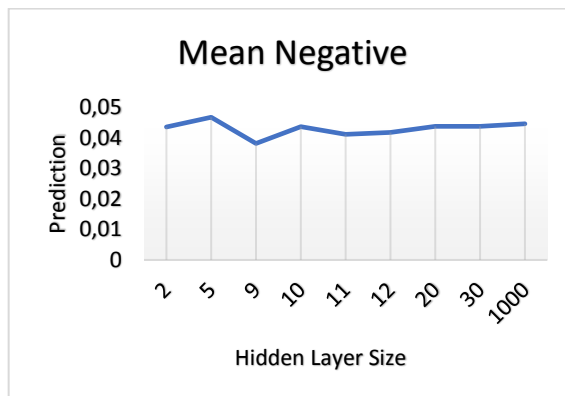


Figure 12. Mean Negative for conjugate gradient backpropagation with Polak-Ribière updates

3.2.1.2 Fletcher-Reeves

The training function `traincgf` is based on Fletcher-Reeves updates. As it can be seen from the graphs several structures have been applied in order to find the best performance.

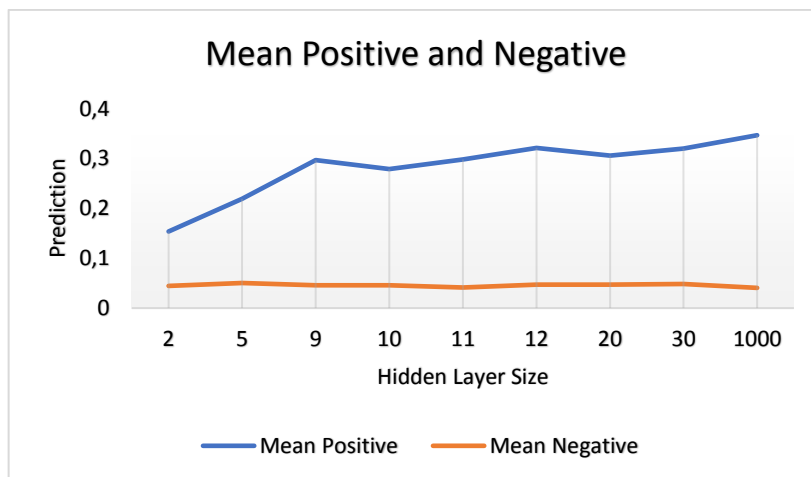


Figure 13. Mean Positive and Negative for Fletcher-Reeves

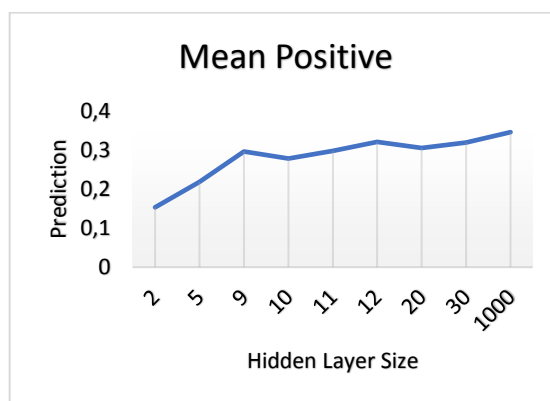


Figure 14. Mean Positive for Fletcher-Reeves

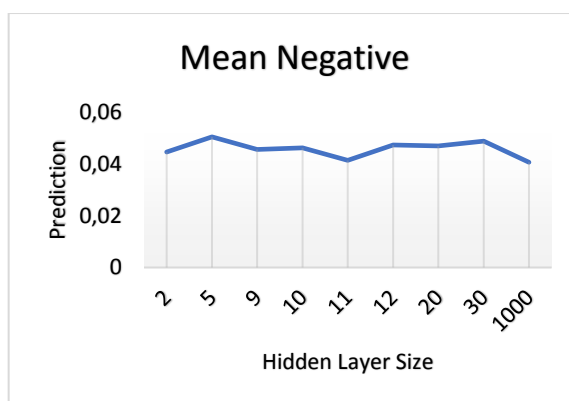


Figure 15. Mean Negative for Fletcher-Reeves

As it can be seen from the results there is not a specific trend between the different structures in this algorithm. The best results are from the hidden layer size 1000 with the value of 0,3470 for mean positives and value of 0,0406 for mean negatives.

3.2.1.3 Levenberg-Marquardt Backpropagation

The training function `trainlm` is based on Levenberg-Marquardt backpropagation algorithm. A general look at the graph might give the idea that this algorithm is performing significantly good, because of superior performance in results for winners.

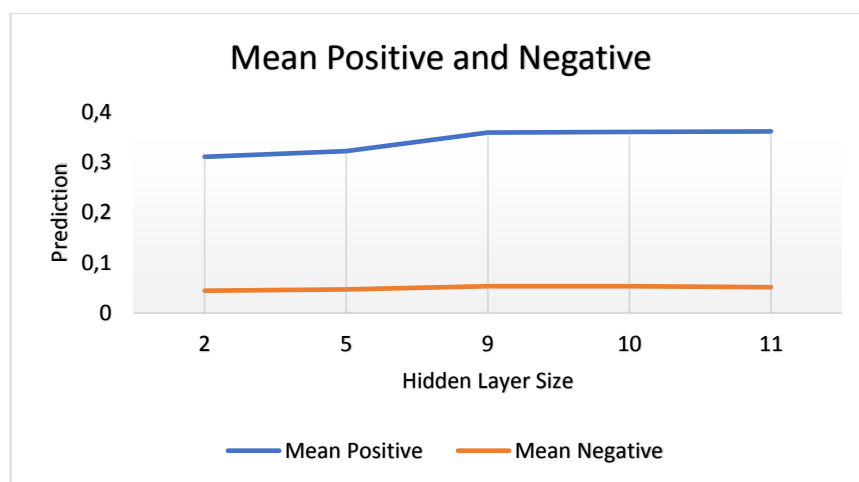


Figure 16. Mean Positive and Negative for Levenberg-Marquardt backpropagation

A closer look, on the other hand, would reveal that the performance of algorithm is not as good as expected, the generated results are very good for winners with values of 0,3601 for 10 hidden layer size structure and value of 0,3612 for 11 hidden layer size structure, but the algorithm has performed extremely poor on losers with values of 0,0532 for 10 hidden layer size structure and value of 0,0513 for 11 hidden layer size structure. In addition, since the number of losers in dataset is about 18 times more than the winners, the output for losers is much more important, so it was decided not to continue with further structures of this method.

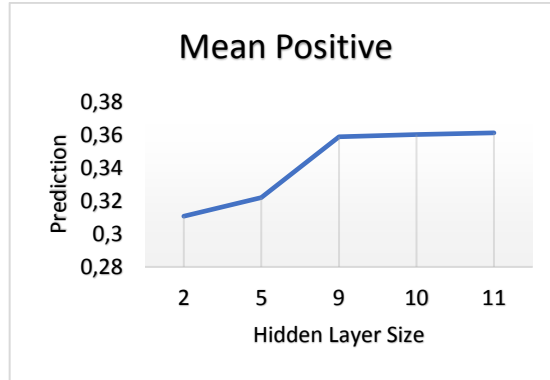


Figure 17. Mean Positive for Levenberg-Marquardt backpropagation

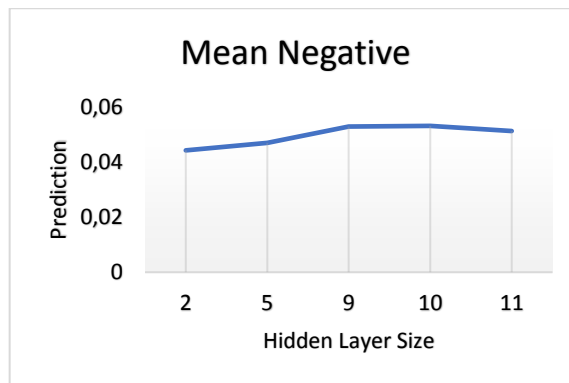


Figure 18. Mean Negative for Levenberg-Marquardt backpropagation

3.2.1.4 Conjugate gradient backpropagation with Powell-Beale restarts

The training function `traincgb` is based on Conjugate gradient backpropagation with Powell-Beale restarts. Looking at the represented graphs the trends in the chart might represent that higher number of neurons would lead to better results.

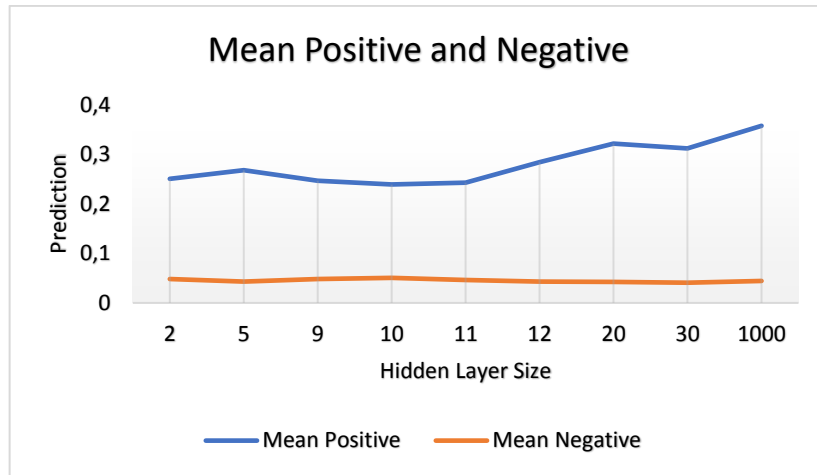


Figure 19. Mean Positive and Negative for conjugate gradient backpropagation with Powell-Beale restarts

but a closer examination shows that there is no trend in this algorithm since the best results is from the 30 neurons in hidden layer size with values of 0,3122 for winners and value of 0,0408 for losers.

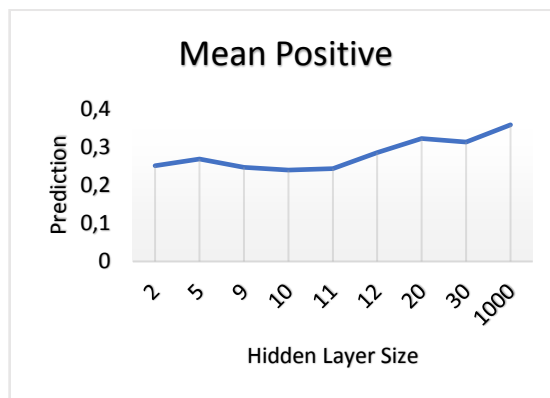


Figure 20. Mean Positive for conjugate gradient backpropagation with Powell-Beale restarts

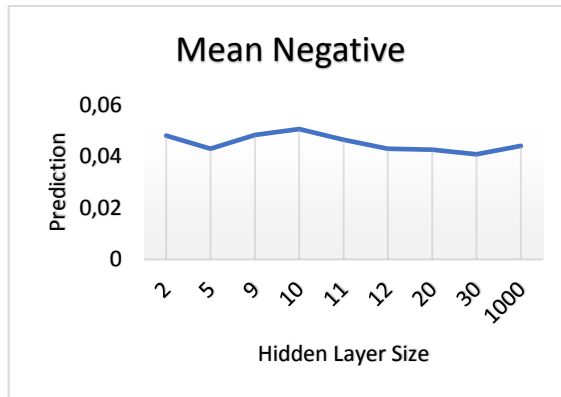


Figure 21. Mean Negative for conjugate gradient backpropagation with Powell-Beale restarts

3.2.1.5 One-step secant Backpropagation

The training function trainoss is based on one-step secant backpropagation algorithm. As it can be seen from the graphs, this algorithm has not shown acceptable results in compare to other algorithms.

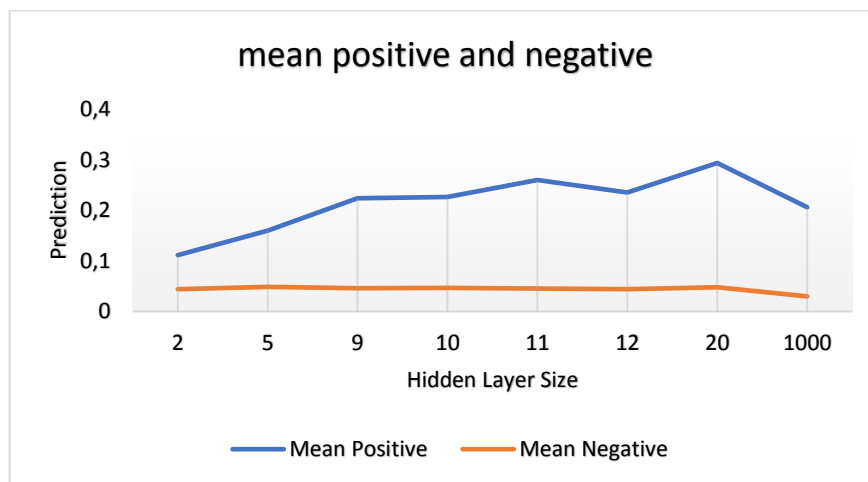


Figure 22. Mean Positive and Negative for one-step secant backpropagation

In this algorithm, the hidden layer size 9 has shown the best results for losers with the value of 0,0423 and the value of 0,2244 for positives. Nevertheless, this algorithm still has a deficient performance in comparison to some of the other methods.

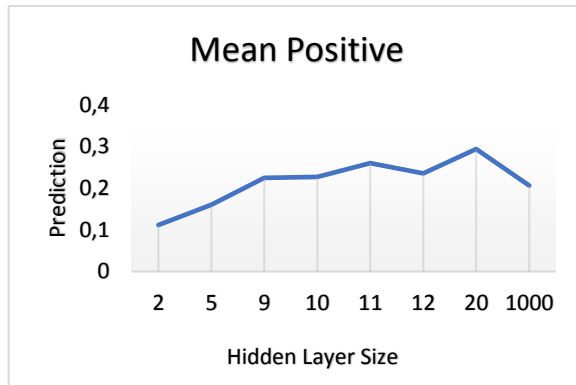


Figure 23. Mean Positive for one-step secant backpropagation

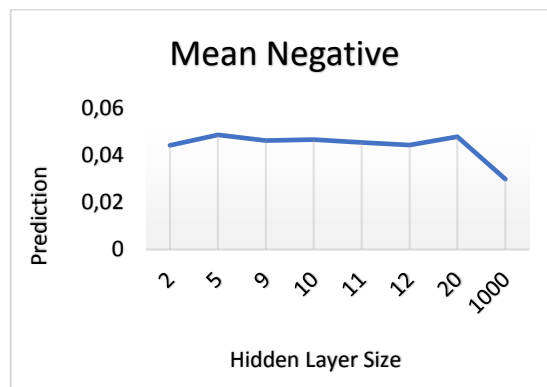


Figure 24. Mean Negative for one-step secant backpropagation

3.2.1.6 Broyden Fletcher Goldfarb Shanno Quasi-Newton Backpropagation

The training function trainbfg is based on Broyden Fletcher Goldfarb Shanno Quasi-Newton backpropagation. Looking at the trends in the graphs it might be concluded that this algorithm is working perfectly fine in prediction both winners and losers.

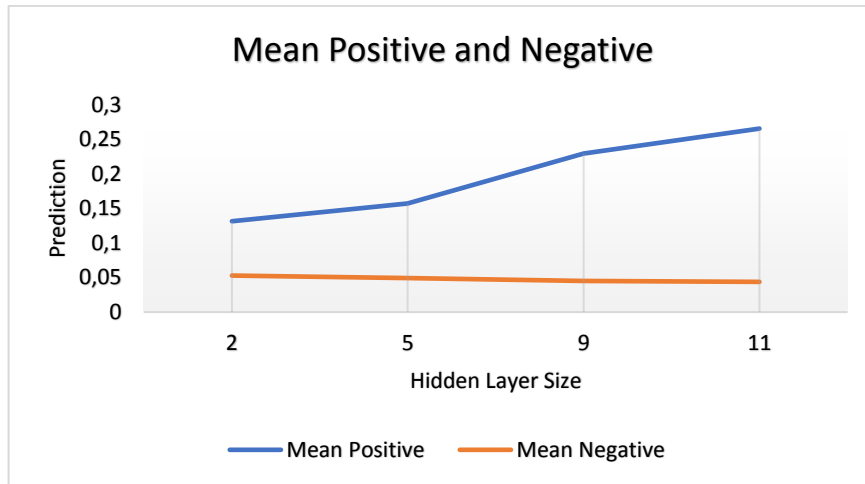


Figure 25. Mean Positive and Negative for Broyden Fletcher Goldfarb Shanno Quasi-Newton backpropagation

Nevertheless, a closer look at the generated result would reveal that the best result which is at 11 hidden layer size, with values of 0,2656 for winners and 0,0438 for losers is not a satisfactory result in compare to other algorithms. On the other hand, since the simulations in this method were significantly slow, it was decided to not examine the further structures for this algorithm.

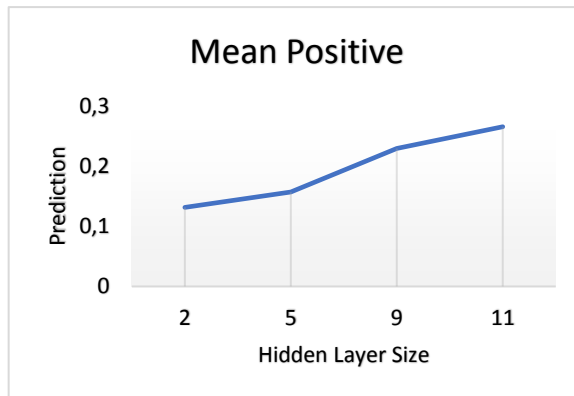


Figure 26. Mean Positive for Broyden Fletcher Goldfarb Shanno Quasi-Newton backpropagation

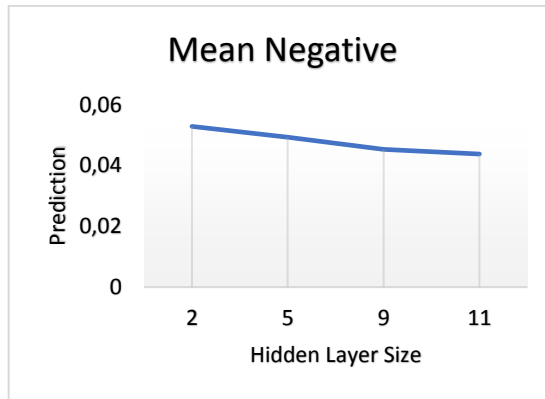


Figure 27. Mean Negative for Broyden Fletcher Goldfarb Shanno Quasi-Newton backpropagation

3.2.1.7 Bayesian Regularization Backpropagation

The trainbr algorithm is based on Bayesian regularization backpropagation. The outcomes of this model are not satisfactory for the losers and since the number of losers is 18 times more than the number of winners in the dataset, it was decided not to continue with further structures of this algorithm.

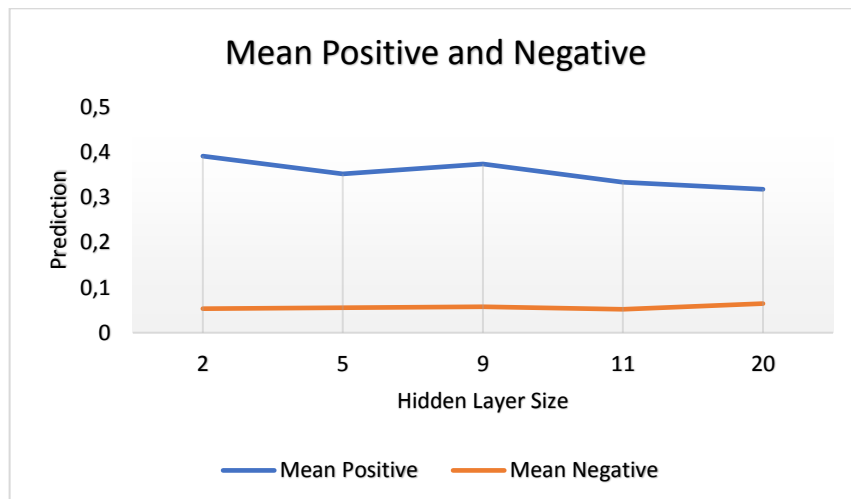


Figure 28. Mean Positive and Negative for Bayesian Regularization Backpropagation

A detailed examination of the graphs would show that best performance in this algorithm is from 2 hidden layers size with value of 0,3917 for winners and value of 0,0533 for losers.

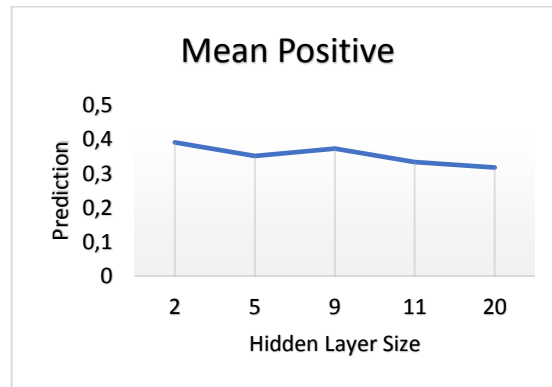


Figure 29. Mean Positive for bayesian regularization backpropagation

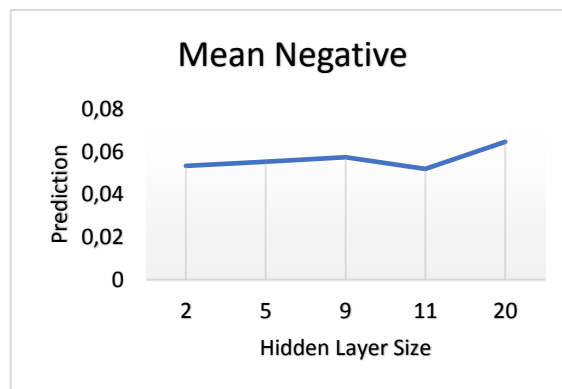


Figure 30. Mean Negative for bayesian regularization backpropagation

3.2.1.8 Scaled Conjugate Gradient

The training function `trainscg` is based on scaled conjugate gradient algorithm. As it can be seen from the graphs, this algorithm has shown satisfactory results in comparison to other algorithms.

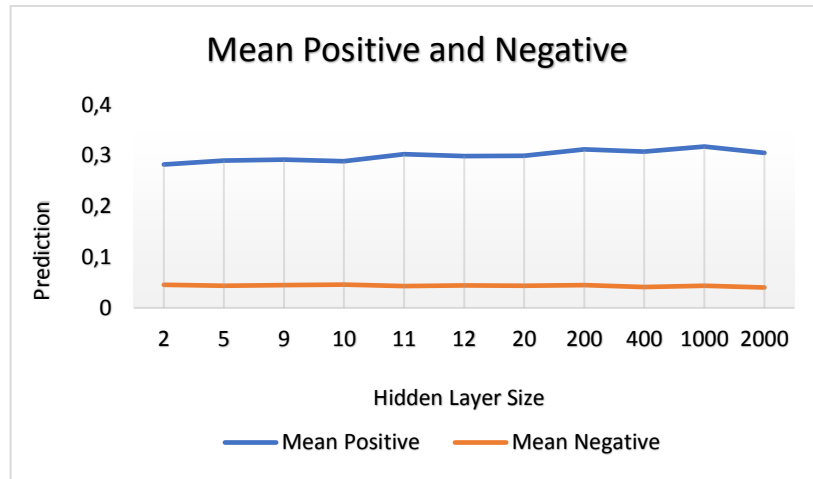


Figure 31. Mean Negative and Positive for scaled conjugate gradient

It can be seen from the graphs that increasing the hidden layer size has shown some improvements in predicted result for the winner but there is no absolute increase in the values since the best result has been reached at 1000 hidden layer size.

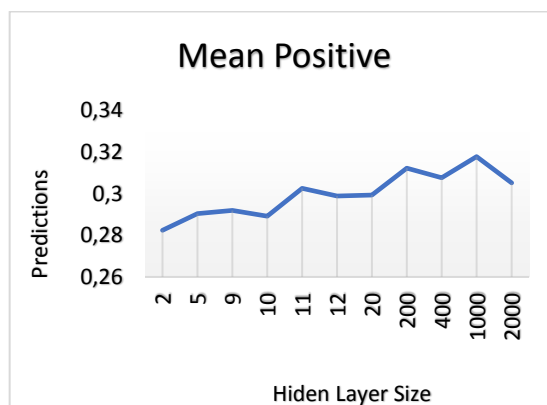


Figure 32. Mean Negative and Positive for scaled conjugate gradient

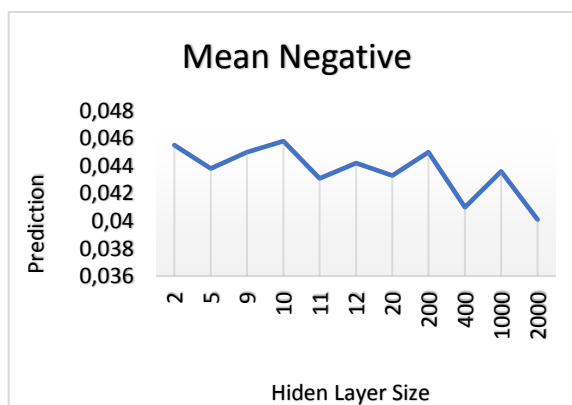


Figure 33. Mean Negative for scaled conjugate gradient

3.2.1.9 Traingdx Gradient Descent with Momentum and Adaptive Learning Rate Backpropagation

The training function traingdx is based on gradient descent with momentum and adaptive learning rate backpropagation. As it is clear from the graphs this algorithm has performed fairly well for prediction both winners and losers.

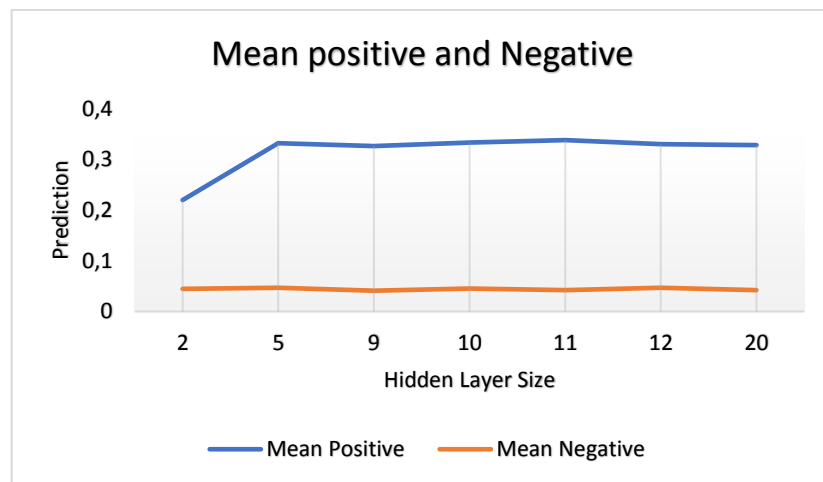


Figure 34. Mean Negative and Positive for gradient descent with momentum and adaptive learning rate backpropagation

Looking closely at the graphs would reveal that although the best results have been, from the 20 hidden layer sizes, there is not a trend between hidden layer size and the output. Besides, in comparison to the previous simulation which was 12 hidden layer size, the algorithm has performed slightly better for predicting losers but also slightly worse for predicting winners.

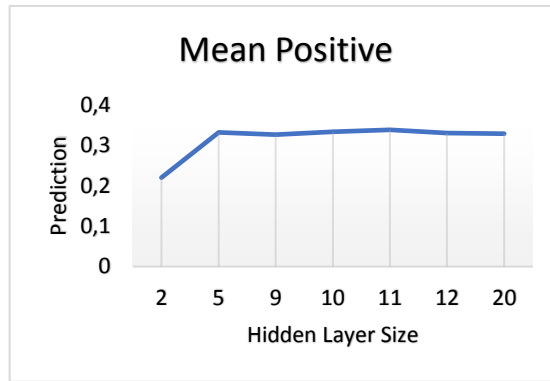


Figure 35. Mean Positive for gradient descent with momentum and adaptive learning rate backpropagation

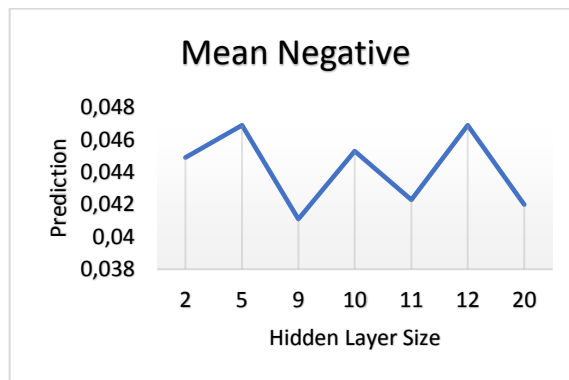


Figure 36. Mean Negative for gradient descent with momentum and adaptive learning rate backpropagation

3.2.1.10 Gradient descent with momentum backpropagation

The training function `traindm` is based on gradient descent with momentum backpropagation.

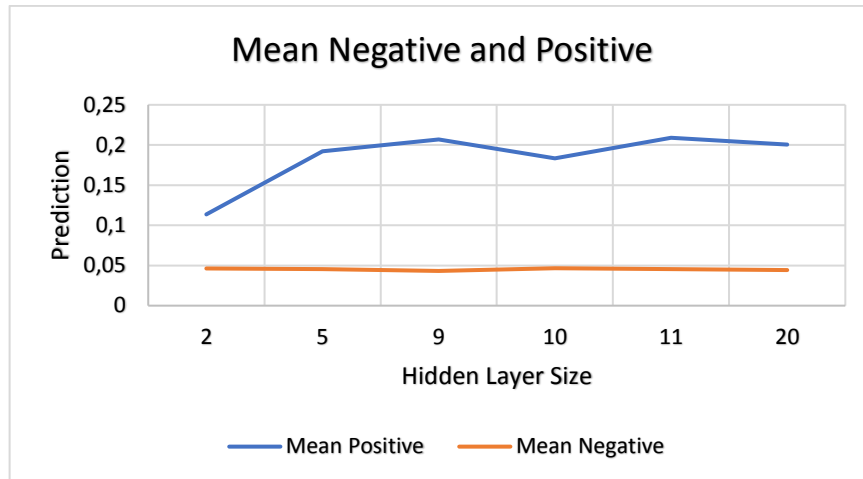


Figure 37. Mean Negative and Positive for gradient descent with momentum backpropagation

Looking at the results it can indicate that, this algorithm has not shown a satisfactory performance since the best result is from 9 hidden layer size with the value of 0,2068 for ones and value of 0,0432 for zeros.

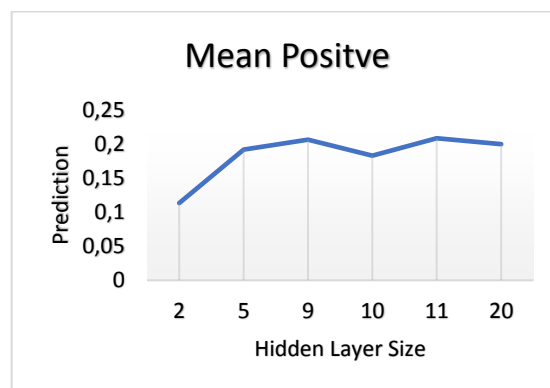


Figure 38. Mean Negative and Positive for gradient descent with momentum backpropagation

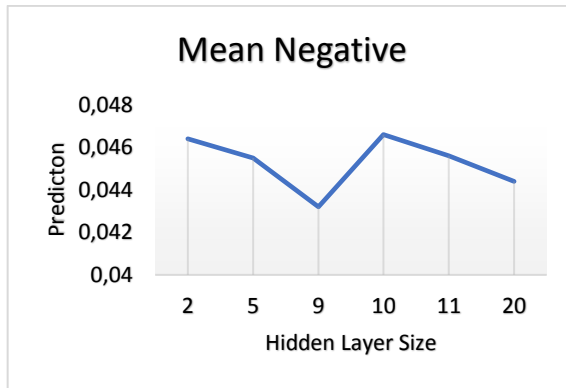


Figure 39. Mean Negative and Positive for gradient descent with momentum backpropagation

3.2.1.11 Resilient Backpropagation

The training function trainrp is based on resilient backpropagation. As it can be seen from the graphs this algorithm has performed extremely poor in predicting winners and losers. The algorithm has been tested with different structures in order to examine its performance, but there have been no improvements in the generated results.

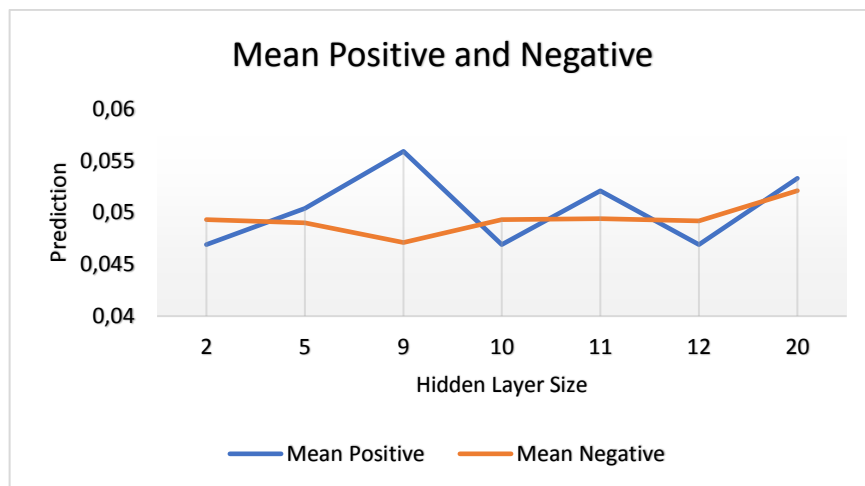


Figure 40. Mean Negative and Positive for resilient backpropagation

Looking closely at the results, it can be said that this algorithm is generating unacceptable results for winners and losers. Results from this algorithm is not even comparable to the outputs in the other algorithms.

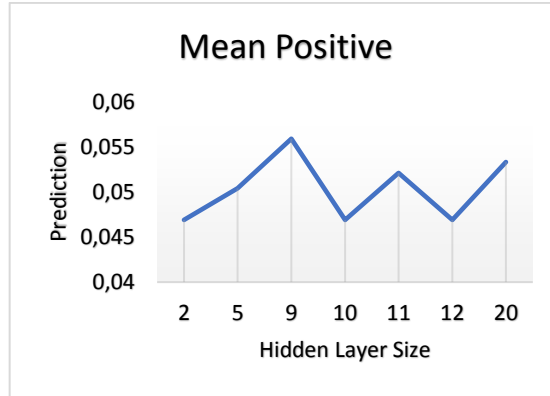


Figure 41. Mean Positive for resilient backpropagation

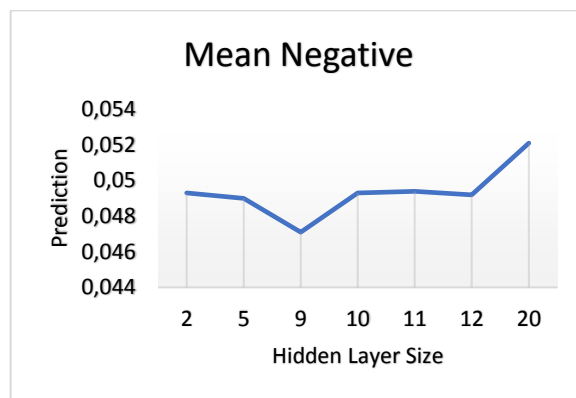


Figure 42. Mean Negative for resilient backpropagation

3.2.1.12 Gradient Descent

The training function `trainscg` is based on gradient descent algorithm. As it can be seen from the graphs, this algorithm has not shown satisfactory results in compare to other algorithms. As the number of hidden layer size increases it has shown poorer results for both negatives and positives therefore, it was decided to not continue with other structures of this algorithm.

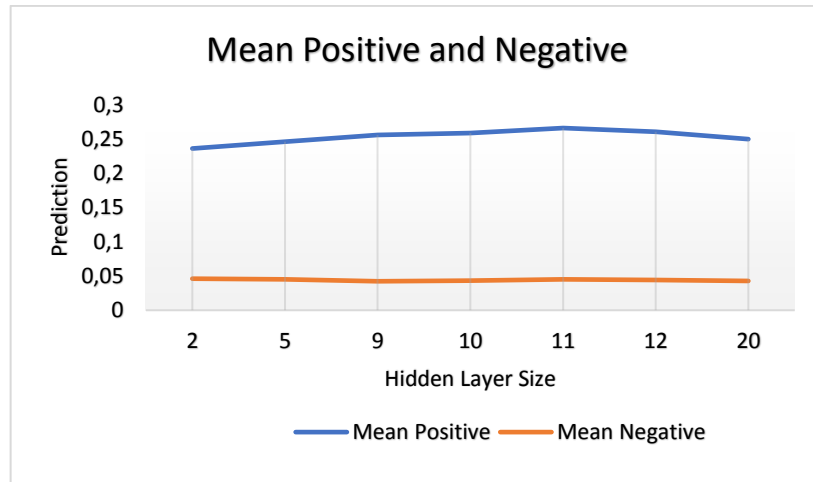


Figure 43. Mean Negative and Positive for gradient descent

In this algorithm, the hidden layer size 9 has shown the best results with the value of 0,0423 for losers and the value of 0,2559 for the winners, which are not satisfactory results.

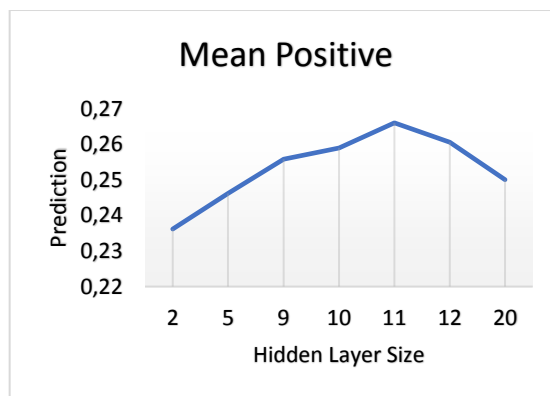


Figure 44. Mean Positive for gradient descent

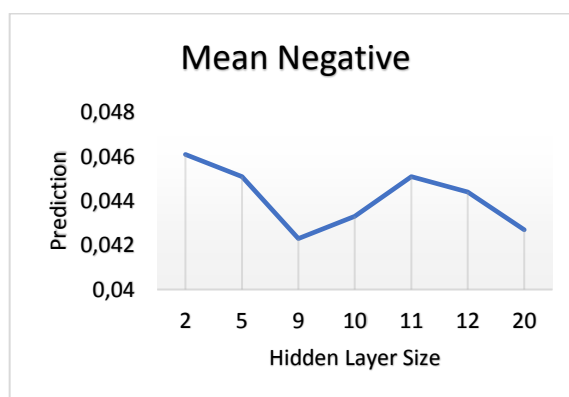


Figure 45. Mean Negative for gradient descent

3.2.2 Generalized Linear Models

The same dataset used in Neural Networks training has been applied to evaluate the output of Generalized Linear Models. This algorithm has performed reasonably well for predicting winners with the mean value of 0,3404 but then the outcome for losers is extremely poor with the mean value of 0,0746.

3.2.3 Support Vector Machines

The very same dataset used in Neural Networks and Generalized Linear Models has been applied on SVM. This method has performed well for both winners and losers with the values of 0,3615 and 0,0480.

3.3 Major findings and recommendations

The purpose of this study was finding out the best model that suits the problem of predicting the most accurate results for bid clearance system in WideOrbit, through examining existing statistical models and reviewing available literatures about the topic. The neural network with training function `traincgp` which is based on conjugate gradient backpropagation with Polak-Ribière updates has performed well overall for assessed structures. As it can be seen from the graphs, the best performance of this method comes from the 9 hidden layers size structure; with the value of 0,3027 for winners and the value of 0,0382 for losers, which is a superior result in compare to other outputs for losers.

4 DISCUSSION AND CONCLUSIONS

4.1 Summary

Accurate predictions for online TV auctions is crucial for customers in order to manage their budgets and time in advance. A range of factors affect the result of predictions of winning or losing the auction; from the preprocessing algorithm used to scale the data to structure of the applied methods. Several steps need to be done in order to find the optimal algorithm that suits the best for a specific problem.

In this thesis, first historical data for past two years from WideOrbit programmatic TV database has been used in order to examine different statistical algorithms' efficiency when predicting outcomes for online TV auctions. After preprocessing the data, three different popular statistical methods of Generalized Linear Models, Neural Networks and Support Vector Machines have been applied in order to predict bid clearance probabilities in first price auctions; enabling the assessment of empirical results in order to choose the best-suited algorithm for the problem.

This thesis was represented based on literature review and computer simulations. This study answered to main question (which statistical model has the best performance for programmatic TV bid clearance predictions?) and sub-question1 (How each individual model would perform based on same training dataset?) in chapter 3. Sub-questions2 (What approaches each model use in order to calculate the result?) and sub-question3 (What preprocessing algorithm can be used to normalize the data?) has been answered in chapter two.

4.2 Major Results

The experimental results indicate that Neural Networks has shown better performance overall in compare to other two methods. Comparison of the performance of Neural Networks' different optimization methods has shown that the conjugate gradient backpropagation with Polak-Ribière updates algorithm and structure of 9 hidden layer size

has shown the best results. This algorithm has predicted the value of 0,3027 for mean of ones(winners) and the value of 0,0382 for mean of zeros(losers).

Of course, some other algorithms have also shown reliable results for predicting the winners, but since the number of losers in this dataset is much higher than the number of winners (almost 18 times more), performing better for the predicting losers makes the algorithm more valuable for us.

4.3 Future Work

The work described here can explore more areas by applying other data preprocessing algorithms than one hot encoding and normalization. Another direction for future research can be additional examination of the structure of the method. This thesis is working on a two-year dataset of WideOrbit company, a further study including more historical data plus additional features could be also of value.

5 REFERENCES

1. Smallbusiness.chron.com. (2017). What Is Linear Marketing. [online] Available at: <http://smallbusiness.chron.com/linear-marketing-22663.html> [Accessed 26 May 2017].
2. WideOrbit Inc. (2017). WideOrbit: The World's Leading Premium Ad Management Platform. [online] Available at: <http://www.WideOrbit.com> [Accessed 27 May 2017].
3. Inc., W. (2017). WideOrbit Inc.: Private Company Information - Bloomberg. [online] Bloomberg.com. Available at: <https://www.bloomberg.com/research/stocks/private/snapshot.asp?privcapId=3110439> [Accessed 25 May 2017].
4. Stateofdigital.com. (2017). [online] Available at: <http://www.stateofdigital.com/what-is-programmatic-marketing-buying-and-advertising/> [Accessed 26 May 2017].
5. Staff, A. and Staff, A. (2017). Programmatic for Dummies. [online] Adweek.com. Available at: <http://www.adweek.com/brand-marketing/programmatic-dummies-153590/> [Accessed 27 May 2017].
6. Shor, M. (2017). Blind Auction - Game Theory .net. [online] Gametheory.net. Available at: <http://www.gametheory.net/dictionary/auctions/BlindAuction.html> [Accessed 27 May 2017].
7. Msdn.microsoft.com. (2017). Normalize Data. [online] Available at: <https://msdn.microsoft.com/en-us/library/azure/dn905838.aspx> [Accessed 8 May 2017].
8. Anon, (2017). [online] Available at: <https://www.quora.com/What-is-one-hot-encoding-and-when-is-it-used-in-data-science> [Accessed 4 May 2017].
9. Li, J., Tao, D. and Li, X. (2012). A probabilistic model for image representation via multiple patterns. *Pattern Recognition*, 45(11), pp.4044-4053.
10. Sebastian Raschka's Website. (2017). Principal Component Analysis. [online] Available at: http://sebastianraschka.com/Articles/2015_pca_in_3_steps.html [Accessed 19 May 2017].

11. George Dallas. (2017). Principal Component Analysis 4 Dummies: Eigenvectors, Eigenvalues and Dimension Reduction. [online] Available at: <https://georgemdallas.wordpress.com/2013/10/30/principal-component-analysis-4-dummies-eigenvectors-eigenvalues-and-dimension-reduction/> [Accessed 20 May 2017].
12. McCullagh, P. and Nelder, J. (2000). Generalized linear models. 2nd ed. London, [etc]: Chapman and Hall/CRC.
13. Wood, S. (2006). Generalized additive models. Boca Raton: Chapman & Hall/CRC, pp.59-140.
14. Fox, J. (2016). Applied regression analysis and generalized linear models. Los Angeles [u.a.]: Sage, pp.379-424.
15. Touloumi, G., Pantazis, N. and Yiannoutsos, C. (2011). GENERALIZED LINEAR MODELS.
16. Suzuki, Y. (2017). Neural Networks by using Self-Reinforcement Reactions. Japan: The 2017 International Conference on Artificial Life and Robotics (ICAROB 2017), pp.595-598.
17. TOSHI STATS SDN.BHD. (2017). 101-4: Logistic regression. [online] Available at: <http://www.toshistats.net/101-4-logistic-regression/> [Accessed 24 May 2017].
18. H. Yu, B. Wilamowski (2012). Neural network training with second order algorithms, in: Human–Computer Systems Interaction: Backgrounds and Applications, Springer, Chap. 2, 463–476.
19. Basheer, I. and Hajmeer, M. (2000). Artificial neural networks: fundamentals, computing, design, and application. Journal of Microbiological Methods, 43, pp.3-31.
20. Ozgur Kisi; Erdal Uncuoghlu (2005), “Comparison of three backpropagation training algorithms for two case studies,” Indian Journal of Engineering & Materials Sciences, Vol. 12, October 2005, page 434-442.
21. Se.mathworks.com. (2017). Train and Apply Multilayer Neural Networks - MATLAB & Simulink - MathWorks United Kingdom. [online] Available at: <https://se.mathworks.com/help/nnet/ug/train-and-apply-multilayer-neural-networks.html> [Accessed 2 May 2017].

22. Falas, T. and Stafilopatis, A. (1999). The Impact of the Error Function Selection in Neural Network-based Classifiers. IEEE, pp.1799-1804.
23. Sadowski, P. Notes on Backpropagation. [ebook] Irvine: University of California Irvine, pp.1-4. Available at:
<https://www.ics.uci.edu/~pjsadows/notes.pdf> [Accessed 1 Jun. 2017].
24. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In Proceedings of the IEEE Int. ConI on Neural Networks (ICNN), pages 586-591, San Francisco, 1993.
25. Slideshare.net. (2017). H2O Deep Learning at Next.ML. [online] Available at:
<https://www.slideshare.net/0xdata/h2o-deeplearning-nextml> [Accessed 9 Jun. 2017].
26. Se.mathworks.com. (2017). Conjugate gradient backpropagation with Powell-Beale restarts - MATLAB traincgb - MathWorks United Kingdom. [online] Available at: <https://se.mathworks.com/help/nnet/ref/traincgb.html> [Accessed 15 Jun. 2017].
27. Anon, (2017). [online] Available at: 44.
<https://se.mathworks.com/help/optim/ug/unconstrained-nonlinear-optimization-algorithms.html> [Accessed 11 Jun. 2017]. Se.mathworks.com. (2017). One-step secant backpropagation - MATLAB trainoss - MathWorks United Kingdom. [online] Available at: <https://se.mathworks.com/help/nnet/ref/trainoss.html> [Accessed 12 Jun. 2017].
28. regression, W. (2017). What is the difference between linear regression and logistic regression. [online] Stats.stackexchange.com. Available at: <https://stats.stackexchange.com/questions/29325/what-is-the-difference-between-linear-regression-and-logistic-regression> [Accessed 12 Jun. 2017].
29. Shaharin, R., Prodhan, U. and Rahman, M. (2017). Performance Study of TDNN Training Algorithm for Speech Recognition. International Journal of Advanced Research in Computer Science & Technology (IJARCST 2014), 2(4), pp.90-95. Se.mathworks.com. (2017). 1-D minimization using Charalambous' method - MATLAB srchcha - MathWorks United Kingdom. [online] Available at: <https://se.mathworks.com/help/nnet/ref/srchcha.html> [Accessed 10 Jun. 2017].

30. MICKELIN, J. (2013). Named Entity Recognition with Support Vector Machines. M.S. Royal Institute of Technology School of Computer Science and Communication.
31. M. R2016b, “Support vector machines (understanding support vector machines),” 2017. MATLAB official documentation, Available at: <http://se.mathworks.com/help/stats/support-vector-machines-svm.html> [Accessed june. 9, 2017].
32. Pan, X., Lee, B. and Zhang, C. (2013). A comparison of neural network backpropagation algorithms for electricity load forecasting. IEEE International Workshop on Intelligent Energy Systems (IWIES), pp.22-27.
33. Timm, I. and Lorig, F. (2015). A SURVEY ON METHODOLOGICAL ASPECTS OF COMPUTER SIMULATION AS RESEARCH TECHNIQUE. IEEE Xplore, pp.2704-2715.
34. WANG, L., JIANG, T., LI, X. and SUN, F. (2010). A HASS profile research method based on computer simulation technology. 2010 IEEE 17Th International Conference on Industrial Engineering and Engineering Management, pp.911-915.
35. Rojas, R. (1996). Neural Networks - A Systematic Introduction. Berlin: Springer-Verlag, pp.151-184.
36. Se.mathworks.com. (2017). Bayesian regularization backpropagation - MATLAB trainbr - MathWorks United Kingdom. [online] Available at: <https://se.mathworks.com/help/nnet/ref/trainbr.html> [Accessed 28 May 2017].
37. Se.mathworks.com. (2017). Conjugate gradient backpropagation with Polak-Ribièreupdates - MATLAB traincgp - MathWorks United Kingdom. [online] Available at: <https://se.mathworks.com/help/nnet/ref/traincgp.html> [Accessed 30 May 2017].

APPENDIX 1. MATLAB code for Neural Networks

```
clc;
clear;
input=csvread('C:\Users\admin\Documents\MATLAB\data.csv',1);
str=sprintf('%s','data read complete');
fprintf('%s\n', str)
%=====
responce=input(:,size(input,2));
uncontVal=input(:,[2:12,18,26]);
mapped=mapper(uncontVal);
semifinal=onhot(mapped);
stddevinput=preprocesscontinuous(input(:,[13:17,21]));
str=sprintf('%s','ONE HOT ENCODING complete');
fprintf('%s\n', str)
%=====
daysrange =40;
%=====
final=[semifinal,stddevinput,responce];
wholeData=[semifinal,stddevinput];
str=sprintf('%s','data merge complete');
fprintf('%s\n', str)
%=====
timestamps =double(input(:,1)) * 1e-7/86400 + 367;
timestampsarray= datestr(timestamps,'yyyy-mm-dd HH:MM:SS.FFF');
%=====
datestamparray=datenum(timestampsarray(:,1:10));
alldays=unique(datestamparray);
days=str2num(timestampsarray(:,[1 2 3 4 6 7 9 10]));
%=====
```

APPENDIX 1. (continues)

```
for i=1:max(size(alldays))-daysrange
```

```
    datacell(i,:)=dateCal(datestamparray,alldays(daysrange+i-  
    1),alldays(i),alldays(daysrange+i));
```

```
end
```

```
str=sprintf('%s','datecell partitioning complete');
```

```
fprintf('%s\n', str)
```

```
%=====
```

```
k1=1;
```

```
k0=1;
```

```
calculatedresponces=[];
```

```
figure;
```

```
hold on;
```

```
grid on;
```

```
h_old=plot(0,0);
```

```
means=[];
```

```
responce1=[];
```

```
meansNeg=[];
```

```
j=0;
```

```
l=0;
```

```
%=====
```

```
for i=1:200
```

```
    valuetest=unique(responce(datacell(i,3):datacell(i,4)));
```

```
    responce1=[responce1;responce(datacell(i,3):datacell(i,4),:)];
```

```
    if size(valuetest,1)==2
```

```
        network =
```

```
        neural(wholeData(datacell(i,1):datacell(i,2),:),responce(datacell(i,1):datacell(i,  
,2),:));
```

APPENDIX 1. (continues)

`% Test the Network`

```
y = network(wholeData(datacell(i,3):datacell(i,4),:));
y=y';
calculatedresponses=[calculatedresponses;y];%#ok    onevalues =
find(responce(datacell(i,3):datacell(i,4)));
zerovalues=find(responce(datacell(i,3):datacell(i,4))==0);

oneserrors(k1) =
computeerror(y,responce(datacell(i,3):datacell(i,4)),onevalues);%#ok
zeroserrors(k0) =
computeerror(y,responce(datacell(i,3):datacell(i,4)),zerovalues);%#ok

k1=k1+1;
k0=k0+1;

means(i-j) = mean(y(onevalues));
meansNeg(i-1) = mean(y(zerovalues));
h=plot(means);
delete(h_old);
h_old=h;
grid on;
drawnow;
mean(y(onevalues))
```

`elseif valuetest==0`

```
network=neural(wholeData(datacell(i,1):datacell(i,2),:),responce(datacell(i,1)
:datacell(i,2),:));
```

APPENDIX 1. (continues)

% Test the Network

```
y=network(wholeData(datacell(i,3):datacell(i,4),:));
y=y';
calculatedresponces=[calculatedresponces;y];%#ok
zerovalues=find(responce(datacell(i,3):datacell(i,4))==0);
zeroserrors(k0)=computeerror(y,responce(datacell(i,3):datacell(i,4)),zerovalu
es);%#ok
k0=k0+1;
j=j+1;
meansNeg(i-1) = mean(y(zerovalues));
```

elseif valuetest==1

```
network=neural(wholeData(datacell(i,1):datacell(i,2),:),
responce(datacell(i,1):datacell(i,2),:));
```

% Test the Network

```
y=network(wholeData(datacell(i,3):datacell(i,4),:));
y=y';
calculatedresponces=[calculatedresponces;y];%#ok
onevalues=find(responce(datacell(i,3):datacell(i,4)));
oneserrors(k1)=computeerror(y,responce(datacell(i,3):datacell(i,4)),onevalue
s);%#ok

k1=k1+1;
l=l+1;
means(i-j) = mean(y(onevalues))
h=plot(means);
```

APPENDIX 1. (continues)

```
        delete(h_old);
        h_old=h;
        grid on;
        drawnow;
        mean(y(onevalues))

    end
    i
end
%=====
responce=responce(datacell(1,3): size(responce,1));
timestamparray=timestamparray(datacell(1,3): size(timestamparray,1),:);
%=====
monthes=str2num(timestamparray(:,[1 2 3 4 6 7]));
umonthes=unique(monthes);
%=====
for k=1:size(umonthes,1)
    monthindex(k,1)=find(monthes==umonthes(k),1,'first');
    monthindex(k,2)=find(monthes==umonthes(k),1,'last');
end
%=====
m=0;
for k=1: size(monthindex,1)
    if monthindex(k,1)<=size(calculatedresponces,1) &&
monthindex(k,2)>=size(calculatedresponces,1)
        m=k;
    end
end
%=====
idx_ones=find(responce1);
```

APPENDIX 1. (continues)

```
idx_zeros=find(responce1==0);
mean_ones=mean(calculatedresponces(idx_ones));
mean_zeros=mean(calculatedresponces(idx_zeros));
%=====

for n=1:m

    idx_month_ones=idx_ones(idx_ones>=monthindex(n,1) &
    idx_ones<=monthindex(n,2));
    idx_month_zeros=idx_zeros(idx_zeros>=monthindex(n,1) &
    idx_zeros<=monthindex(n,2));
    monthdataforone=calculatedresponces(idx_month_ones);
    monthmseforone(n)=sum(monthdataforone)/size(monthdataforone,1);
    monthdataforzero=calculatedresponces( idx_month_zeros);
    monthmseforzero(n)=sum(monthdataforzero)/size(monthdataforzero,1);

end

%=====
% training function trainbr
%=====

function [output]=neural(x,t)
    x =x';
    t = t';
    trainFcn = 'trainbr';
    hiddenLayerSize = 9;
    net = patternnet(hiddenLayerSize, trainFcn);
    net.divideParam.trainRatio = 100/100;
    net.divideParam.valRatio = 0/100;
    net.divideParam.testRatio = 0/100;
    net.trainParam.epochs=150;
```

APPENDIX 1. (continues)

```
[net,tr] = train(net,x,t,'useParallel','yes','showResources','yes');  
output= net;
```

```
%=====
```

```
% training function trainscg
```

```
%=====
```

```
function [output]=neural(x,t)  
    x =x';  
    t = t';  
    trainFcn = 'trainbr';  
    hiddenLayerSize = 9;  
    net = patternnet(hiddenLayerSize, trainFcn);  
    net.divideParam.trainRatio = 100/100;  
    net.divideParam.valRatio = 0/100;  
    net.divideParam.testRatio = 0/100;  
    net.trainParam.epochs=150;
```

```
[net,tr] = train(net,x,t,'useParallel','yes','showResources','yes');  
output= net;
```

```
%=====
```

```
% training function trainlm
```

```
%=====
```

```
function [output]=neural(x,t)  
    x =x';  
    t = t';  
    trainFcn = 'trainlm';  
    hiddenLayerSize = 9;  
    net = patternnet(hiddenLayerSize, trainFcn);
```

APPENDIX 1. (continues)

```
net.divideParam.trainRatio = 100/100;
```

```
net.divideParam.valRatio = 0/100;
```

```
net.divideParam.testRatio = 0/100;
```

```
net.trainParam.epochs=150;
```

```
[net,tr] = train(net,x,t,'useParallel','yes','showResources','yes');
```

```
output= net;
```

```
%=====
% training function trainbfg
%=====
```

```
function [output]=neural(x,t)
```

```
    x =x';
```

```
    t = t';
```

```
    trainFcn = 'trainbfg';
```

```
    hiddenLayerSize = 9;
```

```
    net = patternnet(hiddenLayerSize, trainFcn);
```

```
    net.divideParam.trainRatio = 100/100;
```

```
    net.divideParam.valRatio = 0/100;
```

```
    net.divideParam.testRatio = 0/100;
```

```
    net.trainParam.epochs=150;
```

```
[net,tr] = train(net,x,t,'useParallel','yes','showResources','yes');
```

```
output= net;
```

```
%=====
% training function traingd
%=====
```

```
function [output]=neural(x,t)
```

APPENDIX 1. (continues)

```
x =x';
t = t';
trainFcn = 'traingd';
hiddenLayerSize = 9;
net = patternnet(hiddenLayerSize, trainFcn);
net.divideParam.trainRatio = 100/100;
net.divideParam.valRatio = 0/100;
net.divideParam.testRatio = 0/100;
net.trainParam.epochs=150;

[net,tr] = train(net,x,t,'useParallel','yes','showResources','yes');
output= net;

%=====
% training function traingdm
%=====

function [output]=neural(x,t)
    x =x';
    t = t';
    trainFcn = 'traingdm';
    hiddenLayerSize = 9;
    net = patternnet(hiddenLayerSize, trainFcn);
    net.divideParam.trainRatio = 100/100;
    net.divideParam.valRatio = 0/100;
    net.divideParam.testRatio = 0/100;
    net.trainParam.epochs=150;

    [net,tr] = train(net,x,t,'useParallel','yes','showResources','yes');
    output= net;

%=====
% training function trainrp
```

```
%=====
```

APPENDIX 1. (continues)

```
function [output]=neural(x,t)
    x =x';
    t = t';
    trainFcn = 'trainrp';
    hiddenLayerSize = 9;
    net = patternnet(hiddenLayerSize, trainFcn);
    net.divideParam.trainRatio = 100/100;
    net.divideParam.valRatio = 0/100;
    net.divideParam.testRatio = 0/100;
    net.trainParam.epochs=150;

    [net,tr] = train(net,x,t,'useParallel','yes','showResources','yes');
    output= net;
```

```
%=====
```

```
% training function traincgb
```

```
%=====
```

```
function [output]=neural(x,t)
    x =x';
    t = t';
    trainFcn = 'traincgb';
    hiddenLayerSize = 9;
    net = patternnet(hiddenLayerSize, trainFcn);
    net.divideParam.trainRatio = 100/100;
    net.divideParam.valRatio = 0/100;
    net.divideParam.testRatio = 0/100;
    net.trainParam.epochs=150;

    [net,tr] = train(net,x,t,'useParallel','yes','showResources','yes');
    output= net;
```

APPENDIX 1. (continues)

```
%=====
% training function traincbr
%=====
function [output]=neural(x,t)
    x =x';
    t = t';
    trainFcn = 'trainbr';
    hiddenLayerSize = 9;
    net = patternnet(hiddenLayerSize, trainFcn);
    net.divideParam.trainRatio = 100/100;
    net.divideParam.valRatio = 0/100;
    net.divideParam.testRatio = 0/100;
    net.trainParam.epochs=150;
    [net,tr] = train(net,x,t,'useParallel','yes','showResources','yes');
    output= net;
```

```
%=====
% training function trainoss
%=====
function [output]=neural(x,t)
    x =x';
    t = t';
    trainFcn = 'trainoss';
    hiddenLayerSize = 9;
    net = patternnet(hiddenLayerSize, trainFcn);
    net.divideParam.trainRatio = 100/100;
    net.divideParam.valRatio = 0/100;
    net.divideParam.testRatio = 0/100;
    net.trainParam.epochs=150;
```

APPENDIX 1. (continues)

```
[net,tr] = train(net,x,t,'useParallel','yes','showResources','yes');
output= net;
%=====
% training function traincgf
%=====
function [output]=neural(x,t)
    x =x';
    t = t';
    trainFcn = 'traincgf';
    hiddenLayerSize = 9;
    net = patternnet(hiddenLayerSize, trainFcn);
    net.divideParam.trainRatio = 100/100;
    net.divideParam.valRatio = 0/100;
    net.divideParam.testRatio = 0/100;
    net.trainParam.epochs=150;

    [net,tr] = train(net,x,t,'useParallel','yes','showResources','yes');
    output= net;

%=====
% training function traincgp
%=====
function [output]=neural(x,t)
    x =x';
    t = t';
    trainFcn = 'traincgp';
    hiddenLayerSize = 9;
    net = patternnet(hiddenLayerSize, trainFcn);
    net.divideParam.trainRatio = 100/100;
```

APPENDIX 1. (continues)

```
net.divideParam.valRatio = 0/100;
    net.divideParam.testRatio = 0/100;
    net.trainParam.epochs=150;

    [net,tr] = train(net,x,t,'useParallel','yes','showResources','yes');
    output= net;

%=====
% training function traingdx
%=====

function [output]=neural(x,t)
    x =x';
    t = t';
    trainFcn = 'traingdx';
    hiddenLayerSize = 9;
    net = patternnet(hiddenLayerSize, trainFcn);
    net.divideParam.trainRatio = 100/100;
    net.divideParam.valRatio = 0/100;
    net.divideParam.testRatio = 0/100;
    net.trainParam.epochs=150;

    [net,tr] = train(net,x,t,'useParallel','yes','showResources','yes');
    output= net;
```

APPENDIX 2. MATLAB code for Generalized Linear Models

```
clc;
clear;
input=csvread('/Users/apple/Documents/MATLAB/data.csv',1);
str=sprintf('%s','data read complete');
fprintf('%s\n', str)
%=====
response=input(:,size(input,2));
uncontVal=input(:,[2:12,18,26]);
mapped=mapper(uncontVal);
semifinal=onhot(mapped);
stddevinput=preprocesscontinuous(input(:,[13:17,21]));
str=sprintf('%s','ONE HOT ENCODING complete');
fprintf('%s\n', str)
%=====
daysrange =40;
%=====
final=[semifinal,stddevinput,response];
wholeData=[semifinal,stddevinput];
str=sprintf('%s','data merge complete');
fprintf('%s\n', str)
%=====
timestamps =double(input(:,1)) * 1e-7/86400 + 367;
timestampsarray= datestr(timestamps,'yyyy-mm-dd HH:MM:SS.FFF');
%=====
datestamparray=datenum(timestampsarray(:,1:10));
alldays=unique(datestamparray);
days=str2num(timestampsarray(:,[1 2 3 4 6 7 9 10]));
%=====

for i=1:max(size(alldays))-daysrange
```

APPENDIX 2. (continues)

```
    datacell(i,:)=dateCal(datestamparray,alldays(daysrange+i-
1),alldays(i),alldays(daysrange+i));
end
str=sprintf('%s','datecell partitioning complete');
fprintf('%s\n', str)
%=====
k1=1;
k0=1;
calculatedresponces=[];
figure;
hold on;
grid on;
h_old=plot(0,0);
means=[];
responce1=[];
meansNeg=[];
j=0;
l=0;

%=====
for i=1:200

    valuetest=unique(responce(datacell(i,3):datacell(i,4)));
    responce1=[responce1;responce(datacell(i,3):datacell(i,4),:)];

    if size(valuetest,1)==2

        [b, dev, stats] = glmfit(wholeData(datacell(i,1):datacell(i,2),:),
responce(datacell(i,1):datacell(i,2),:), 'binomial', 'link', 'logit');
```

APPENDIX 2. (continues)

```
[yfit, yhi, ylo] = glmval(b,wholeData(datacell(i,3):datacell(i,4),:),'logit',  
stats);
```

```
calculatedresponces=[calculatedresponces;yfit];
```

```
onevalues=find(responce(datacell(i,3):datacell(i,4)));
```

```
zerovalues=find(responce(datacell(i,3):datacell(i,4))==0);
```

```
oneserrors(k1) =
```

```
computeerror(yfit,responce(datacell(i,3):datacell(i,4)),onevalues);%#ok
```

```
zeroserrors(k0)=computeerror(yfit,responce(datacell(i,3):datacell(i,4)),zerova  
lues);%#ok
```

```
k1=k1+1;
```

```
k0=k0+1;
```

```
means(i-j) = mean(yfit(onevalues));
```

```
meansNeg(i-1) = mean(yfit(zerovalues));
```

```
h=plot(means);
```

```
delete(h_old);
```

```
h_old=h;
```

```
grid on;
```

```
drawnow;
```

```
mean(yfit(onevalues))
```

```
elseif valuetest==0
```

```
[b, dev, stats] = glmfit(wholeData(datacell(i,1):datacell(i,2),:),
```

```
responce(datacell(i,1):datacell(i,2),:),'binomial', 'link', 'logit');
```

```
[yfit, yhi, ylo] = glmval(b,wholeData(datacell(i,3):datacell(i,4),:),'logit',  
stats);
```

APPENDIX 2. (continues)

```
calculatedresponces=[calculatedresponces;yfit];%#ok
zerovalues=find(responce(datacell(i,3):datacell(i,4))==0);

zeroserrors(k0)=computeerror(yfit,responce(datacell(i,3):datacell(i,4)),zerova
lues);%#ok
k0=k0+1;
j=j+1;
meansNeg(i-1) = mean(yfit(zerovalues));
```

```
elseif valuetest==1
```

```
[b, dev, stats] = glmfit(wholeData(datacell(i,1):datacell(i,2),:),
responce(datacell(i,1):datacell(i,2),:), 'binomial', 'link', 'logit');
[yfit, yhi, ylo] = glmval(b,wholeData(datacell(i,3):datacell(i,4),:),'logit',
stats);
```

```
calculatedresponces=[calculatedresponces;yfit];%#ok
onevalues=find(responce(datacell(i,3):datacell(i,4)));

oneserrors(k1) =
computeerror(yfit,responce(datacell(i,3):datacell(i,4)),onevalues);%#ok
k1=k1+1;
onevalues=find(responce(datacell(i,3):datacell(i,4)));
oneserrors(k1) =
computeerror(yfit,responce(datacell(i,3):datacell(i,4)),onevalues);%#ok
k1=k1+1;

means(i-j) = mean(yfit(onevalues))

h=plot(means);
```

APPENDIX 2. (continues)

```
delete(h_old);
    h_old=h;
    grid on;
    drawnow;
    mean(yfit(onevalues))
    l=l+1;

end
i
end
%=====
response=response(datacell(1,3):size(response,1));
timestamparray=timestamparray(datacell(1,3):size(timestamparray,1),:);
%=====
monthes=str2num(timestamparray(:,[1 2 3 4 6 7]));
umonthes=unique(monthes);
%=====

for k=1:size(umonthes,1)
    monthindex(k,1)=find(monthes==umonthes(k),1,'first');
    monthindex(k,2)=find(monthes==umonthes(k),1,'last');
end
%=====
m=0;
for k=1:size(monthindex,1)
    if monthindex(k,1)<=size(calculatedresponces,1) &&
        monthindex(k,2)>=size(calculatedresponces,1)
        m=k;
    end
end
%=====
```

APPENDIX 2. (continues)

```
idx_ones=find(responce1);
idx_zeros=find(responce1==0);
mean_ones=mean(calculatedresponces(idx_ones));
mean_zeros=mean(calculatedresponces(idx_zeros));

for n=1:m

    idx_month_ones=idx_ones(idx_ones>=monthindex(n,1) &
    idx_ones<=monthindex(n,2));
    idx_month_zeros=idx_zeros(idx_zeros>=monthindex(n,1) &
    idx_zeros<=monthindex(n,2));
    monthdataforone=calculatedresponces(idx_month_ones);
    monthmseforone(n)=sum(monthdataforone)/size(monthdataforone,1);
    monthdataforzero=calculatedresponces( idx_month_zeros);
    monthmseforzero(n)=sum(monthdataforzero)/size(monthdataforzero,1);

end
```

APPENDIX 3. MATLAB code for Support Vector Machines

```
clc;
clear;
input=csvread('/Users/apple/Documents/MATLAB/data.csv',1);
str=sprintf('%s','data read complete');
fprintf('%s\n', str)
%=====
responce=input(:,size(input,2));
compresponce=input(:,size(input,2));
uncontVal=input(:,[2:12,18,26]);
mapped=mapper(uncontVal);
semifinal=onhot(mapped);
stddevinput=preprocesscontinuous(input(:,[13:17,21]));
str=sprintf('%s','ONE HOT ENCODING complete');
fprintf('%s\n', str)
%=====
daysrange = 40;
%=====
final=[semifinal,stddevinput,responce];
wholeData=[semifinal,stddevinput];
str=sprintf('%s','data merge complete');
fprintf('%s\n', str)
%=====

timestamps =double(input(:,1)) * 1e-7/86400 + 367;
timestampsarray= datestr(timestamps,'yyyy-mm-dd HH:MM:SS.FFF');
str=sprintf('%s','time stamps conversion complete');
fprintf('%s\n', str)
%=====

datestampsarray=datenum(timestampsarray(:,1:10));
```

APPENDIX 3. (continues)

```
alldays=unique(timestampsarray);
days=str2num(timestampsarray(:,[1 2 3 4 6 7 9 10]));
str=sprintf('%s','time stamps to days conversion complete');
fprintf('%s\n', str)
%=====
for i=1:max(size(alldays))-daysrange

        datacell(i,:)=dateCal(timestampsarray,alldays(daysrange+i-
        1),alldays(i),alldays(daysrange+i));

end
%=====
str=sprintf('%s','datecell partitioning complete');
fprintf('%s\n', str)
%=====
k1=1;
k0=1;
figure;
hold on;
grid on;
h_old=plot(0,0);
%=====
means=[];
responce1=[];
meansNeg=[];
j=0;
l=0;
calculatedresponces=[];
str=sprintf('%s','Starting model fitting');
fprintf('%s\n', str)
%=====
```

APPENDIX 3. (continues)

```
for i=1:20
```

```
    valuetest=unique(response(datacell(i,3):datacell(i,4)));  
    response1=[response1;response(datacell(i,3):datacell(i,4),:)];
```

```
if size(valuetest,1)==2
```

```
    SVMModel =  
    fitsvm(wholeData(datacell(i,1):datacell(i,2),:),response(datacell(i,1):datacell  
(i,2),:),'KernelFunction','RBF', 'KernelScale','auto');  
  
    SVMModelPosteriorProb = fitSVMPosterior(SVMModel);  
    [label,probability] =  
    predict(SVMModelPosteriorProb,wholeData(datacell(i,3):datacell(i,4),:));  
  
    bp=probability(:,2);  
    calculatedresponses=[calculatedresponses;bp];  
    onevalues=find(response(datacell(i,3):datacell(i,4)));  
    zerovalues=find(response(datacell(i,3):datacell(i,4))==0);  
    oneserrors(k1) =  
    computeerror(bp,response(datacell(i,3):datacell(i,4)),onevalues);%#ok  
    zeroserrors(k0)=computeerror(bp,response(datacell(i,3):datacell(i,4)),zeroval  
ues);%#ok  
    k1=k1+1;  
    k0=k0+1;  
    means(i-j) = mean(bp(onevalues));  
    meansNeg(i-1) = mean(bp(zerovalues));  
    h=plot(means);  
    delete(h_old);  
    h_old=h;
```

APPENDIX 3. (continues)

grid on;

```
drawnow;  
mean(bp(onevalues))
```

elseif valuetest==0

```
SVMMModel =  
fitsvm(wholeData(datacell(i,1):datacell(i,2),:),response(datacell(i,1):datacell  
(i,2),:),'KernelFunction','RBF', 'KernelScale','auto');
```

```
SVMMModelPosteriorProb = fitSVMPosterior(SVMMModel);  
[label,probability] =  
predict(SVMMModelPosteriorProb,wholeData(datacell(i,3):datacell(i,4),:));  
bp=probability(:,2);  
calculatedresponses=[calculatedresponses;bp];%#ok  
zerovalues=find(response(datacell(i,3):datacell(i,4))==0);
```

```
zeroserrors(k0)=computeerror(bp,response(datacell(i,3):datacell(i,4)),zeroval  
ues);%#ok  
k0=k0+1;  
j=j+1;  
meansNeg(i-1) = mean(bp(zerovalues));
```

elseif valuetest==1

```
SVMMModel =  
fitsvm(wholeData(datacell(i,1):datacell(i,2),:),response(datacell(i,1):datacell  
(i,2),:),'KernelFunction','RBF', 'KernelScale','auto');
```

```
SVMMModelPosteriorProb = fitSVMPosterior(SVMMModel);
```

APPENDIX 3. (continues)

```
[label,probability] =
predict(SVMModelPosteriorProb,wholeData(datacell(i,3):datacell(i,4),:));
bp=probability(:,2);

calculatedresponces=[calculatedresponces;bp];%#ok
onevalues=find(responce(datacell(i,3):datacell(i,4)));
oneserrors(k1) =
computeerror(bp,responce(datacell(i,3):datacell(i,4)),onevalues);%#ok

k1=k1+1;

means(i-j) = mean(bp(onevalues))
h=plot(means);
delete(h_old);
h_old=h;
grid on;
drawnow;
mean(bp(onevalues))
l=l+1;

end
i
end

%=====
responce=responce(datacell(1,3):size(responce,1));
timestampsarray=timestampsarray(datacell(1,3):size(timestampsarray,1),:);
%=====
%Dividing the data
%=====
monthes=str2num(timestampsarray(:,[1 2 3 4 6 7]));
umonthes=unique(monthes);
```

APPENDIX 3. (continues)

```
%=====
for k=1:size(umonthes,1)

    monthindex(k,1)=find(monthes==umonthes(k),1,'first');
    monthindex(k,2)=find(monthes==umonthes(k),1,'last');

end

%=====
m=0;

for k=1:size(monthindex,1)
    if monthindex(k,1)<=size(calculatedresponces,1)
        &&monthindex(k,2)>=size(calculatedresponces,1)

        m=k;
    end
end

idx_ones=find(responce1);
idx_zeros=find(responce1==0);

mean_ones=mean(calculatedresponces(idx_ones));
mean_zeros=mean(calculatedresponces(idx_zeros));

for n=1:m

    idx_month_ones=idx_ones(idx_ones>=monthindex(n,1) &
    idx_ones<=monthindex(n,2));
```

APPENDIX 3. (continues)

```
idx_month_zeros=idx_zeros(idx_zeros>=monthindex(n,1) &  
idx_zeros<=monthindex(n,2));  
monthdataforone=calculatedresponces(idx_month_ones);  
monthmseforone(n)=sum(monthdataforone)/size(monthdataforone,1);  
monthdataforzero=calculatedresponces( idx_month_zeros);  
monthmseforzero(n)=sum(monthdataforzero)/size(monthdataforzero,1);
```

end

APPENDIX 4. MATLAB code for Preprocessing

```
%=====
%preprocessing function
%=====

function [output]=preprocesscontinuous(input)

contVal=input;
meanvalue= mean(contVal);
stnddev= std(contVal);
    for j=1 : size(contVal,2)
        for i=1 : size(contVal,1)
            contVal(i,j) = (contVal(i,j)-meanvalue(j))/stnddev(j);
        end
    end
output=contVal;

%=====
% The counterpart conversion from System.
% DateTime to MATLAB serial date number:
%=====

function MATLABserialtime = dt2mt(datetimeticks)

    MATLABserialtime = double(datetimeticks) * 1e-7/86400 + 367;
end

datestr(times,'yyyy-mm-dd HH:MM:SS.FFF')
%=====
% calculating days range of data
%=====
```

APPENDIX 4. (continues)

```
function [output]=dateCal(alldates,lastdate,firstdate,day)

    daysrange=[find(alldates==firstdate,1,'first'),find(alldates==lastdate,1,'last')];
    dayrange=[find(alldates==day,1,'first'),find(alldates==day,1,'last')];

output=[daysrange,dayrange];
    end

%=====
%Error computation
%=====
function [error]=computeerror(yfit,respnce,valuetype)

    error=sum((yfit(valuetype)-respnce(valuetype)).^2)/size(yfit(valuetype),1);

end

%=====
%One hot encoding
%=====
function [onehtoutput]=onhot(input)
onehtoutput=zeros(size(input,1),1);

for i=1:size(input,2)
    onehtoutput=[onehtoutput,input(:,i)==1:max(input(:,i))];%#ok
end
onehtoutput=onehtoutput(:,2:size(onehtoutput,2));

%=====
%mapper function
%=====
function [output]=mapper(input)
for i=1:size(input,2)
```

APPENDIX 4. (continues)

```
uniquedcolumn=unique(input(:,i));
for j=1:size(input,1)
    for k=1:size(uniquedcolumn,1)
        if input(j,i)==uniquedcolumn(k)
            output(j,i)=k; %#ok
        end
    end
end
end
end
```

APPENDIX 5. MATLAB code for PCA

```
clc;
clear;
input=csvread('C:\Users\admin\Documents\MATLAB\data.csv',1);
str=sprintf('%s','data read complete');
fprintf('%s\n', str)
%=====
response=input(:,size(input,2));
uncontVal=input(:,[2:12,18,26]);
mapped=mapper(uncontVal);
semifinal=onhot(mapped);
stddevinput=preprocesscontinuous(input(:,[2:18,21,26]));
str=sprintf('%s','ONE HOT ENCODING complete');
fprintf('%s\n', str)
%=====
daysrange =40;
%=====
wholeData=[semifinal,stddevinput,response];
str=sprintf('%s','data merge complete');
fprintf('%s\n', str)
%=====
timestamps =double(input(:,1)) * 1e-7/86400 + 367;
timestampsarray= datestr(timestamps,'yyyy-mm-dd HH:MM:SS.FFF');
%=====
datestamparray=datenum(timestampsarray(:,1:10));
alldays=unique(datestamparray);
days=str2num(timestampsarray(:,[1 2 3 4 6 7 9 10]));
%=====
for i=1:max(size(alldays))-daysrange
```

APPENDIX 5. (continues)

```
        datacell(i,:)=dateCal(datestamparray,alldays(daysrange+i-
        1),alldays(i),alldays(daysrange+i));

end

%=====
str=sprintf('%s','datecell partitioning complete');
fprintf('%s\n', str)
%=====

k1=1;
k0=1;
figure;
hold on;
grid on;
h_old=plot(0,0);
means=[];
responce1=[];
meansNeg=[];
calculatedresponces=[];
j=0;
l=0;
%=====

[COEFF,SCORE,latent,tsquare] = princomp(wholeData(:,727:747));
v_new=COEFF(1:3,:)
data_new = v_new*wholeData(:,727:747)';
hold on;
a=find(responce==1);
b=find(responce==0);

scatter3(data_new(1,a),data_new(2,a),data_new(3,a),'b*');
scatter3(data_new(1,b),data_new(2,b),data_new(3,b),'r');
```