TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY

Department of Mechatronics

Chair of Mechatronics systems

MHK

Tornike Mzhavia

Vehicle insurance claim data study and forecasting model using Artificial Neural Networks

MSc thesis

The author applies for
the academic degree
Master of Science in Engineering

Tallinn

2016

Author's declaration

I declare that I have written this graduation thesis independently.

These materials have not been submitted for any academic degree.

All the works of other authors used in this thesis have been referenced.

The thesis was completed under _____ supervision

Author

_____ signature

The thesis complies with the requirements for graduation theses.

Supervisor

_____ signature

Accepted for defense.

_____ Chairman of the defense committee

_____ signature

Chair of Mechatronics systems

MSc THESIS TASK SHEET

2016 spring semester

Student	Tornike Mzhavia 145137MAHM
Study program	Mechatronics
Specialty	-
Supervisor	Research Scientist Dmitry Shvarts, Associate Professor Eduard Petlenkov
Consultants	-

THESIS TOPIC:

(In Estonian) Tehisnärvivõrkudel põhinev klientide klassifitseerimise ja riskigruppide määramise süsteem liikluskindlustuse jaoks

(In English) Vehicle insurance claim data study and forecasting model using Artificial Neural Networks

Assignments to be completed and schedule for their completion:

No.	Task description	Deadline
1.	Data study, preprocessing and feature selection	01/03/2016
2.	Model training and tuning	01/04/2016
3.	Model evaluation, testing and deployment	01/05/2016

Engineering and economic problems to be solved:

The problem to be solved is to devise a reasonably successful predictive model for vehicle insurance claim risk assessment based on real life data using Artificial Neural Networks.

Language of the thesis: English

Deadline for submission of the application for defense: 16/05/2015

Deadline for submission of the thesis: 20/05/2016

Student	Tornike Mzhavia	signature:	date: 12/02/2016
Supervisor	Eduard Petlenkov	signature:	date: 12/02/2016
Supervisor	Dmitry Shvarts	signature:	date: 12/02/2016

Confidentiality requirements and other corporate terms and conditions shall be set out on the reverse side.

TABLE OF CONTENTS

FOREWORD 6
1. **INTRODUCTION** 7
2. **JUSTIFICATION OF METHODOLOGY** 9
3. **DATA PREPROCESSING** 10
3.1. Provided dataset characteristics and problem definition 10
3.1.1. Numerical data 11
3.1.2. Categorical data 12
3.1.3. Problem statement 12
3.2. Training data preparation 13
3.2.1. Missing attributes 13
3.2.2. Scaling of the data 14
3.2.3. Categorical data encoding 14
3.3. Dimensionality reduction and feature selection 16
3.4. Data splitting and sampling 18
3.5. Target data 19
4. **NETWORK ARCHITECTURE** 20
4.1. Feed-forward network architecture 21
4.1.1. Input layer 23
4.1.2. Hidden layers 24
4.1.3. Output layer 27
4.2. Deep autoencoder or replicator network architecture 28
4.2.1. Encoder layers 29
4.2.2. Output layer 30
4.3. Initializing the network 31
5. **TRAINING OF THE CLASIFFIER** 32
5.1. Network performance function 33
5.1.1. Mean squared error performance function (MSE) 33
5.1.2. Cross-entropy performance function 34
5.1.3. Regularization parameter for the performance function 35
5.2. Backpropagation algorithm 38
5.2.1. Modes of training 39
5.2.2. Variations on the backpropagation method 40
5.2.3. Validation checks and other stopping criterions 44

5.3. Parallel computing modes for the network training 46
6. **MODEL EVALUATION AND FINE-TUNING** 48
6.1. Imbalanced dataset classifier evaluation metrics 48
6.1.1. Confusion matrix 49
6.1.2. Precision and Recall 50
6.1.3. F score 51
6.1.4. Receiver operating characteristic 52
6.2. Applying evaluation metrics to classifiers 53
6.3. Fine tuning the classifiers and comparing to an existing solution 55
6.3.1. Optimal threshold value based on $\boldsymbol{F0,5}$ score 57
6.3.2. Optimal threshold value based on $\boldsymbol{F1}$ score 59
6.3.3. Optimal threshold value based on $\boldsymbol{F2}$ score 60
6.4 Network deployment 62
CONCLUSION 63
SUMMARY 64
REFERENCES 67
APPENDICES 70
Appendix 1. Receiver Operating Characteristic for Scaled Conjugate Gradient method 70
Appendix 2. Training performance plot for Scaled Conjugate Gradient method 71
Appendix 3. Receiver Operating Characteristic for Bayesian Regularization method 72
Appendix 4. Training performance plot for Bayesian Regularization method 73
Appendix 5. Deployed function code snippet for Autoencoder network 74
Appendix 6. Deployed function code snippet for Feed-forward network 75

FOREWORD

This thesis represents the work done for the project, with the main aim of studying real life vehicle insurance data for finding correlations and causations which lead to the insurance claims made by the customers. The aim of this project was to, if possible, create a predictive model based on Artificial Neural Network technology which would be trained based on the said database in order to successfully forecasting the risks of claims and creating fair ratemaking for the insurance company and their customers.

Data was provided by the Estonian insurance company "ERGO" and was provided to the "Alpha Control Lab" (A-lab) within the department of Computer Control in Tallinn University of Technology. The project was conducted and managed by the said laboratory and provided as a Master Degree project for me under the supervision of Associate Professor Eduard Petlenkov.

The work was done in the laboratory and workstation, provided by the "Alpha Control Lab" within the building of the institution. All the programming and experiments were conducted and executed by using 64 bit version of Matlab R2015b with version number: 8.6.0.267246 and Neural Network Toolbox version number 8.4.

The supervision from the Department of Mechatronics, chair of Mechatronic Systems was provided by the Research Scientist Dmitry Shvarts.

I would like to thank the supervisors and the institution for providing an interesting research topic, the well-equipped laboratory for conducting a successful research and for providing supervision, inspiration and guidance for making this work possible.

1. INTRODUCTION

One of the problems facing modern insurance companies include assessment of the potential customers and risks associated with underwriting and accepting liability on covering the costs in case of damage occurs. In order for the insurance company to ensure they are setting fair and adequate premiums they implement the process which is called ratemaking. [1]

Calculating proper ratemaking is one of the main mathematical problems faced by insurance companies and car insurance is not an exception. It is a complex task during which company should take account for quite a big number of factors and variables. Since creating a realistic and optimal ratemaking model can save money and also make an insurance company more competitive, it's not a surprise that such model would be quite demanded and needed. [2]

The same problem had arisen with a large Estonian insurance company "Ergo". Company is interested in creating the model which would take the problem of ratemaking, meaning that it would accept the potential customer data as an input and output a risk factor associated with signing a contract and underwriting. On the behalf of their big experience in business they are providing a large dataset (containing approximately 500 000 samples) of insurance claims made by their customers.

Additional data or latent variables can be added during the research if they seem to benefit the predictive model and if they will be openly available for an access.

Using this data an appropriate binary classification model will be created, which will distinguish between customers with high and low probability of filling an insurance claim. This would benefit the insurance company and make more realistic predictions depending on customer data. This will result in more fair underwriting prices and increased risk management and control for the insurance company. Expected result is to get the model which would output a satisfactory risk assessment and would show it's correctness through time by optimizing insurance pricing and company expenditures.

Studying this data can also have a beneficial effect on understanding the risk variables involved in a car accidents, which could be used as a knowledge for minimizing lethal cases, injuries and financial expenditures during accidents on the road.

The current model used in most of the insurance companies have been unchanged for a long time and still use the analytical models, which means they are not taking the advantage of the big data they possess in order to improve their predictive abilities.

The insurance company underwriters gather the information of their customers in order to determine whether or not the risk should be taken by the company. They study past insurance losses and using traditional statistical methods loss-contributing characteristics are looked for. When the positive relationship is found between the policy characteristics and future losses, the guidelines are created for the underwriters to follow. This methods can be criticized for their precision. This is caused mainly by underwriters not using all the available information and data accumulated by the time period of automobile insurance, which can be studied and used for improving the model. With the use of modern adaptive and learning methods this information can be used to make the underwriting decisions more accurate. [3]

For a long time computers and their computation power were not considered important in the underwriting process and leading experts believed that the human judgment factor in the decision process was too complex to model and handle. [4]

In later years this statement have been proven wrong and that underwriting judgment could be modeled and implemented on a computer. This points can be observed in number of works and been shown possible by many researchers. [3][4][5][6][7]

Conducting a market research was unrealistic since this information is mostly confidential for each of the insurance companies and it is hard to find clear information about the intelligent analytical methods used within the insurance field. Although for the background research several promising theoretical works have been studied which had shown insight in the field of applying intelligent models for studying insurance ratemaking problems. Interesting examples were provided in the works [2] and [3] where several techniques where compared for making a predictive models. This works have studied the statistical distribution characteristics and have provided theoretical insight into the problem. Methods like decision trees, linear regression, support vector machine, ordinary neural networks and neural networks with Softmax layer were compared for modeling the data. Their performance was compared with each other and also with existing classical models. As a conclusion they showed that neural networks when applied to automobile insurance ratemaking allowed to identify with increased precision the risks associated with each customer.

In the following chapters I will discuss: data preprocessing steps which were necessary for creating a successful predictive classifier also network architectures which can provide the best results and their comparison, then network training methods comparison study based on their computational requirements and capabilities. Next I will evaluate each approach by using best practices of assessing binary classifiers and conclude by choosing the most promising of them.

2. JUSTIFICATION OF METHODOLOGY

The Artificial Neural Networks have become a standard in the field of machine learning and data mining and this has its own reasons. One of the reasons is that big data have been more available as the databases have been standardized and used in large scales and the second reason is the ability of neural networks to adapt and learn or classify the patterns in in this data. They represent a powerful class of quantitative measurement tool, which have been successfully applied to many areas of industry, busyness and science. [8]

There are several reasons why the Artificial Neural Network model approach would be reasonable and promising for the problem faced. This include:

1. Neural networks as opposed to traditional model based methods do not require an estimation about an underlying data. The Artificial Neural Network modeling process is adaptive and learning is driven by patterns from the data itself. This approach is ideal in the cases when amount of data is plentiful but the knowledge about its structure are not yet discovered.
2. One of the mathematical properties of Artificial Neural Network include the universal approximation capabilities which make the network more general and flexible.
3. The Artificial Neural Networks are high order, nonlinear models this means that with their nonlinear and nonparametric nature they are more capable of modeling real life complex relationships.
4. They can be trained using imprecise or noisy data. This error can occur in datasets for many reasons including the human error.
5. Although training a neural network requires high computational power, the already trained network is computationally cheap and can be deployed in the user friendly business environment for an easy use. [9]

Since the approach is empirical in order to measure a success of the method, models predictive precision and accuracy will be compared using the provided data. The part of the dataset will be used for testing and the prediction made by the model will be then compared with the expected values.

3. DATA PREPROCESSING

In this chapter I will describe the data provided by the insurance company and its properties. I will show you what fields and features are available for extraction of knowledge from the dataset, their types, ranges and their importance in order to select ones which are noteworthy and useful for Artificial Neural Network Training. Data preprocessing methods will be chosen and compared in order to prepare input feature matrix for obtaining optimal results.

Data preprocessing is an important step for creating a successful classifier model. A correct extraction of features, their normalization and encoding will provide the best input for training and adjusting the predictive model. The aim of the good preprocessing step is to remove irrelevant and redundant information or noise from the data and the result is the final training set which will be used as an input for the network.

As several past researches have shown, the overall success of the learning algorithm is strongly dependent on the quality of the training data. Data which is too noisy or inadequate will result in classifier which may perform poorly or not be good for any use at all. A well planned data preprocessing step can have a significantly positive impact on generalization performance of an algorithm. [10]

3.1. Provided dataset characteristics and problem definition

The dataset provided by the insurance company is represented as a data matrix consisting of the customer information stored by the company. The data is depersonalized meaning that all the private information like personal identification code is not accessible. This data table consists of features or fields represented as columns and 540 490 samples represented as rows of the said matrix. Each row represents a distinct customer of the company and columns hold the information for each of them.

The data holds several types including numerical or scalar types, categorical types including binary or true/false values. The units for the data were given by the insurance company and they were not altered, since they don’t have an effect on model training. The aim was to prepare an appropriate subset of this data and encode it for creating an optimal training set for statistical study.

3.1.1. Numerical data

Table 3.1. Numerical features of the dataset

Feature name	Min. value	Max. value	Description
Contract start date	734 553	736 207	Number of days since year 1900
Contract duration	1	366	Duration of contract in days
Engine power	1	456	Represented by horsepower - hp
Vehicle age	0	91	Number of years since the build date
Vehicle mass	110	47 700	Mass of vehicle in kilograms - kg
Number of seats	1	55	Number of seats in the vehicle
Customer age	1 474	42 118	Number of days since the birth date
Number of claims	0	4	Number of filled claims by the customer

As it can be seen from Table 3.1 we have quite a few numerical features available, which can be used for the network training. Contract start date and contract duration represent the date when the contract became active between the insurance company and the customer and for how long it would remain active. Start date is represented by the number of days starting from 01.01.1900 up until the day of activation of the contract. The duration is the number of days for which the contract will remain active and it does not go above 366 days or 1 year.

Features like the engine power, vehicle age, and vehicle mass represent the technical state of the customer's car. This can be useful since it can be assumed that the good state of the car, little mass and reasonable engine power can help in preventing an accident.

Customer age represents the number of days since the birth date of the customer, up until the date of the insurance contract activation. This feature will help in weighting age as a risk factor.

It should be noted that the last feature "Number of claims" represents the incremental counter of claims submitted by the company customers. This field shows us how many times accidents were registered in the insurance company database and this feature will be a target data for the classification. It can assumed that if number of claims per customer is more than 0 it means that this customer should be classified as a customer of higher risk.

3.1.2. Categorical data

Table 3.2. Categorical features of the dataset

Feature name	Min. value	Max. value	Description
Yes/No Contract	0	1	Agreement for the contract
Policyholder loyalty	1	8	Customer type
Gender	0	1	Customer gender
Usage type	1	18	Vehicle usage type
Risk region	1	20	Region in which customer is registered
Vehicle maker	1	249	Maker company of the vehicle
Vehicle model	1	16 695	Model of the vehicle
Fuel type	1	7	Fuel type vehicle is using
Frame type	1	49	Vehicle frame type

Categorical data represented in the dataset is of different types as it is shown in Table 3.2. Some are represented as Boolean values but most of them are enumerations of possible categories from the database. This enumerated categories are represented as foreign keys in the data table. The hold the identity number of the category for the said data entry. The table for categories hold the identifier for the entry and the name for the entry.

3.1.3. Problem statement

Now it is possible to clarify the problem definition which is to create a binary classification algorithm which will receive as an input customer features and will classify them as the one who will fill the claim in future or not.

It should be noted, that studying number of claims it was found that only 11 551 entries or approximately 2,137% out of whole dataset consists of customers who have filled any claims during their insurance contract period and other 97,863% have never done this. Since the classification categories of customers who have filled at least one claim and customers who

have not filled any are not represented equally, this means that we have a binary classification problem with imbalanced dataset, which will be taken into consideration, since classification of imbalanced datasets, as research has shown, needs additional and special methods for creation and evaluation of the model, which will be discussed in details. [11][12][13]

3.2. Training data preparation

Before starting the classification model training process the training set should be prepared. The model based on the neural networks can only get as accurate as the data that was used for training the said network. It is important for the sample space, within the training set, cover as much of the range of the possible input space as possible. [14]

In order to get the best results from the given dataset we have to prepare the subset data as an input for the Artificial Neural Networks. This step includes: handling missing attributes, normalization and scaling of the data and categorical data encoding. Each of these steps will be discussed in details.

3.2.1. Missing attributes

Dealing with incomplete data or missing attributes is a common problem of studying real life datasets. They can occur when information is not available at the moment of data creation or by just human error. This problem was not an exception. In the numerical attributes like: vehicle engine power, vehicle mass and customer age there were quite a few missing values represented as zeros which does not make sense for the study purposes and should be dealt with since this might create a bias for our classification model [10]. Shown in Table 3.3.

Table 3.3. Missing values attributes

Feature name	Number of missing attributes	Percentage of the data	Mean
Engine power	36 481	6,75%	85,0069
Vehicle mass	1 391	0,26%	2 188,4
Customer age	1	$1{,}85 * 10^{-4}\%$	16 507

Several missing attribute handling methods were considered and most appropriate was found to be the mean substitution method for each missing attribute. The missing values were replaced by calculating means of the available attributes of the features and filling the missing values with the mean.

3.2.2. Scaling of the data

Data normalization or scaling is an approach used in order to standardize the range of input variables. It should be noted that this step of data preprocessing deals with the numerical input features only. Since the range of each individual feature attribute may vary widely it is a best practice to map or scale them to the same range, so that each feature will contribute proportionally on the training process. When dealing with Artificial Neural Network classifiers, like in this case, this is not a necessary step but as practice shows normalization can improve the efficiency and speed of training algorithm which will lead to better classifier overall [15].

One of the methods for scaling is Min-max (MM). It scales the feature attributes to values to the required interval which in the case of the given problem was chosen to be recommended and common [0, 1] interval. This will rescale all the data of the feature and represent minimal value as 0 and maximal value as 1. The normalization equation is give as Equation 3.1. [16]

Equation 3.1. Min-max scaling [16]

$$MM(x_{ij}) = \frac{x_{ij} - x_{min}}{x_{max} - x_{min}}, \tag{3.1}$$

Where x_{ij} – numerical feature attribute value,

x_{min} – Minimal value of the feature,

x_{max}- Maximal vale of the feature.

3.2.3. Categorical data encoding

Categorical in the given database is represented using an identification number for each types of the category. This approach is reasonable when we want to optimally search through the database but this identification numbers have no information value for our Artificial Neural Network training. Since the identification numbers are just incremental they don't represent

any information about the underlying data and they don't have any particular ordering which will be helpful for knowledge discovery. This was also proved empirically. When trying to train the network using the foreign keys of each categories it was behaving as a random noise and gave no classification benefit to the model. Even more this noisy data was leading to a poor and unusable classification model. This means that the given categorical data should be encoded so it can become useful for training purposes.

When dealing with categorical data as an input to the Artificial Neural Network the standard approach is to apply 1-out-of-N encoding. This means that all the possible values of the categorical feature are represented as a binary input to the network. We can imagine such an approach as representing the categorical data as an identity matrix of dimension n where n is the number of all the possible values of the feature. [17]

This approach has 2 drawbacks in case of the model we want to build:

1. First is that the number of categories gets quite big. For example the number of possible values a vehicle model can get is 16 695 as it is shown in Table 3.2. This means that just the data for encoding possible vehicle models at the input layer of our network should be increased by 16 695 which is a drastic increase and will affect training times. This option is not best since it will be computationally expensive to have such a large number of inputs.
2. Second and bigger problem with using 1-out-of-N encoding for the purposes of this research is that in real life scenario new vehicle models will be registered into the database of the insurance company. This means that input size will increase dynamically during the deployment period of the classifier and the existing network will be unable to adapt to this change. In order to adapt to changing input size the network must be retrained every time new categorical value is introduced to the system. This is result in very inflexible and unusable model.

Considering this two arguments a new representation of the categorical features should be created so it can correctly represent the data and also should be useful to the training meaning it should represent the importance of the feature by its magnitude.

For this purposes a new variable is introduced which will be called a risk factor (RF). It is calculated by the Equation 3.2. This variable represents the probability associated with the said categorical value of how likely is this category to fill the accident claim in the future based on the previous entries in the database. The calculated risk factor is within the range of [0, 1], since

it is a probability, and this values are used in the training set for the network. This approach solves both of the problems addressed above. Firstly the number of inputs will be equal to number of categorical variables and not to all possible values it can get and secondly when the new category is registered in the database its risk factor can be calculated and used as an input for the model without retraining it each time.

Equation 3.2. Risk factor calculation

$$RF(x_{ij}) = \frac{\sum_{k=0}^{n}[f(x_{kj}) * g(x_{kl})]}{\sum_{k=0}^{n} f(x_{kj})}, \tag{3.2}$$

Where x_{ij} – categorical feature attribute identification number,

n – Number of samples or rows in the database,

i – Row index of the attribute in the database,

j – Column index of the attribute in the database

l – Index of "number of claims" column in the database

$$f(x_{kj}) = \begin{cases} 1, & if\ x_{kj} = x_{ij} \\ 0, & otherwise \end{cases}$$

$$g(x_{kl}) = \begin{cases} 1, & if\ x_{kl} > 0 \\ 0, & otherwise \end{cases}$$

3.3. Dimensionality reduction and feature selection

Since training is computationally expensive algorithm a dimensionality reduction was considered. A reduced training set will improve the speed of the network training and also the input vector size needed for the classification will be reduced and easier to acquire. Also this step of preprocessing will reduce the number of irrelevant and redundant dimensions. [9]

The objective of feature selection step is to identify features in the dataset which are important for the classification and discard others which are irrelevant and redundant. The approach which was considered on the preprocessing stage is called a correlation based feature selection. The aim is to calculate correlations between the categorical features and number of accident claims.

The good features should be highly correlated with the target class and yet show low correlation between each other. [18]

Some of the fields from the provided dataset can be discarded from the beginning. Features like the contract identification code and vehicle registration plate have showed no correlation with the target class and were not considered for including in the training set. For the other categorical features a correlation based feature selection, method which is discussed in research paper [18] was applied. The results can be seen on Figure 3.1 from which we can see that for example vehicle model shows the highest correlation and can be used as an important feature for classification. On the other hand features like risk region and customer gender have shown low correlation meaning that they can be discarded without a big difference for the final results. This assumption was tested empirically and indeed the features with low correlation did not affect the performance of the classifier much. All the results of conducted tests with selected features are shown and discussed in Chapter 6.Model Evaluation and Fine-Tuning.

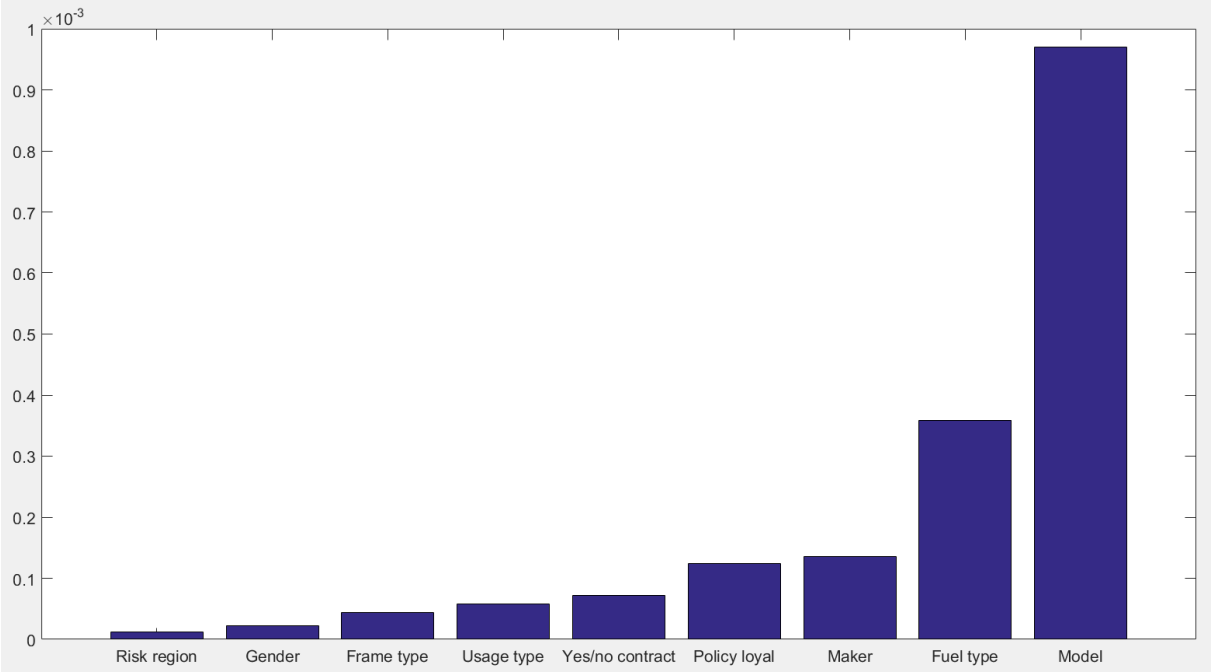


Figure 3.1. Correlation of categorical features with the target class

To summarize the data preprocessing step of the research, it can be said that this tests and methods have shown a good insight into the data which was provided by the insurance company. I have studied the properties of characteristics of numerical and categorical data provided, normalized and scaled all the scalar features, so they can be affectively used for training and the importance of each categorical features was analyzed.

3.4. Data splitting and sampling

Another important method of training data preparation for artificial neural networks includes data splitting in order to perform a cross-validation and evaluation of the classifier generalization possibilities. This step helps to identify at which point of the training of the network gets the optimal performance and when to stop or tune the training process in order to avoid overfitting on the data and to get the best performing model. [19]

In order to avoid a poor generalization and overfitting by the classifier, a common method is to apply the hold-out cross-validation method [20]. In order to successfully apply this approach for evaluating the model, which is discussed in details in chapter 6, an appropriate data splitting is required.

When training a neural network the most commonly used practice is to divide the samples or rows of the training matrix into three mutually disjoint subsets [14]. These subsets are:

1. Training set – this set usually is the largest and it is used for tuning the network weight parameters and to compute the gradients at each iteration of the training algorithm.
2. Validation set – the error calculated using the validation set is monitored during the training process. Usually both training and validation errors decrease at the beginning of the training but as soon as the validation set error starts to grow this is a good indication that training method has started to overfit the data, which is unwanted behavior. Than the weight parameters at the minimal validation set error are being chosen.
3. Test set – which is not used in anyway during the training step, but it is used on an already trained model for evaluating its performance with an independent set, never used before.

In order to get the same distribution of classified classes in the subsets as it is in the whole set, the data is randomly divided into subsets. The division ratios may vary, but general approach is to get 70% of data as training subset, 15% as validation subset and 15% for testing subset. This operation is achieved by using Matlab function `'dividerand'` as a network divider function `'net.divideFcn'`. [14]

3.5. Target data

Since the problem to be solved is a supervised classification problem we will need to provide the network with the correct or expected outputs for each of the training samples, which the training method of the network will try to approximate by tuning the network weights. For the target data a binary matrix will be used with number of rows same as the number of sample in the training set and number of columns representing the possible values classificatory model should output. In our case the number of possible class outputs is 2. Customers with low risk of filling an insurance claim and second class representing customers which have high risk of filling the claim.

If in the training sample the column represented as “number of filled claims” is greater than zero, meaning that the sample has filed the accident claim, then the target matrix row will be encoded as (1, 0) row vector. In case if the number of filled claims is equal to zero than the target matrix row for this sample will be encoded as row vector (0, 1).

This target matrix will serve as a function values which should be approximated by the training algorithm of the Artificial Neural Network based on the input feature vector.

4. NETWORK ARCHITECTURE

Next step in workflow of the neural network classifier design, after preparing the training data, is to create the artificial neural network object, configuring the network parameters and initializing it for further steps of training and evaluation. Selecting the right architecture depends on the problem specification which needs to be solved and it is dictated by the requirements to the classifier. A well suited architecture can improve training possibilities of the network and will produce a better overall result in the end [21].

In this chapter several promising approaches will be considered and compared. Designing steps will be taken for each architecture which will be considered. This includes:

- Defining network input layer and number of input nodes based on the input characteristics for the network.
- Definition of the network hidden layers, finding the required number of hidden layers and layer parameters like number of neurons and activation or transfer function for each of them.
- Defining an output layer with number of neurons based on the output requirements of the classifier model and activation or transfer function based on characteristics of the output data.
- Network initialization methods for optimal training.

One of designs covered in this chapter, considered as a state of the art approach when dealing with classification problems, is a Feed forward multilayer network (FF). In several past studies they have shown good performance for prediction and classification for insurance ratemaking [2]. This architecture is well studied and has shown good flexibility in training and adapting possibilities for vast number of classification problems [21].

Another approach which has shown a promising performance capabilities in classification problems, especially problems which are dealing with imbalanced classes [22], is called autoencoder or replicator neural network architecture. The specifications of this architecture will be discussed in this chapter.

For both of this architectures the benefits and drawbacks will be considered and the final result of this two approaches are shown in Chapter 6.Model Evaluation and Fine-Tuning.

4.1. Feed-forward network architecture

The main objective of this is to briefly explain the architecture and working principles of the artificial neural networks, their properties and characteristics. The basics of the approach will be covered since explaining the details would require a prolonged explanation, which is not included into the scope of the study. Majority of the information was taken from several sources including but not limited to [21], [15], [2] , [23] and [24]. These references can be used for more detailed information about the methodology.

A feedforward neural network (FF) is an architecture where connections between the layers of the network do not form a cycle. Typically feedforward network contain an input layer which gets the input data and propagates it forward into the next layer called the hidden layer. After this the propagation continues to the output layer where the classification results have been acquired.

The most basic component of an artificial neural network is a single neuron. A single neuron has an inputs which are coming from the input vector or from other neurons. These connections have weight associated with them, which represent the priority of an each input. The input vector is than multiplied by the weight vector and the result is passed to an activation or transfer function, which can be chosen for each layer individually. There are several commonly used transfer functions. The requirements for these functions are to be defined for the whole real number set and it has to have a well-defined derivative for training purposes [21]. The single neuron model with multiple inputs can be seen on Figure 4.1. The output of the neuron can be calculated using an Equation 4.1.

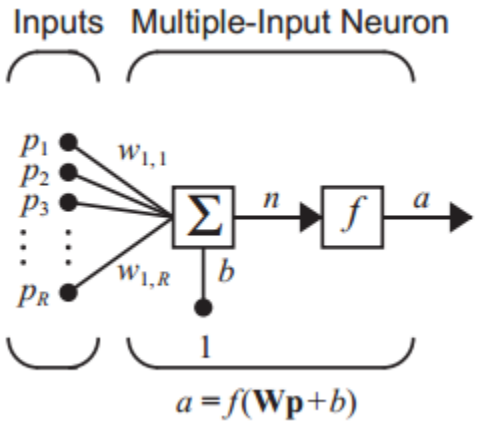


Figure 4.1. Multiple-Input neuron model [21]

Equation 4.1. Neuron output calculation [21]

$$a = f(W * p + b)\,, \tag{4.1}$$

Where a – an output value of the neuron,

f – A transfer function of the neuron,

W – The row vector of neuron connection weights,

p – Vector of input values,

b – Bias weight of the neuron

In most cases using only one neuron is not sufficient and the layer of neurons should be introduced. The neurons in the same layer operate in parallel and are not connected to each other. The model of the layer of neurons is shown on Figure 4.2. In case of the layer the weight row vector, which we had with the single neuron, is changed to the weight matrix representing the connection weights for the whole layer. The bias value is a value for each neuron and for the layer it is represented as a vector of biases of the layer. This means that the layer output shown in Equation 4.1 is now outputting a vector of output values which will serve as an input vector for the following layer. With the same approach several layer can be stacked in sequence to get a multilayer neural network [21].

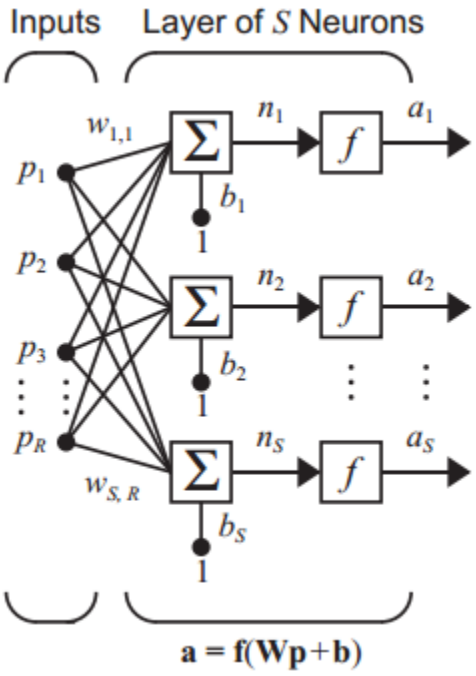


Figure 4.2. Layer of neurons [21]

4.1.1. Input layer

First layer which has to be considered is and input layer. It will receive the input vector for each sample and propagate it forward into the network. The size of an input layer only depends on the number of features which are used as an input for the network. As a starting point the subset of the numerical and categorical features were considered, which were discussed in Chapter 3.1. The ordering and the encoding of each input is shown in Table **Error! No text of specified style in document.**4.1. The encoding of each feature was done with Equation 3.1 for numerical data and Equation 3.2 for categorical data.

It should be noted that features, which have shown low correlation with the target matrix represented on the Figure 3.1., like gender, risk region and frame type were omitted from the list of input features. The motivation for this is to simplify the model and training as much as possible without decreasing the classification capabilities of the model, using as small number of features as possible. When comparing the models with and without these features no significant difference in performance was observed, as expected.

This selection of features fixes the number of neurons in the hidden layer to 11.

Table **Error! No text of specified style in document.**4.1. Input features of the network

No	Feature name	Min.	Max. value	Encoding
1	Yes/No Contract	0	1	A risk factor within the range [0, 1]
2	Contract duration	1	366	Min-max scaled to range [0, 1]
3	Policyholder loyalty	0	1	A risk factor within the range [0, 1]
4	Customer age	1 474	42 118	Min-max scaled to range [0, 1]
5	Usage type	0	1	A risk factor within the range [0, 1]
6	Engine power	1	456	Min-max scaled to range [0, 1]
7	Vehicle age	0	91	Min-max scaled to range [0, 1]
8	Vehicle model	0	1	A risk factor within the range [0, 1]
9	Vehicle mass	110	47 700	Min-max scaled to range [0, 1]
10	Bonus malus	0	1	A risk factor within the range [0, 1]

11	Fuel type	0	1	A risk factor within the range [0, 1]

In order to choose the optimal transfer function for the input layer the best practices were used as described in [15]. Nonlinear transfer functions give the network their nonlinear classification capabilities. One of the most common transfer function type used in artificial neural network training is the sigmoid type functions. These functions approach some finite value while the argument approaches positive or negative infinity. The function chosen for the input layer transfer function is a hyperbolic tangent, which is shown in Equation 4.2. This function is being chosen, because it often converges faster to an optimal solution compared to other sigmoid functions [15].

Equation 4.2. Hyperbolic tangent transfer function [15]

$$\sigma(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}, \tag{4.2}$$

Where σ – a transfer function of the layer,

x – An input argument for the transfer function.

In Matlab environment the transfer function of the input layer is defined by the `'net.layers{1}.transferFcn'` property, which is set to a hyperbolic tangent function with function name `'tansig'` [14].

4.1.2. Hidden layers

Next step is to set up the hidden layers of the network. After the input layer calculates the output, it is pass to the hidden layer as an input and the next step of creating the neural network architecture is to decide how many hidden layers to use and with how many neurons in each.

The transfer function for the hidden layers remains the same as it was for the input layer. It is set to hyperbolic tangent.

Increasing number of neurons also increases the complexity of the model which might lead the model to overfitting and poor generalization, but on the other hand using too simple model will not get the good performance. There are no best practices for selecting the best number of these parameters, other than the empirical testing of each of them. As tests have shown increasing the

number of hidden layers strongly increases the time needed for network training but it does not translate to network performing well. For this tests each approach was measured with F_1score, which is one of the main classifier evaluation method and it is discussed in details in Chapter 6 of this work. Higher the F_1 score, the better is the overall performance of the network.

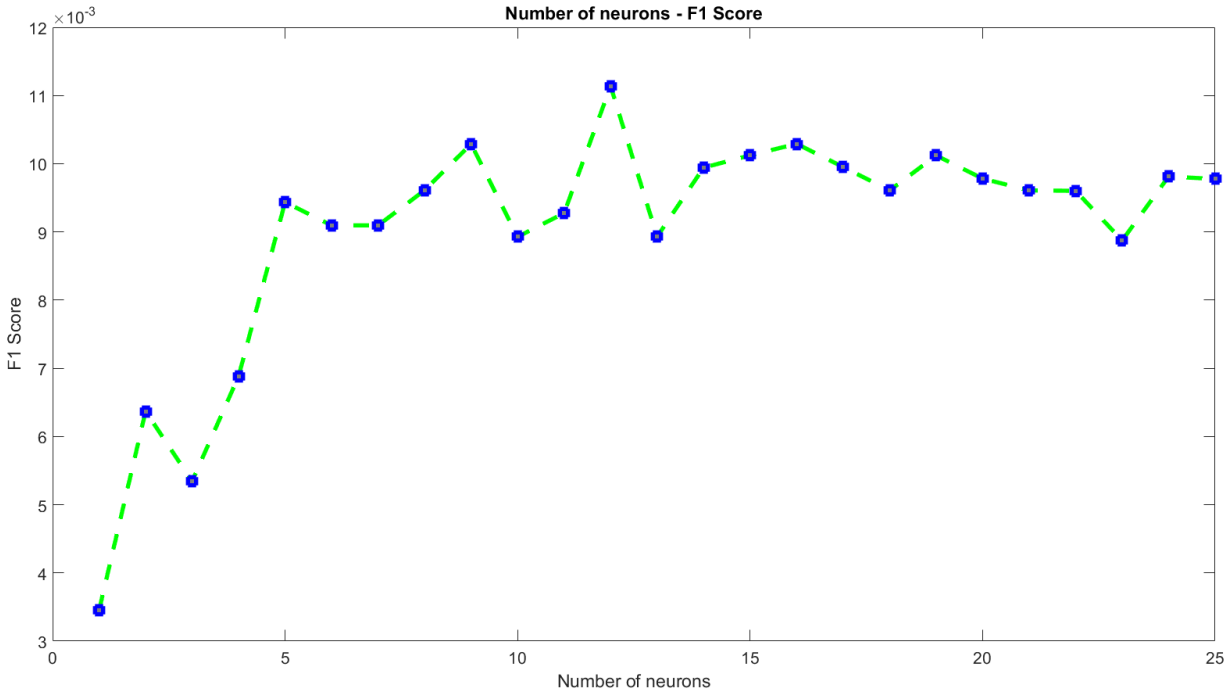


Figure 4.3. F_1 Score over the number of neurons in the single hidden layer (higher the better)

From the Figure 4.3 we can see that number of nodes in the hidden layer affects the performance or classification capabilities of the model. Setting number of neurons too low gives us considerably lower results. The performance capabilities increase with the number of neurons up until the breaking point, which is around 5 nodes per layer. Interesting point to note is that after this optimal point is reached, the F_1 score starts to fluctuate and overall does not increase too much.

There is an element of randomness in these results since the network weight matrix initialization is done with random numbers and it gives different starting points for the training algorithm, but similar results were gathered after repeating same test for several times. This plot was generated by testing each setting separately and since the results did not change much in this

range, it was assumed that neurons in the range of 10 to 15 nodes result in most optimal classificatory model. This number of neurons were also used for further testing of the network.

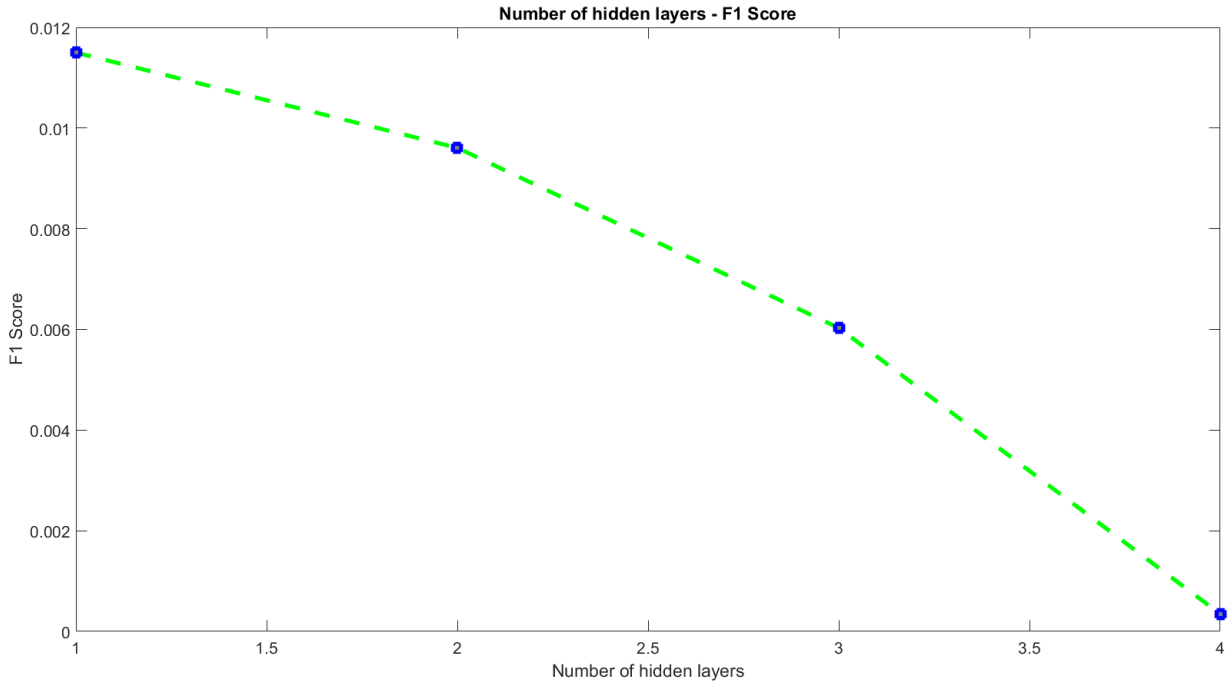


Figure 4.4. F_1 Score over the number of hidden layers of the network (higher the better)

After fixing the optimal number of neurons for the single hidden layer, I also studied the potential benefits of increasing the number of the hidden layers and measuring the performance difference for each of them. Although using multiple hidden layers, or "deep networks" as they are called, have proved to be a powerful network architecture for different problems [25], it does not mean that it would be a good solution for this problem also.

Increasing number of layers dramatically increases the training time and the neural model function complexity, which can lead poor generalization and high variance [15].

As it can be seen on Figure 4.4 the performance of the network starts to degrade with the addition of the hidden layers. The decrease is stable and the same results were shown after repeating the tests for multiple times. From this results it was assumed not consider increasing the number of hidden layers, since they did not provide benefit for the classification and they result in higher processing power and time for training.

Conducting this tests helped to better understand the complexity of the problem. This results will help to create an artificial neural network model which will be prone of high variance and should generalize well for getting better classification performance.

4.1.3. Output layer

As much as the input layer was influenced by the number of features in the input vector, the same decision is to be made with the output layer. There are two basic properties to be set. One is number of neurons in the output layer, which is dictated by the encoding of the target matrix, explained in Chapter 3.5 and number of classes which the classifier should distinguish.

The other important characteristic is the activation or transfer function of the output layer. Since this layer is responsible for giving the final answer of the model, it is much more dependent on the transfer function used on this layer.

Since the aim of the model is to perform a binary classification, it is a best practice to use 1-out-of-N encoding [17]. This encoding approach outputs a vector with probability values, where each element represents the probability of the input sample belonging to each class. The size of the vector is same as number of expected classes, which is 2 for this problem.

Dictated by the target matrix our classifier will output a vector (0, 1) for the input samples which will classify customer as having no risk of filling the claim and the output vector of (1, 0) will represent the customer with certainty of filling the vehicle accident claim. For the general form of the output it can be defined as (p1, p2) where p1 is the probability of the sample belonging to class number 1 and p2 being the probability of sample input belonging to class number 2.

Naturally, when dealing with this type of encoding, the sum of elements of the output vector will sum up to 1. This approach is also justified by the fact that network outputs not only binary yes or no answer but the probability or certainty of the sample belonging to the class. This means that these probabilities can be used for proper thresholding of the output. This step is discussed in Chapter 6, when fine tuning the model.

In order to achieve this type of probabilistic classification the transfer function which is called normalized exponential of Softmax is used [24], which is a generalization of the logistic sigmoid function. This function meets the requirements needed since it outputs the values in the range of 0 to 1 and sum of the outputs adds up to 1. Softmax function is the gradient log

normalizer of the categorical probability distribution and for this reason it is used in various multiclass classification problems [24]. The function is calculated by the Equation 4.3.

Equation 4.3. Softmax transfer function [15]

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \tag{4.3}$$

Where σ – a Softmax transfer function which returns the probability array,

x_i – An input argument for the transfer function,

n – Number of classes to classify.

4.2. Deep autoencoder or replicator network architecture

Another architecture which was considered for the neural network is called Autoencoder or Replicator neural network. This architecture has shown quite a promising results in several fields and particularly in classifying imbalanced datasets. It is often used in studies when dealing with fraud or anomaly detection in the dataset, where they perform with good efficiency [22], [26].

The objective of the autoencoder architecture is to learn a representation or encoding of a data for a purpose of dimensionality reduction [27]. Although it is similar to the feedforward network, described above, in the sense of propagating the output forward into the network without loops, it has a difference in training approach. Each hidden layer of the network, called an encoder, is trained separately and with different target data. After each layer is trained they are stacked in sequence with an edition of the output layer which gives the final output of the predictive model. [14]

The aim of training the encoder layer of the network is for it to replicate the input at its output, but using the reduced dimensionality for achieving this. Training of each layer is unsupervised. This meaning that no additional labeled target data is needed for each layer since the input data itself is used as the target to be approximated. [22]

Model of this architecture can be seen on Figure 4.5. On this figure it is shown that an input vector is propagated forward to the layers with decreasing number of neurons. This layers are trained to encode the data in the smaller dimension because they are forced to replicate the same data at the output. [22]

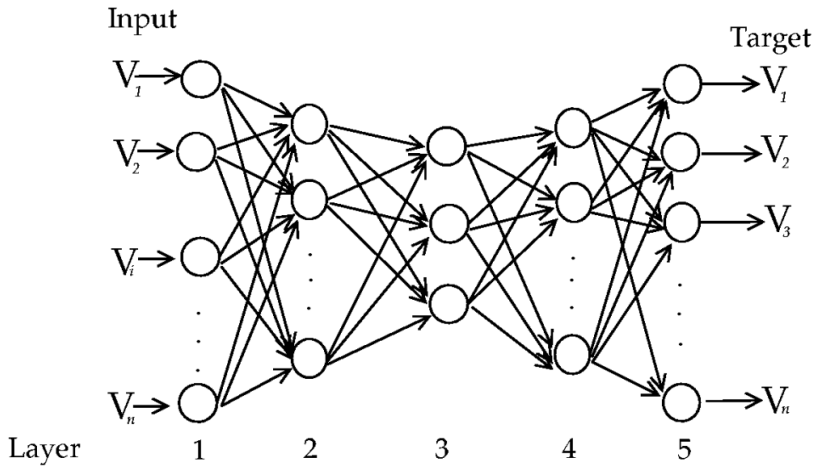


Figure 4.5. A schematic view of a fully connected Replicator Neural Network [22]

4.2.1. Encoder layers

First step of creating the stacked autoencoder classifier is to first train the encoders by using the input feature vector. Several encoder layers can be trained sequentially and then stacked together with an output layer, in order to produce a strong classifier.

As an input for the first encoder layer the same input features and encodings were used as shown in Table **Error! No text of specified style in document.**4.1. This input layer with 11 neurons than was connected with the encoder layer consisting of 10 neurons. After which this layer is connected to the output layer consisting again of 11 neurons, same as the input. The same input date is used at the output as the target vector. This layout gives the possibility to extract 10 dimensional feature vector which will replicate the input data at the output.

Training two encoder layers and stacking them in sequence has shown the best performance for the classifier model and this layout is also the recommended one from the documentation [14].

There are several parameter which can be adjusted during the encoder training. The recommended parameters from the Neural Network Toolbox documentation [14] were used and tweaked if needed. This parameters include:

- Transfer function of the encoder – much like the layers in the feedforward network discussed in Chapter 4.1.2, the sigmoid transfer function has shown to be the most optimal decision to be chosen.
- Weight regularization parameter – the coefficient which is used for regularizing the magnitude of the network layer weights. This parameter is important in order to keep the network ability in generalization and to keep balance between overfitting and under fitting model. Since the coefficient of 0.0001 has shown any limitations in network's performance it was chosen as the regularization coefficient.
- Sparsity regularization – a positive scalar coefficient enforces a constraint on the sparsity of the output of the network. Sparsity can be measured by different regularization terms. The Kullback-Leibler divergence [28] was chosen for this project purposes, since it measures how different the two distributions are. The value is 0 when two distributions are the same and it increases as they diverge.
- Sparsity proportion – is a positive scalar value between 0 and 1 and it represents the desired proportion of training examples a neuron reacts to. A low proportion leads to each neuron only giving a high output for a small number of training samples, meaning that a low proportion results in higher sparsity.

For training the encoder layer a built in Neural Network Toolbox method `'trainAutoEncoder'` was used, implemented in Matlab environment. [14]

4.2.2. Output layer

For creating the output layer of the autoencoder network the same principles were considered as discussed in Chapter 4.1.3. The output layer is represented by a 2 neuron layer with Softmax activation function, which outputs the probabilities of the input belonging to each class.

After the encoder layers are trained, the next step is to train the output Softmax layer, in order to stack it next after the encoder layers and use it as an output of the network. For the output layer training the feature vector output from the previous layer is used as a training data. The accident claim binary target data, discussed in Chapter 3.5, is used at an output.

For training the autoencoder output layer a built in Neural Network Toolbox method ‘trainSoftMaxLayer’ was used, implemented in Matlab environment. This function return a network object with the same number of neurons as number of classes in the target data [14].

After all the layers are trained and prepared the Matlab Neural Network toolbox function ‘stack’ is used for chaining the encoder layers and output layers together.

4.3. Initializing the network

In order to achieve maximum training ability from the network, it is an important step to consider the network initialization approaches, both for feedforward and autoencoder network architectures. The initialization step implies setting the weights of the network connections and biases to the values which will help the training method converge to an optimal solution. The correctly chosen initialization method will also decrease the necessary training time [29].

When the aim is to train the multilayer neural network, the most common approach is to initialize weights and biases to small random values. For example uniformly distributed random values in the range of [-0,5, 0,5]. Problem with initializing the network to zero values is that the training algorithm may fall into the saddle point of the performance surface and this will result in suboptimal solution. On the other hand setting the initial weights and biases to large values will make the training algorithm fall on the flat part of the performance surface, which also provides non satisfactory results [21].

Also there is another approach, commonly used in multilayer architectures. This approach considers setting weights and biases according to the Nguyen-Widrow algorithm [29]. The aim of this algorithm is to choose values in order to distribute the activity region of each neuron evenly on to the input space. This algorithm also contains the degree of randomness so training algorithm will be able to perform well and converge faster. The requirement of this initialization function is to use the transfer functions with finite output range for which the chosen hyperbolic tangent function meets this requirements. Another benefit of this approach, other than improved convergence time, is that fewer neurons are wasted since all of them cover the input space [21].

In Matlab environment the Nguyen-Widrow layer initialization function was implemented by setting the ‘net.initFcn’ property to ‘initlay’ and then setting each layer property ‘net.layers{i}.initFcn’ to an inbuilt neural network toolbox function of 'initnw'. [14]

5. TRAINING OF THE CLASIFFIER

The next necessary step in the flow of neural network design, after initializing the network state by setting weights and biases, is to train the network. The process of network training involves choosing the values for weights for connections and biases so that the error between the input feature vector and output target vector is minimal. [14]

In this chapter several promising training methods for classification purposes will be considered. Each have their benefits and drawbacks when applying to the network training and each of them works best with particular set of problems. It is difficult to know beforehand which training method will perform the best for a given problem. It depends on problem complexity, number of samples in the training dataset, number of connections in the network and the error goal. Recommended way is to try several best practices and compare the final performance [21].

For training the neural network a numerical optimization algorithm should we used, which will optimize the performance function, or to put in another way, will try to find the minimum of the performance curve, where the error is minimal and corresponding weights will be the solution of the optimization problem [21]. In many sources performance function is also mentioned by the name of cost, error or loss function.

In addition to finding a minimum of the function the aim of the training method is to find weights which also make a model which will generalizes well during testing and deployment. There are two most common generalization techniques for maximizing the network performance when using the backpropagation training [15]. One of the techniques is accomplished by adding a regularization term to the performance function. The regularization, also called weight decay method in several sources, will be discussed in Chapter 5.1.3. Another technique is called an early stopping, which forces the training method to stop until it starts overfitting. This approach is discussed in Chapter 5.2.

There are several performance evaluation functions and generalization methods, which are commonly used in training applications. In the Chapter 5.1 the most common and recommended ones are considered and implemented.

Out of many numerical optimization approaches, there are several that have shown an excellent performance for an artificial neural network training purposes. This techniques are called

gradient-based learning methods, which involve the error backpropagation backwards through the network and based on gradient of the error between the expected output and actual output the weights are tweaked in the direction which minimizes the error [15]. There are many tuning parameters for this method which are discussed in details in Chapter 5.2 and the theoretically most promising ones are implemented for further testing.

Other methods for speeding up the training algorithm, other than good initialization and data preparation, is to use parallel or distributed computing methods for improving the computational efficiency of the algorithm. This approaches are discussed in Chapter 5.3.

5.1. Network performance function

The performance function used by the training algorithm is used for measuring the error size between the input and output vectors of the network. The input argument for the cost function is the weights and biases of the network and as an output it gives the measure of how good is a neural network at approximating the output. [15]

The optimization is done by computing the gradient on the performance function surface at the corresponding point of the network connection weights. Using this computed gradient the weights are than updated by the value depending on the learning rate of the algorithm and the magnitude of the gradient [21].

Two most commonly used functions in classification problems, which fulfill the requirements and has shown good performance are the Mean Square Error performance function or MSE and Cross-entropy. This two performance functions were implemented and tested for the purposes of designing the classifier neural network.

5.1.1. Mean squared error performance function (MSE)

Mean squared normalized error performance function is a function which measures the networks performance by calculating the mean of squared errors between the target outputs and predicted network outputs. This functions is defined with the Equation 5.1.

This performance function is the standard evaluation function for multilayer neural networks. It performs best when all inputs in the training set have equal probability of occurring [21].

Equation 5.1. Mean squared error performance function [14]

$$E = \frac{1}{N}\sum_{i=1}^{N}(t_i - a_i)^2 \ , \tag{5.1}$$

Where E - mean squared error performance function,

N – Number of samples in the training set,

t_i – The target output value of {0, 1} for the sample i,

a_i – Neural network output value of {0, 1} for the sample i

In Matlab environment the mean squared error function is implemented by setting the `'net.performFcn'` property to the value `'mse'`. [14]

5.1.2. Cross-entropy performance function

Mean squared error performs well for function approximation problems, when the target output values are continuous, but when applied to pattern recognition and binary values then other performance functions may be more appropriate [30]. This is the reason why another function called cross entropy was considered as a performance function during the neural network training process. Minimizing the cross entropy performance function usually leads to a good classifier. The equation for this error evaluation function is given in Equation 5.2.

The cross entropy function can be characterized by its property of heavily penalizing outputs that are inaccurate and penalizing very little outputs which are fairly correct. A good argument for using a cross entropy function over more common mean squared error function, is given in a book by C.M. Bishop [30]. Problem with MSE performance function is that it tends to give similar absolute error values for each pattern and therefor it should give relatively large errors for small output values. This suggests that using cross entropy performance function is likely to perform better than MSE when estimating low probabilities. Since we are dealing with an imbalanced dataset where the accident probabilities are relatively low it is a reasonable assumption to test the cross entropy function and compare the results with more common MSE.

The comparison of the two is performed on the evaluation stage of the project and it is described in Chapter 6.

Equation 5.2. Cross entropy performance function [30]

$$E = -\sum_{i=1}^{n}(t_i \ln a_i + (1 - t_i)\ln(1 - a_i)), \quad (5.2)$$

Where E– Cross entropy performance function,

N – Number of samples in the training set,

t_i – The target output value for the sample i,

a_i – Neural network output value for the sample i

In Matlab environment the mean squared error function is implemented by setting the `'net.performFcn'` property to the value `'crossentropy'`. [14]

5.1.3. Regularization parameter for the performance function

One of the problems which may occur during the training process of the neural network is called overfitting. This problem arises when the neural network learns the exact representation of the training dataset and not the statistical model of the process. This is an important issue if we want the model to generalize well when dealing with new inputs. This problem can be diagnosed when error on the training set gets lower but the error for the validation set starts to increase [30].

One of the reasons of the overfitting occurring might be the size of the network being larger than needed for an adequate fit. This will lead to more complex model than needed and thus create the poorly generalizing model. Unfortunately there is no easy solution for knowing the needed network size beforehand for each specific problem. Another way of solving the overfitting problem is to collect more data and this should lead to more clearly optimizable statistical model [21]. But this option is not available for this study since the training set size is fixed and getting more samples is outside the scope of this research.

In order to avoid this problem a technique called regularization is discussed in this chapter. In order to find an optimal balance point between a too simplistic model resulting in high bias and an overfitting model resulting in high variance, there is need for a parameter which will balance the complexity of the model [30].

The regularization method involves modifying the chosen performance function by adding additional penalty terms and it also gives us means of controlling this penalization degree using a regularization parameter. The equation for the modified performance function can be seen on Equation 5.3.

Equation 5.3. Modified performance function with regularization [30]

$$\tilde{E} = E + \upsilon\Omega, \tag{5.3}$$

Where $\tilde{E}$– Modified performance function with regularization,

E – Performance function chosen for the training method from the methods discussed in Chapter 5.1.1 and 5.1.2,

υ– Regularization parameter controlling the extent of the penalty term,

Ω – The regularization term

In order to implement this modified performance function it was needed to determine and fix the regularization or the penalization function Ω. The requirement for this function is to return higher values as the complexity of the model increases. One of the simplest forms of the regularization function is called a *weight decay*. It has been shown empirically that using this function can significantly improve the generalization capabilities of the network [31]. The weight decay regularization function consists of sum of squares of the connection weights of the network. It is calculated by the Equation 5.4. Adding this term to the performance function forces the training algorithm to find optimal solutions with smaller connection weights, which results in smoother approximation function, which is less likely to overfit [21].

Equation 5.4. Weight decay regularization function [30]

$$\Omega = \frac{1}{2}\sum_{i=1}^{n} w_i^2, \tag{5.4}$$

Where Ω– Weight decay regularization term,

n – Number of connections in the network,

w_i– Weight of each connection of the network

The problem with regularization is that it is not trivial to find an optimal regularization parameter v. If the parameter is made too large than the model might overfit and if the parameter is too large than model might not adequately fit the data. There is a training method which automatically sets the regularization parameter, which will be discussed in more details in Chapter 5.2.

Since most of training methods can not automatically set this parameter an empirical test was done to determine which value of the regularization term will result in best performance. The test was done by modifying the cross entropy performance function and adding a weight decay regularization to it. The results are plotted on the Figure 5.1 where the argument is the regularization term and the value is the F_1 score, which is one of the main classifier evaluation method and it is discussed in details in Chapter 6 of this work. Higher the F_1 score, the better is the overall performance of the network. As the test has shown the highest performance score was achieved when regularization parameter is close to zero. This can be explained by the number of samples in the training set being much larger compared to the number of connections in the network, resulting in a model unlikely to suffer from overfitting [14].

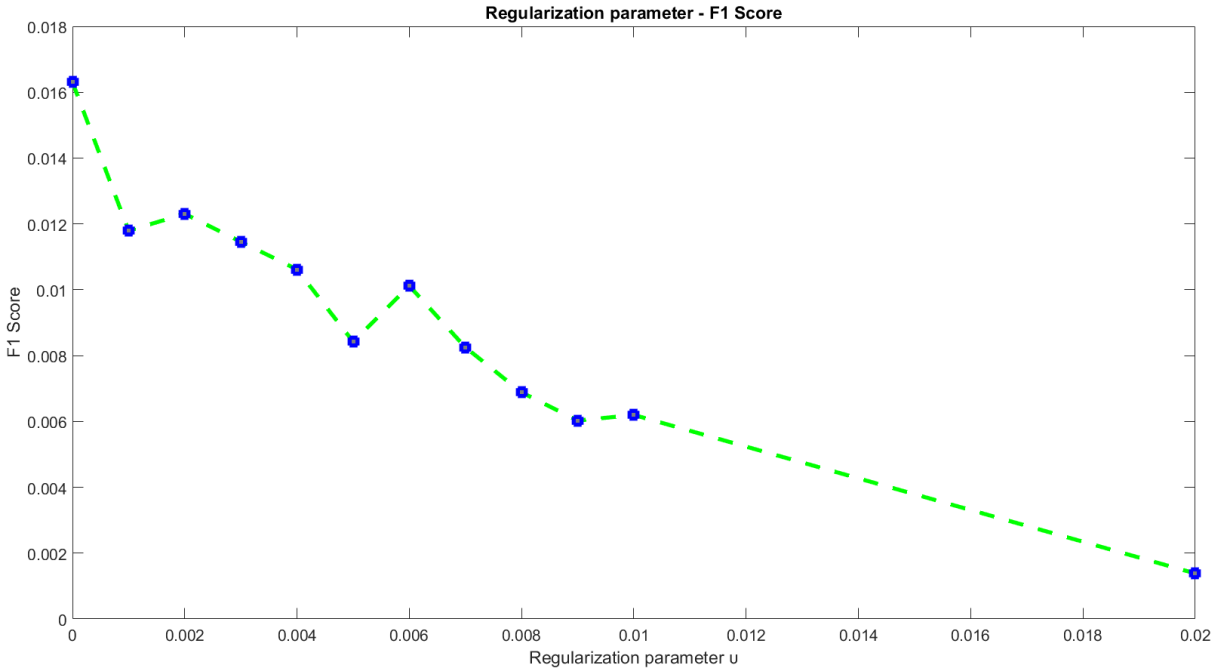


Figure 5.1. F_1 Score over the regularization parameter (higher the better)

In Matlab environment the regularization parameter is set by updating the `'net.performFcn'` property to desirable regularization parameter in the range of [0, 1]. [14]

5.2. Backpropagation algorithm

Backpropagation or backward propagation of errors is a common method for training multilayer neural networks used together with the gradient descent optimization method. It is a supervised method and requires a known expected output for each input sample since the optimization is done by calculating error between the actual and expected output. [23]

In this chapter a short overview will be given for this algorithm. Since it has been thoroughly studied by many researchers more information can be found in sources like [21], [15], [23] [24]. Backpropagation method is iterative and each iteration can be divided in two main parts: forward propagation and weight update during backward propagation. On the forward pass the weights of network connections are not altered and the input vector is passed through layers of network by using the Equation 4.1. On this pass the current weight values and layer transfer functions are used to calculate the actual network output. After the forward pass is done the error is calculated by comparing the output with the target value.

On the next phase of the backward pass the error calculated at the output layer of the network is passed back layer by layer. During the pass at each layer the error gradient is calculated and according to the learning rate parameter each weight is updated accordingly. It should be noted that the sign of the gradient indicates where the error increases so it should be reversed to find the minimum of the performance function. The weight update calculation procedure is shown on Equation 5.5. [23]

Equation 5.5. Network weight adjustment with delta rule [23]

$$\Delta \mathrm{w_{ij}}(n) = -\eta \frac{\partial E}{\partial w_{ij}}, \tag{5.5}$$

Where $\Delta \mathrm{w_{ij}}(n)$– Change in connection weight j at layer i,

n – Number of training iterations,

η– Learning rate of the backpropagation algorithm,

E – Error calculated using the performance function discussed in Chapter 5.1.

$\frac{\partial E}{\partial w_{ij}}$ – Partial derivative of the error with respect to weight w_{ij}

In order to acquire an optimal solution for the training dataset this weight update rule should be applied iteratively. Usually as the solution approaches the optimal point the error derivative decreases and the weight matrix get more precise. One of the hyper parameters which has to be chosen before the training is the learning rate η. If it is chosen too small, than the learning will be slow since the changes to weights on each iteration will be small. On the other hand setting high learning rate can introduce instability in training and the network state can start to oscillate without improving the solution. [23]

Although backpropagation algorithm is widely used when training then neural network, there are limitations and problems it might encounter. One of them is that it is not guaranteed to find the global minimum of the performance function. This can be caused by the non-convex nature of the performance function for the specific problem, meaning that the gradient descent will find the local minimum closest to the initial values of the network weights. This problem is partially solved by the random initialization step discussed in Chapter 4.3 which gives each training a new starting point, which might lead to a different solution and hopefully the better one. [15]

Another limitation of the backpropagation algorithm is the speed of convergence. The basic algorithm is too slow for most practical applications. To overcome this limitation there are set of variations for this method which provide a significant speedup and make the algorithm practical [21]. This variations are discussed, chosen and compared in Chapter 5.2.2.

5.2.1. Modes of training

After choosing the training algorithm, the next step is to consider the training modes it can work with. There are two different styles which are used when implementing a backpropagation algorithm:

1. Incremental training – also called sequential, on-line or stochastic style of training, is the mode when the network weight update procedure is performed after the presentation of each training examples to the network. During each iteration the training samples are used sequentially and network adapts to each of the separately. The benefits of using this mode include reduced storage requirements during training and since the training

samples are presented in random manner it is less likely the backpropagation algorithm will get stuck in the local optimum of the performance function.

2. Batch training – in this mode the weights and biases of the network are updated after all the input samples are presented for each training iteration. This means that the training data is fed to the network as a single matrix which is used for error calculation and weight update. Advantages of the batch training mode are that the conditions for convergence are well studied and understood also there are many accelerating techniques like conjugate gradient, which is discussed in details in the following chapter and another advantage is that theoretical analysis of the convergence rates are simpler [15].

Although the incremental training is simpler to implement when programming from the ground up, the Matlab documentation recommends using the batch mode since it performs more efficiently when compared to the sequential mode. [14] This means that batch mode will be used for the further training purposes.

5.2.2. Variations on the backpropagation method

The basic backpropagation algorithm, without any modifications, is based on the steepest descent optimization method, which is often slow to converge when dealing with real-life problems. In this chapter some heuristic modifications will be discussed which can improve the performance of the method. The ones which are more theoretically promising for the classification problem were selected for implementation and further testing.

When the gradient descent is applied to the problem there are some obstacles which can decrease the efficiency of the optimization algorithm. The convergence can get slow for multilayers networks where the performance function is not quadratic, is non convex and high dimensional with many local optimum points, where the convergence can get stuck. When the training dataset is large and multidimensional, as it is in this case, there are no guarantee that the network will converge to a good solution or even if the convergence occurs at all [15]. This is the reason why the modifications have to be chosen specifically for the large dataset classification problem, especially when dealing with an imbalanced classes.

Choosing the training function which will perform fastest for the specific problem depends on several factors including the complexity of the problem, number of connections in the network,

number of training samples, the performance function used and whether problem we are solving is the regression or the classification [14].

For choosing the set of training function to test them further several sources were used including [21] and [30]. Bases on the recommendations and best practices 4 training functions were selected. These functions have shown fastest convergence rates and performance improvements for specifically classification problems. Computation power requirements were also considered during compiling this selection.

Training functions which were implemented and tested, include:

- **Scaled conjugate gradient backpropagation (SCG) –** is a supervised training algorithm which is based on a class of optimization technique called conjugate gradient. Unlike the basic gradient approach it does not depend on manually setting hyper parameters, like for example learning rate, which are crucial for good performance and can be set inefficiently when done manually. This method uses the second order derivative information from the error surface but only requires $O(n)$ memory, where n is the number of connections in the network. This approach does not include user dependent parameters and also avoids the time consuming line search at each iteration, which is often used by other second order training functions. This function is introduced and well-studied in the paper by Martin Meiller [32] where the author shows that SCG learning algorithm is more effective when compared to the standard backpropagation algorithm and is faster by at least one order of magnitude.
  In other source [14], the SCG algorithm has performed well in tests when trained with large networks and for both, function fitting and classification purposes. It has shown low memory requirements while still performing considerably faster than an ordinary gradient descent approach when compared on identical networks and training sets.
  In Matlab environment SCG training method is implemented by using an inbuilt function called `'trainscg'` and it is set as a training function for the network `'net.trainFcn'`. [14]

- **Conjugate gradient backpropagation with Powell-Beale restarts (CGB) –** is the conjugate gradient algorithm which does not require the calculation of the second order derivatives which is represented by calculating costly Hessian matrix, and yet it maintains the quadratic convergence property. Important properties of the CGB algorithm include the fact that on each iteration it tries to find the descent direction

which will improve on the direction found on the previous iteration. It uses a line search approach and it only works on batch training mode [15]. The rate of convergence of the algorithm is linear unless the iterative process is occasionally restarted. An optimal and promising restart approach was suggested by M. J. D. Powell in his study [33]. Like other conjugate gradient methods CGB requires quite modest memory requirements during training and has shown good convergence rates for classification problems. [14]

In Matlab environment CGB training method is implemented by using an inbuilt function called `'traincgb'` and it is set as a training function for the network `'net.trainFcn'`. [14]

- **Resilient backpropagation (RP) –** is the training algorithm which was designed in order to overcome the disadvantages of the basic gradient descent optimization. It performs a local adaptation of the connection weights based on the behavior of the performance function. This algorithm's performance, unlike other adaptive techniques, does not degrade when the error derivative gets small, which gives it the promising capabilities. Different to other algorithms, only the sign and not the magnitude of the partial error derivative is used to perform the update of the weights. This algorithm was introduced, tested and proved to be robust in the study by M. Riedmiller and H. Braun [34].

  RP training function has performed as the fastest algorithm for pattern recognition problems even with low memory requirements, but the it has also been shown that the performance degrades when the goal error value is reduced too low [14].
  In Matlab environment RP training method is implemented by using an inbuilt function called `'trainrp'` and it is set as a training function for the network `'net.trainFcn'`. [14]

- **Bayesian regularization backpropagation (BR) –** is a neural network training function that updates the values of the network connection weights based on a Levenberg-Marquardt optimization. It strictly uses the mean squared error (MSE) performance function during the training, since it uses the Jacobian matrix for calculations. Using the best linear combination of weights, it trains the network that generalizes well. One feature of this approach is that it provides information of how many network weights are used effectively. BR algorithm works best when the training and target data is scaled. It is discussed in details in the study [35] and [36].

But the good generalization capabilities of this network come at a cost. Computational requirements are higher when compared to other training methods, since it requires computation of the Hessian matrix, which results in longer training time [37]. In Matlab environment BR training method is implemented by using an inbuilt function called `'trainbr'` and it is set as a training function for the network `'net.trainFcn'`. [14]

These training functions were implemented and tested in depth. The results of performance evaluation are discussed in Chapter 6.

Since the computational requirements for each approach is different they use different amount of memory and time during the training step. Although, these requirements do not directly correlate with the performance of each. The average training iteration time in seconds and number of iterations was recorded for each of them and can be seen on the Table 5.1. This table shows how many seconds it took for the training function to finish one training iteration and also average number of iterations to finish the training. As it was mentioned before, Bayesian regularization backpropagation (BR) has shown the highest training iteration time compared to other methods. This observation helps to understand the time and computation cost associated with each of the suggested training functions. The time measuring test was conducted on single thread process on Intel® Core™ i7-4700MQ Processor with 16 GB of Random-access memory (RAM).

Table 5.2. Average training time in seconds for each training function.

Training function	Average iteration time, s	Average number of iterations	Average training time, s
RP	0,2016	250	50,4
SCG	0,3849	200	76,98
Autoencoder + SCG	0,476	1000	476
CGB	0,8927	200	178,54
BR	2,1787	250	544,675

5.2.3. Validation checks and other stopping criterions

The next step in tuning the training procedure is to decide when to stop the training process. There are several criterions which can be used as a sign to terminate the training, since the network has reached the most optimal or required performance state. It should be noted that this parameters can be set for each training functions listed above. In this Chapter this criterions will be discussed and compared.

One of the stopping criteria is the training time. The algorithm can be stopped whenever the time set by the user is elapsed. Since the training time was not the limitation for testing purposes on this particular project, it was not used as a stopping factor for the algorithm.

Another parameter, which can be used, is a minimum performance value or the goal of the training. This parameter controls for how low error value can be accepted as enough for the final model. Value of 0 was used in order to acquire the best approximation possible.

Number of iterations or epochs for the training functions was set to default value of 1000. Problem with this parameter is that it is difficult to know how many iterations would yield the best results [30]. Using any larger value, during testing, did not provide any performance benefit since other topping criterions usually ended the training before this limit was reached.

Minimum gradient is the criterion which stops the training when the partial derivative of the performance function reaches values near to 0 and training becomes inefficient and clos to optimal. For the testing purposes this parameter was set to different values in the range of $[10^{-7}, 10^{-5}]$.

The most important stopping criterion, which stops the training when the network has reached the best generalization and performance balance, is called validation check or early stopping [21]. Much like the regularization parameter, this method is aimed on improving the network generalization ability and avoid overfitting. The idea behind this approach is that as training progresses the network can start to overfit, meaning that the complexity of the network increases so that it fits the training set but fails to generalize on the validation set. In order to monitor the state of the network, a method called cross-validation is used, which uses the validation set discussed in Chapter 3.4.

During a typical training procedure the error measured on the training set decreases with number of iterations. However, the validation set error often shows decrease at first, followed by an increase, indicating that the network starts to overfit. When this indicator is still present

during several iterations it can be assumed that the network has reached the optimal point and training can be stopped. The parameters with lowest validation set error are chosen as the best solution and are used as the network connection weights. The number of iterations during which the validation increase is monitored is the stopping parameter called number of validation checks. [30]

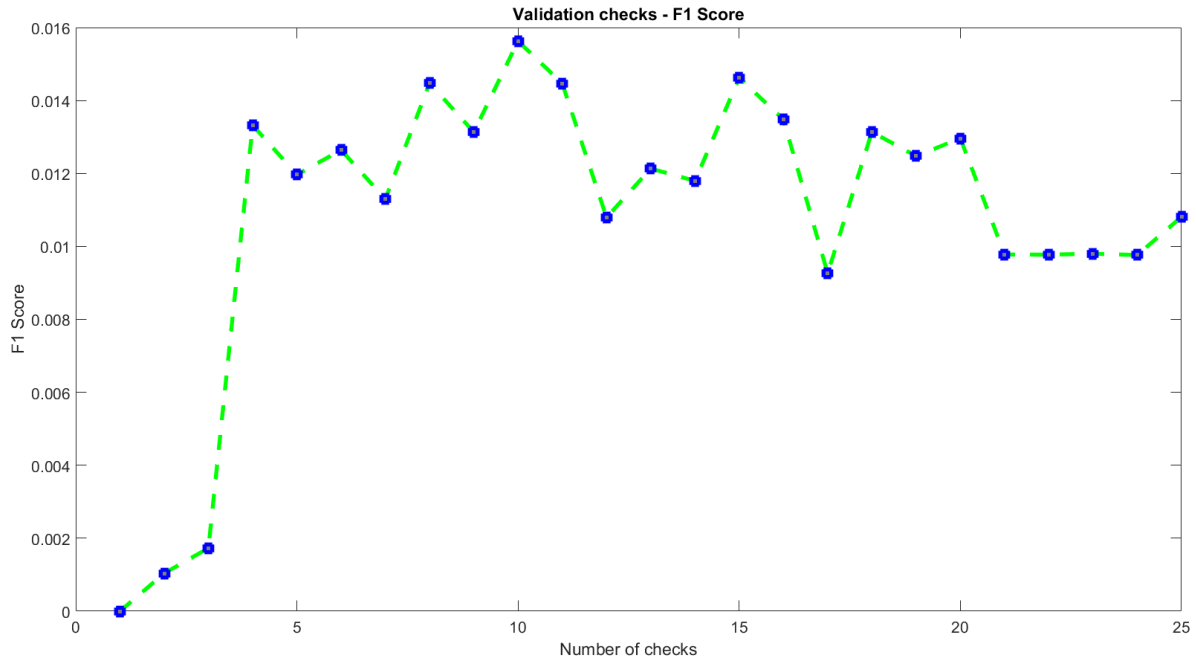


Figure 5.2. Number of validation checks over F_1 score (higher the better)

In order to choose the best number of validation checks, each setting from 0 to 25 was empirically tested. The performance of each setting was measured with the F_1 score and the results are shown on Figure 5.2. As the data shows using a number higher than 5 generally results in better generalization and the best performance was achieved when using number of checks at 10. This setting was used for further testing and evaluation of the network. More details about the F_1 score evaluation metric are discussed in Chapter 6.

In Matlab environment this parameter is set by updating `'net.trainParam.max_fail' property.`

5.3. Parallel computing modes for the network training

Another approach which can improve the speed and decrease the time need for the network training is using the parallel nature of the artificial neural networks. Although it should be noted that this approach does not change the overall performance of the model and the classification does not change in any way. In this part the parallel training approaches will be discussed and compared.

By default the Matlab Neural Network toolbox uses the single thread of the Central Processing Unit (CPU), which can be a limiting factor for the batch training approach. By using the Matlab Parallel Computing Toolbox in conjunction with Neural Network Toolbox the training and simulation steps can take an advantage of parallelism modes [14]. This computing modes were accessed by specifying the `'train'` function call parameters `'useParallel'` and `'useGPU'` in Matlab environment.

First mode which was considered is the distributing computing mode. This approach can be used on multicore CPUs, which was used for the test, or on multiple computers in the cluster. Underneath the hood by default toolbox splits the training data equally between the threads from the pool and each thread performs the backpropagation steps independently.

The second mode supporting the backpropagation training, which can be implemented using the Parallel Computing Toolbox, uses the graphical processing unit (GPU) of the computer. This computation mode is promising since the GPUs are flexible enough to perform numerical computations and are showing good improvements for the problems which can be solved in parallel threads [14]. It should be noted that only gradient training methods are supported and Jacobian training methods can't be used when using this mode. Also the computing mode called distributed GPU was considered. This approach uses both CPU and/or GPU in parallel.

The benefits of these approaches strongly depend on the computing capabilities of the machine on which the testing was performed on and even better results can be achieved with higher performance components. For this specific test setup a personal computer with the CPU: Intel® Core™ i7-4700MQ Processor, GPU: GeForce GT 740M and Random-access memory (RAM) of 16 GB was used.

Training function for the test was set to Scaled Conjugate Gradient with 12 neurons in the hidden layer, with number of validation checks set to 10, regularization parameter set to 0 and cross-entropy used as a performance function.

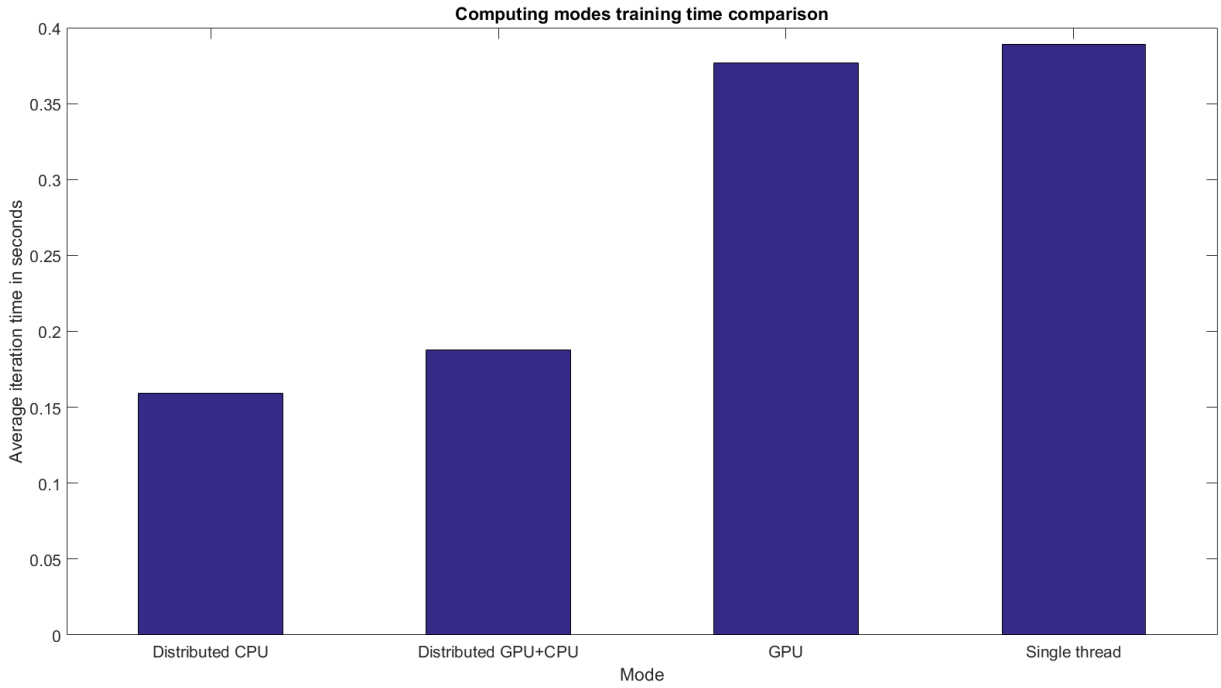


Figure 5.3. Average training iteration times in seconds for several computing modes (lower the better)

Each computational mode was selected and performed 3 times. Average time of the single iteration in seconds was taken and compared. Results are shown on Figure 5.3.

As it can be seen on the plot, using the default single thread for training shows the highest time of 0,389 seconds per iteration. Next highest average iteration time was set by the GPU equal to 0,377 seconds. Next comes the distributed training using both CPU and GPU showing result of 0,188 seconds per iteration, which is a good improvement. However the best result was shown by using the parallel threads of the CPU and the average training iteration time was 0,159 seconds which compared with default single thread time of 0,389 seconds is an improvement of almost 60% (59,126% to be more precise).

6. MODEL EVALUATION AND FINE-TUNING

The next step in a typical workflow of neural network design is to evaluate the trained classifiers with appropriate metrics. This step is important for understanding how well they perform and which parameters to tune further in order to improve the results. After the training is done all the discussed network architectures and training functions undergo an evaluation step.

In this chapter effective evaluation methods for classifiers will be discussed, applied and compared. This evaluation metrics will be used for comparing the training models between each other and also comparing their performance to an existing, non-neural network approach. Next the classifiers will be fine-tuned for optimal performance and the best one will be identified. Lastly network will be prepared for deployment after which it can be easily implemented in software product and used in practice.

6.1. Imbalanced dataset classifier evaluation metrics

On the evaluation step it is important to acknowledge that the provided training set is inherently imbalanced, holding 97,863% samples of the policies without filled insurance claims, which was mentioned in Chapter 3, meaning that samples are not equally represented. Since the classification categories are not represented equally it is not a recommended practice to use predictive accuracy as a performance evaluation metric, since it will not give a meaningful comparison for the classifiers [38]. Accuracy can be measured by the ratio of successfully classified samples over the number of total number of samples.

In order to see why the accuracy is not a recommended evaluation criterion it can be explained by the example of totally trivial classifier which would classify every input sample as a safe driver. This classifier would get 97,863% accuracy on the training set, only because this class is dominant in the training set, which is not a good performance indicator at all and can be misleading.

Luckily problem of imbalanced dataset classification is well studied and happens quite often when dealing with real life problems. In this chapter recommended approaches for this particular problem will be listed and discussed. This metrics will then be applied to the training functions discussed in Chapter 5.2. Results of this evaluation will be presented by the end of this chapter for choosing the best option.

6.1.1. Confusion matrix

A common way to visualize and present the performance of the binary classifier is to use a confusion matrix. This matrix holds the results produced by the classifier in organized way and it makes easy to see how the system performs [38].

In order to build this type of matrix particularly for the binary classification problem a table with 2 rows and 2 columns should be constructed. Rows represent the predicted classes and columns – the target or actual classes. The model of the confusion matrix can be seen on Table 6.1. Each cell of this table contains the number of predictions made by a classifier that fall into one of these cells. Here is what they represent in case of the discussed problem:

- **True positives (TP)** - number of correctly identified dangerous samples. Samples which were predicted to have an accident and they really had according to the dataset. This number goes to the top left corner of the 2 by 2 matrix.
- **True negatives (TN)** – number of correctly identified safe samples. Meaning samples which were predicted not to have an accident and actually did not have according to the dataset. This number is stored in the bottom right corner of the matrix.
- **False positives (FP)** – number of incorrectly classified safe samples. Meaning samples which were predicted to have an accident in reality did not have it. This is also called type 1 error. This number is stored in top right corner of the confusion matrix.
- **False negatives (FN)** – number of incorrectly classified risky samples. Meaning samples which were predicted not to have an accident but in reality actually had one. This is also called type 2 error. This number is stored in bottom left corner of the matrix.

Table 6.1. A model of the confusion matrix [38].

	Actual positive	Actual negative
Predicted positive	TP	FP
Predicted negative	FN	TN

A confusion matrix produced by a good classifier should maximize the numbers located on the main diagonal of the confusion matrix. Meaning maximizing the number of true positives and true negatives while minimizing the numbers on the non-main diagonal, particularly false positives and false negatives [38].

6.1.2. Precision and Recall

There is a lot of useful information which can be extracted from the confusion matrix. One of the most common and applicable are called Precision and Recall. Both of this metrics are used in order to understand and measure the relevance of the classifier [38].

Precision is calculated using the fraction shown in Equation 6.1. This fraction gets the values between the 0 and 1, 0 meaning that not a single instance was classified as true positive and 1 meaning a highest precision classifier, which has not produced any false positives. Getting a low precision value is undesirable, since in the case of this problem it means falsely accusing safe drivers that they will get into an accident.

Recall is calculated using the fraction shown in Equation 6.2. This fraction also gets values between 0 and 1, where again 0 means that not a single true positive was identified and 1 meaning a highest recall classifier, which has not produced a false negative classification. Getting a low recall value is also undesirable, since in the case of this problem it means identifying driver as safe when in fact they got into an accident.

Main goal of good classification is to improve the recall without hurting the precision. Although this two goals can be often conflicting, since when increasing the rate of true positives for the dangerous driver's class, the number of false positives can also increase which is undesirable [38]. This means the optimal balance between the two should be found.

Equation 6.1. Precision metric [38]

$$\mathrm{P} = \frac{TP}{TP+FP}, \tag{6.1}$$

Where P– Precision of the classifier within the range of [0, 1],

TP – Number of true positives,

FP– Number of false positives

Equation 6.2. Recall metric [38]

$$\mathrm{R} = \frac{TP}{TP+FN}, \tag{6.2}$$

Where R– Recall of the classifier within the range of [0, 1],

TP – Number of true positives,

FN– Number of false negatives

6.1.3. F score

F score is commonly used for measuring classifier performance. It considers the balance of both precision P and recall R to compute a single score for evaluation. It is used to balance the tradeoffs between these two terms and the benefit is it can be tuned to give more weight to one or another [38]. F score was derived and studied in details in the work of C. J. van Rijsbergen [39].

The equation for computing the F score is given in Equation 6.3. It returns the values in the range of [0, 1] where higher value means better performance. We can see that in addition to the precision and recall terms, this score also uses β term, which is the positive real value representing the balance and priority between the two.

Most commonly used values for β are: 0,5 during which the final score is more dependent on the precision of the classifier, meaning that precision has higher priority. 1 during which the score represents a harmonic mean, or the balance mean between the precision and recall and 2 which puts higher priority and emphasis on the recall.

Using $F_{0,5}$ score for evaluation will result in classifier which has higher precision and will produce less false accusations. On the other hand F_2 will result in classifier with higher recall and will identify more dangerous customers for the company.

All three scores were measured for each classifier. Decision for which one to choose for real life applications is a business decision depending on how “strict” classifier is needed and which precision or recall is more important for the company. This decision has to be done by an insurance company and it is outside the scope of the research. All three results will be presented and it is an easy procedure to apply this measures to get the final result.

Equation 6.3. F score metric [38]

$$\mathrm{F}_{\beta} = (1 + \beta^2) * \frac{P*R}{\beta^2*P+R}, \tag{6.3}$$

Where F– F score within the range of [0, 1],

β– Positive real value representing relative weight between precision and recall,

P – Precision of the classifier,

R– Recall of the classifier

Balanced F_1 score have been used as a main assessment tool throughout the research when choosing optimal parameters for the network and training functions.

6.1.4. Receiver operating characteristic

The receiver operating characteristic (ROC) curves are commonly used plots for summarizing the performance of the binary classifier over a range of discrimination threshold between true positive rates, which are computed in the same way as the recall, on y-axis and false positive rates, which are computed using the Equation 6.4., on the x-axis.

Equation 6.4. False positive rate (FPR) [38]

$$FPR = \frac{\mathrm{FP}}{FP+TN}, \tag{6.4}$$

Where FPR– False positive rate within the range of [0, 1],

FP– Number of false positive classifications,

TN– Number of true negative classifications

Accepted performance metric for the ROC curve is are under the curve. Bigger area means better classification performance. An ideal point on this curve is a point (0,1) which means that all the positive examples were classified correctly and no negative examples were misclassified as positive [38]. ROC curves as a performance metric is studied in details in [40].

6.2. Applying evaluation metrics to classifiers

The metrics, which were discussed, were applied to each training algorithm and network architecture after the networks reached the stopping point of the training step. Evaluation was done on the output matrix of the network, which was discussed earlier in Chapter 4.1.3. The discrimination threshold for classification during this evaluation was default value of 0,5. This means that one of the output probability of the network should be more than 0,5 for the sample to be classified as a safe or dangerous customer. For this evaluation all the data was used including training, evaluation and training sets. Although these results vary depending on weight initialization of the network in the beginning of the training, they do not affect the results dramatically and the final results were the same during several tests. The results of the evaluation for each approach are listed on Table 6.2.

Already from this table it can be seen that the Bayesian Regularization Backpropagation (BR) has shown the best performance out of the selected algorithms in classifying the customers. It has shown the highest results in all performance metrics. But this is not enough to give final conclusions. Still more thorough tests were done to correctly measure the training functions and their performances.

All graphical results for receiver operating characteristic curves for each of this approaches can be found in appendix section of this work. As it can be seen from the areas under the ROC curves all the training functions have found a positive correlation and were successful in classification.

Table 6.2. Evaluation of classifiers with the discrimination threshold value of 0,5

Training function	**Precision**	**Recall**	$F_{0,5}$	F_1	F_2
RP	0,5607	0,0052	0,025	0,0103	0,0065
SCG	0,4017	0,008	0,0369	0,0156	0,0099
Autoencoder + SCG	0,5789	0,0038	0,0186	0,0076	0,0048
CGB	0,463	0,0065	0,0307	0,0128	0,0081
BR	0,6714	0,0081	0,0388	0,0161	0,0101

The confusion matrix for the BR training function can be seen on Figure 6.1. As it can be seen on the matrix the classifier have identified majority of save drivers and some of the dangerous drivers with the precision value of 67,1%. Although, as we will see in the following chapter its identification capabilities can be improved by adjusting the threshold of classification. Confusion matrixes for other training methods can be seen in the Appendix section of this work.

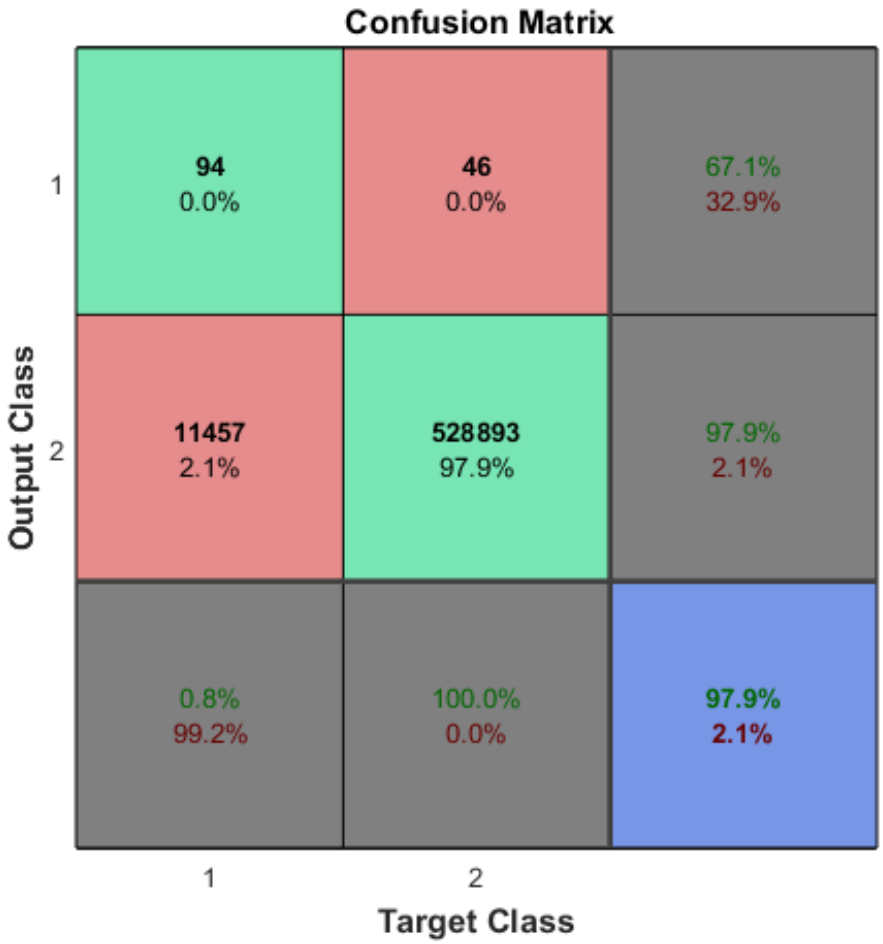


Figure 6.1. Bayesian Regularization backpropagation confusion matrix with the discrimination threshold of 0,5.

6.3. Fine tuning the classifiers and comparing to an existing solution

Based on the ROC curves, which give information about classifier performance with different threshold values between 0 and 1, it can be assumed that the performance of the classifier can be improved by correctly setting the threshold value for the output probabilities of the network.

This was tested by taking the default threshold value of 0,5 and iteratively setting every value up until 1 with the step size of 0,001. Than all the performance metrics like precision, recall and f score were plotted in regards to the new threshold value.

Setting higher threshold values means that the classifier should give higher probabilities in order to classify samples into dangerous and safe ones. This threshold adjustment probing will give the value of probability threshold, when used will get the optimal classification capabilities based on performance metrics.

For the performance plots 3 best performing algorithms were chosen: Bayesian Regularization (BR), Autoencoder with Scaled Conjugate Gradient training and Feed forward network with Scaled Conjugate Gradient training.

For the comparison with methods, which are already in use in industry, Bonus Malus classification was taken from the dataset. This is the customer classification method used in insurance companies which is derived from the driving history of the customer. This categorization consists of 15 categories and higher category means higher risk driver. Same performance metrics of precision, recall and F score were applied to this categorization method in order to see how the neural network approach compares to it. This test gives a visualization if the neural network approach have improved on an existing categorization technique.

The values of precision and recall of the classifiers over the threshold value can be seen on Figure 6.2., and Figure 6.3. As it can be seen of the plots all 3 neural networks have reached higher precision when compared to an existing solution. Out of this 3 the Bayesian Regularization shows better results when compared to others. It reached 0,6714 precision value while the Autoencoder network achieved 0,5789 and the Scaled Conjugate Gradient achieved 0,4017.

On the recall part we can see that the value is quite low until the threshold value exceeds the 0,91 mark after which the recall for the neural network classifiers starts to grow and overcomes the existing approach.

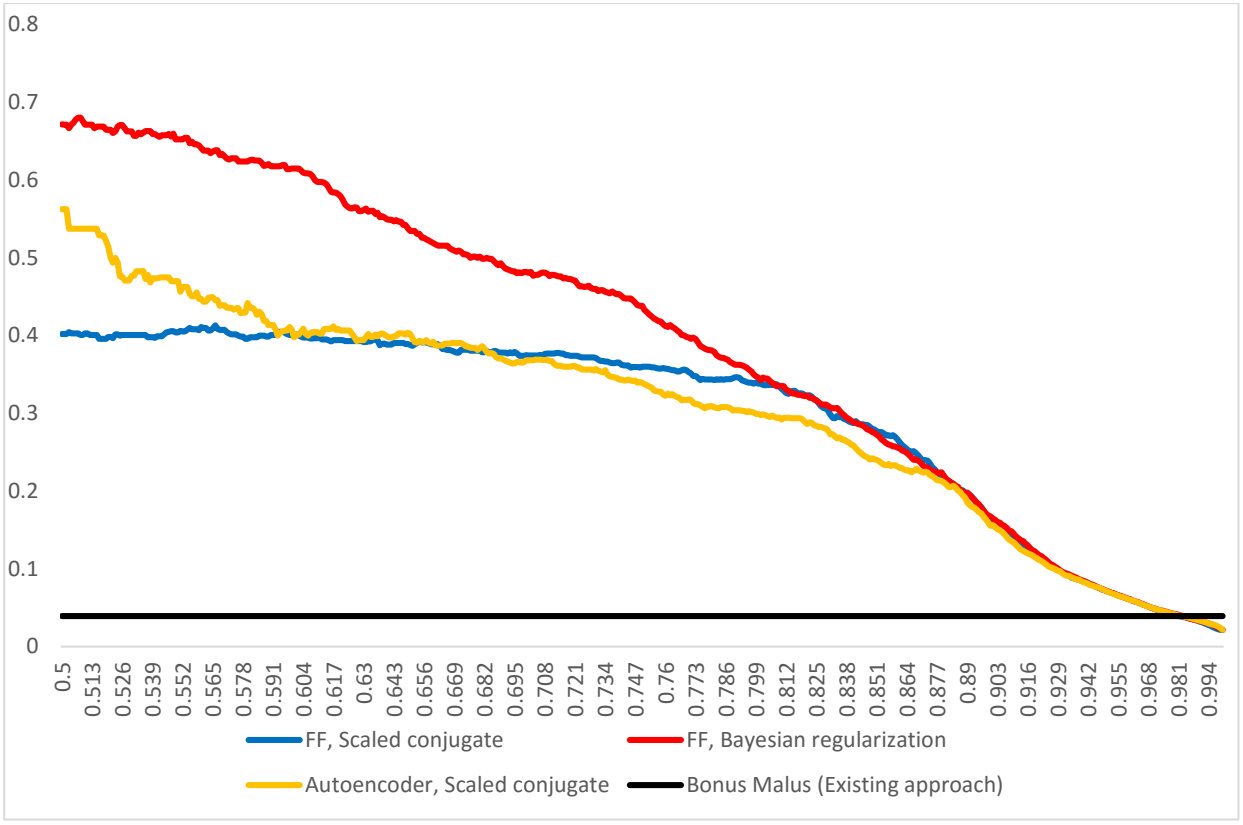


Figure 6.2. Precision values of neural networks over the discrimination threshold.

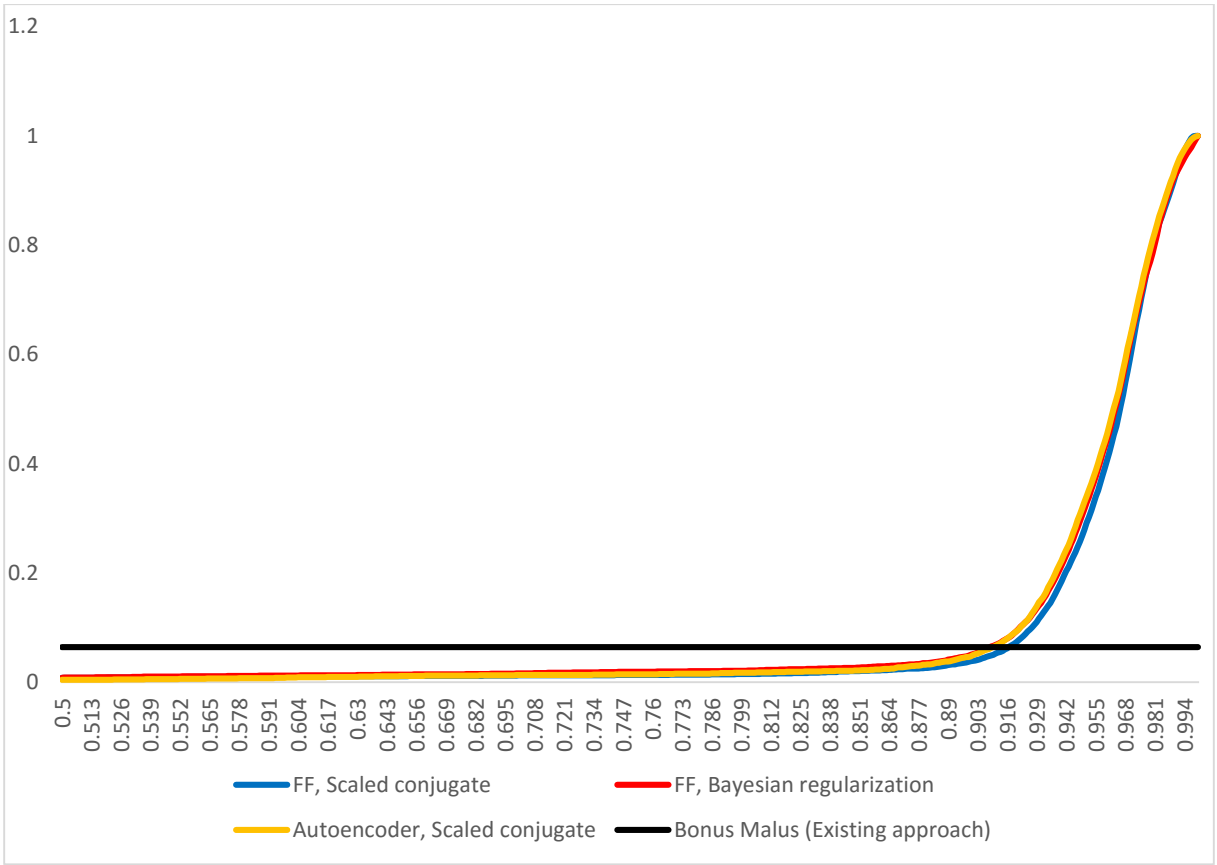


Figure 6.3. Recall values of neural networks over the discrimination threshold.

But as it was shown in Chapter 6.1.3., in order to choose the best threshold value the balanced measure of precision and recall is more informative and useful than using them separately. So the measures of $F_{0,5}, F_1$ and F_2 were also taken for each training algorithm on the threshold region of [0,5, 1] with the step size of 0,001.

6.3.1. Optimal threshold value based on $F_{0,5}$ score

The results of evaluation with $F_{0,5}$ measure are shown on Figure 6.3. From this plot we can see that Bayesian regularization, represented with red line, has shown a best highest score of the three neural networks. At the threshold value of 0,909 it has reached $F_{0,5}$ score of 0,1183. The confusion matrix for this threshold value is shown on Figure 6.4.

Second highest result was shown by an Autoencoder network. At the threshold value of 0,906 it reached the maximal $F_{0,5}$ score of 0,116.

Third result was shown by the feed-forward network with Scaled Conjugate Gradient algorithm. It reached its maximum score at the threshold value of 0,925, with the value of 0,1036.

All three approaches have shown much better result when compared to an existing solution which only reached the value of 0,0426.

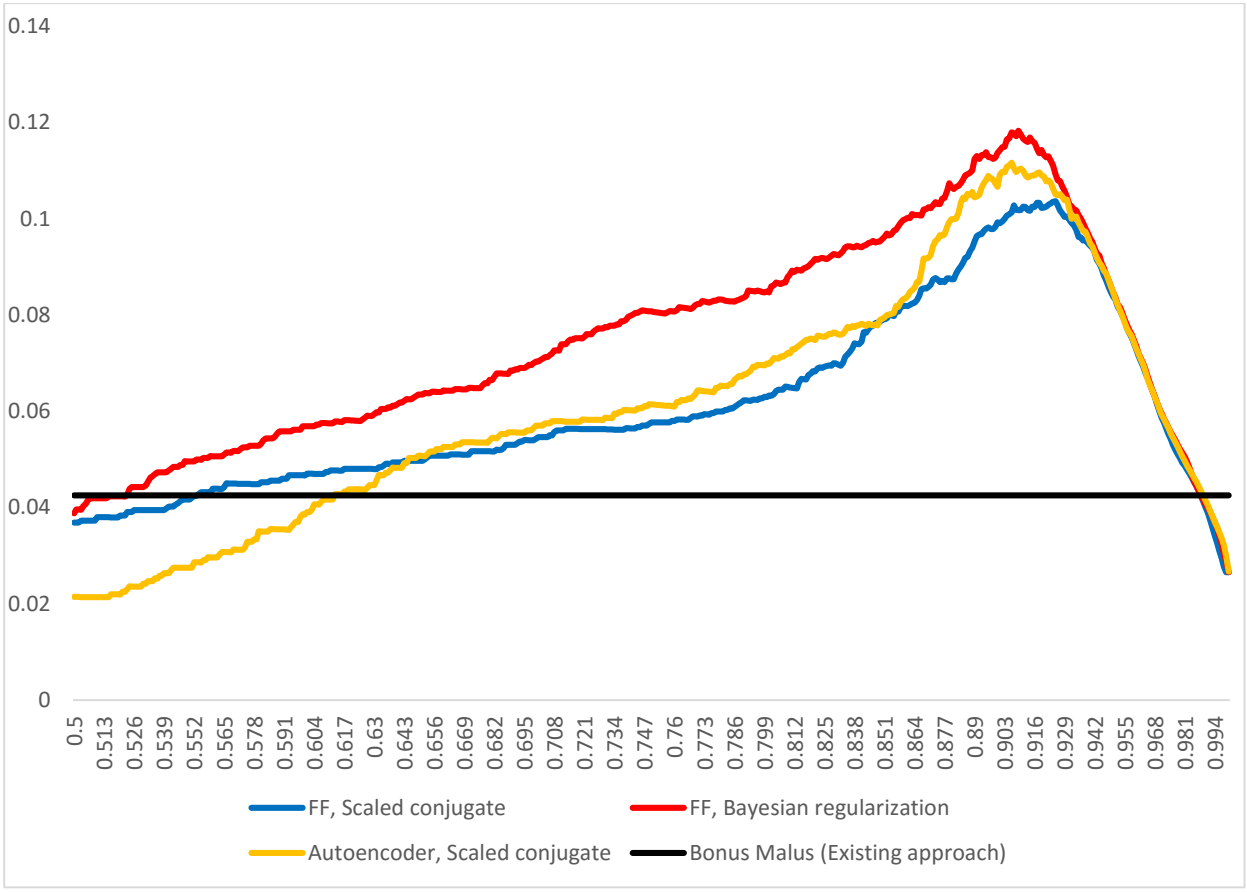


Figure 6.3. $F_{0,5}$ Score values of neural networks over the discrimination threshold.

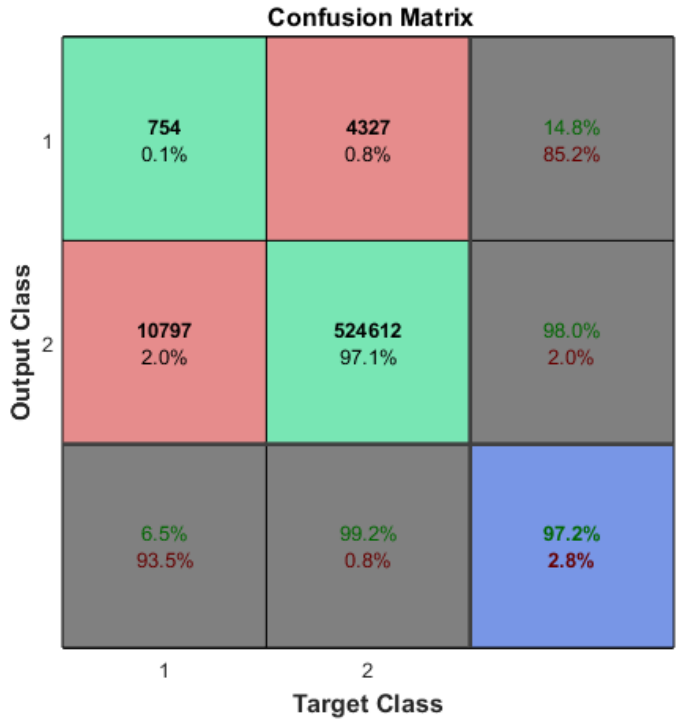


Figure 6.4. Bayesian Regularization confusion matrix with optimal $F_{0,5}$ threshold of 0,909

6.3.2. Optimal threshold value based on F_1 score

The same test was conducted for the balanced F_1 score. The results of this test are shown on Figure 6.5. From this plot we can see that again the Bayesian regularization has shown highest score. At the threshold value of 0,941 it reached the F_1 score of 0,1214. The confusion matrix at this threshold value is shown on Figure 6.6. Same high score was reached by an Autoencoder network. At the threshold value of 0,938 it reached the maximal F_1 score of same 0,1214.

The third result of 0,1168 was reached by the Scaled Conjugate Gradient method at the threshold value of 0,942. Again all three networks performed better when compared to an existing solution, which reached the score of 0,0572.

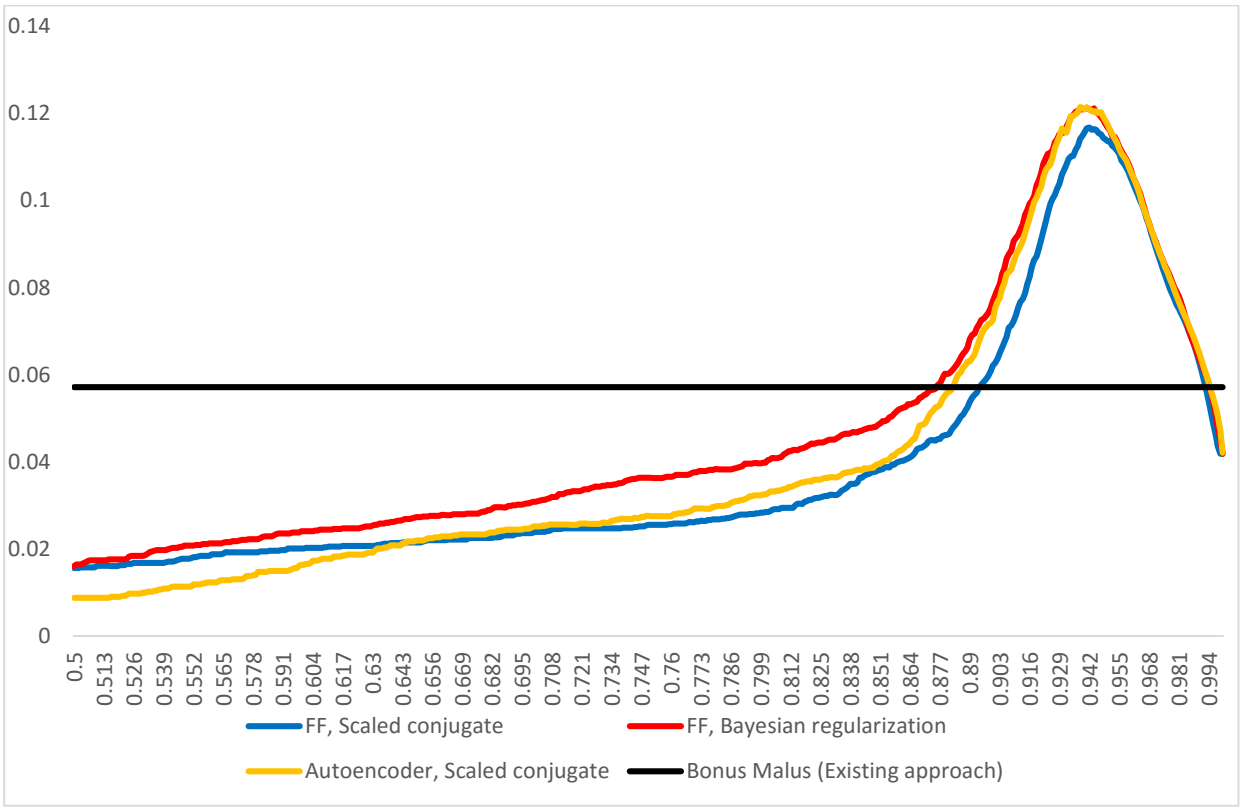


Figure 6.5. F_1 Score values of neural networks over the discrimination threshold.

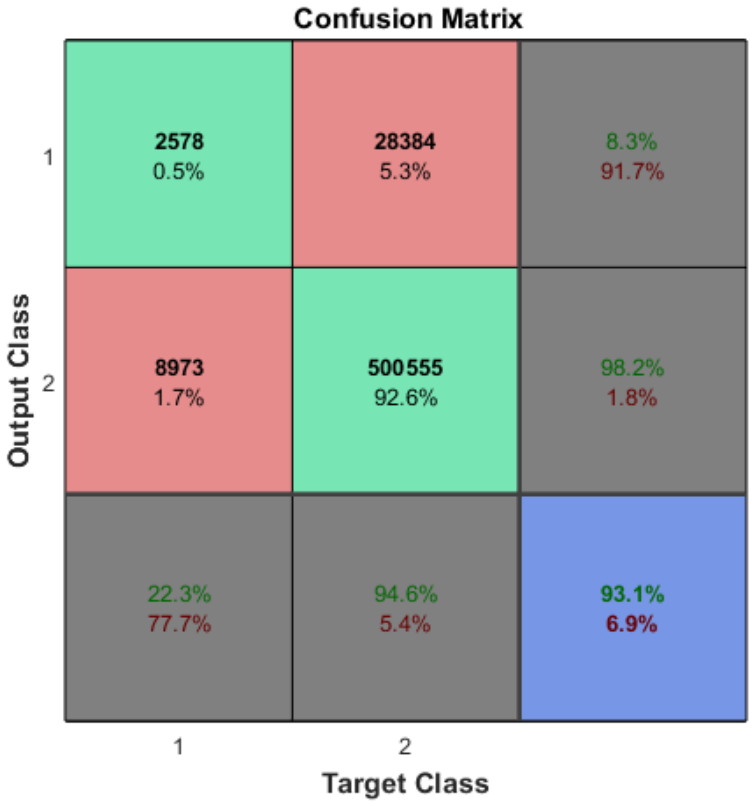


Figure 6.6. Bayesian Regularization confusion matrix with optimal F_1 threshold of 0,941

6.3.3. Optimal threshold value based on F_2 score

The same test was conducted for the F_2 score. The results of this test are shown on Figure 6.7. From this test we can see that the Bayesian regularization has shown second highest score. At the threshold value of 0,959 it reached the F_2 score of 0,1969. The confusion matrix at this threshold value is shown on Figure 6.8. Highest score was reached by an Autoencoder network. At the threshold value of 0,958 it reached the maximal F_2 score of 0,1977.

The third result of 0,1899 was reached by the Scaled Conjugate Gradient method at the threshold value of 0,963. Again all three networks performed better when compared to an existing solution, which reached the score of 0,1158.

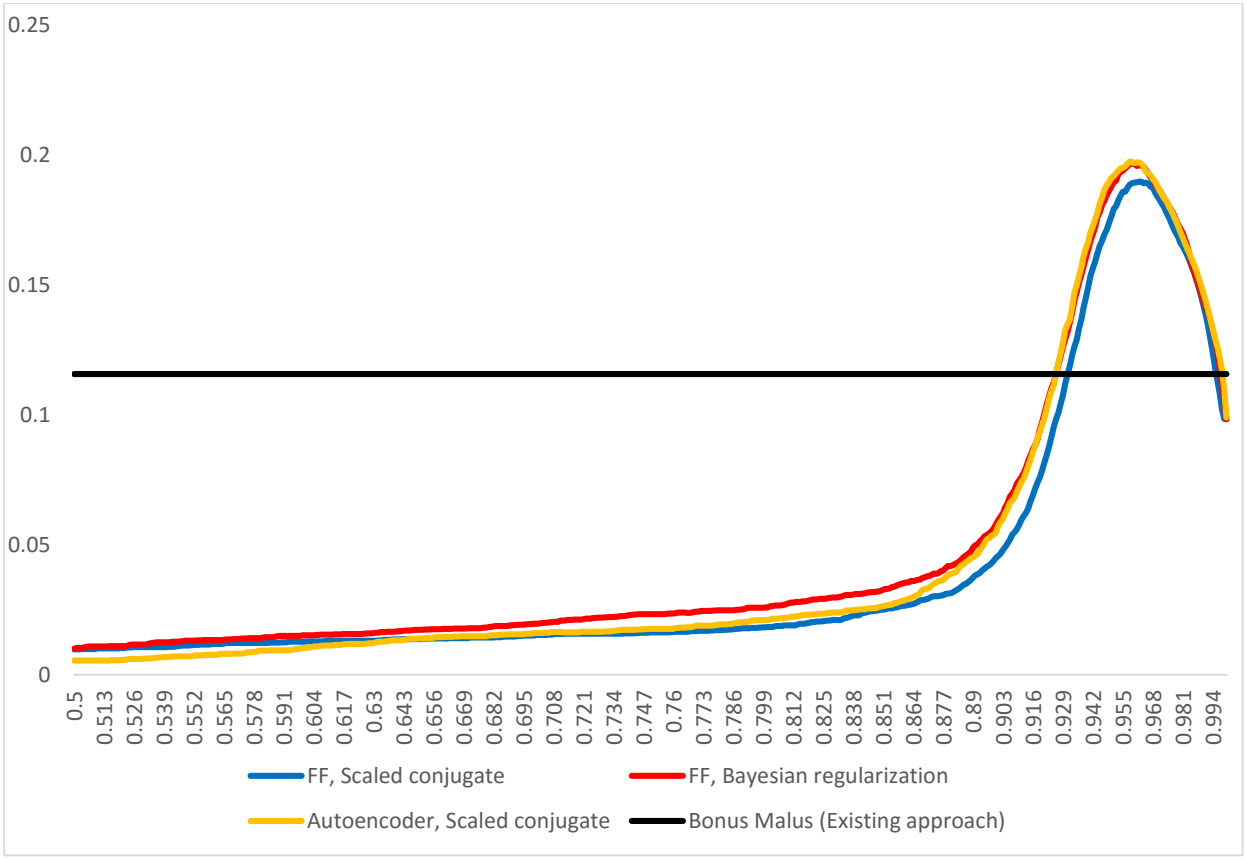


Figure 6.7. F_2 Score values of neural networks over the discrimination threshold.

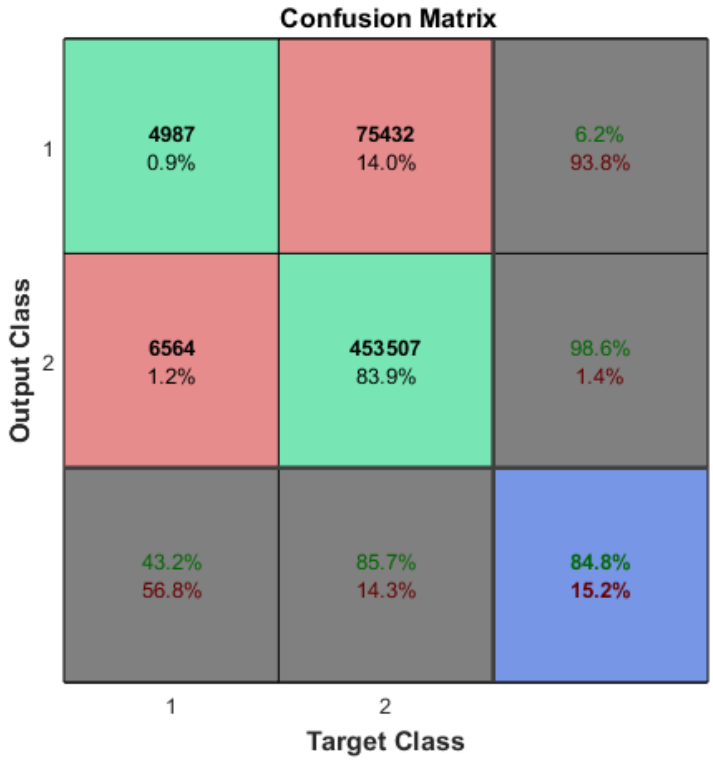


Figure 6.8. Bayesian Regularization confusion matrix with optimal F_2 threshold of 0,959

6.4 Network deployment

After the networks were tested, evaluated and compared the next step is to deploy them as a usable functions. This deployment should be done in a way to be easily to implementable for software, which want to use it further.

Since Feed-forward network with Bayesian Regularization and Autoencoder network with Scaled Conjugate Gradient method has shown the best results during the evaluation step, they were the chosen classifiers to deploy.

Deployment was done by generating a function in Matlab environment, which would accept as an input the customer feature vector with length of 11, which was laid out in Table 4.1. As a return value, function returns the output of the neural network, which are probabilities p_1 and p_2. First denotes the probability of the feature vector belonging to dangerous customer class and the second one denotes the probability of the feature vector belonging to the safe customer class. The thresholding has to be done additionally with the recommended threshold values explained in Chapter 6.3.

Name of the functions are `‘BayesianNeuralNetworkFunction’` and `‘AutoencoderNeuralNetworkFunction’` and they are ready to be used and tested in real life scenarios.

The functions were generated by using an inbuilt Matlab method ‘getFunction’. It takes as an argument a neural network object and outputs a function with the said properties. The functions were generated so they contain the weight and bias matrixes of the networks, which are used to calculate the output of the network. This decision was made since it will be simpler to implement in any programming language desired, since the only thing required to calculate the output is to perform a basic matrix operations like addition, multiplication and exponentiation. This means that this functions can be easily ported to mobile, desktop or web application.

CONCLUSION

The evaluation step has shown that two neural networks has performed the best and shown the optimal classification of the dangerous and safe drivers. Both this neural networks have shown better results when compared to the previous approach.

One of them, which has shown best results in majority of metrics, is a feedforward neural network architecture with 11 neurons at the input layer, 12 neurons at the single hidden layer and with 2 neurons at the output layer. It uses hyperbolic tangent as the activation function and Softmax as the output layer activation function. This network was trained with a Bayesian Regularization Backpropagation training method with the Mean Squared Error performance function using the 70% of the dataset as a training data, 15% as a validation data and 15% as testing data. This network output resulted in highest $F_{0,5}$ score of 0,1183 at the discrimination threshold value of 0,909, highest F_1 score of 0,1214 at the threshold value of 0,941 and highest F_2 score of 0,1969 at the threshold value of 0,959.

Another neural network which performed well, especially in cases when higher recall value was evaluated is an Autoencoder neural network with 2 encoder layers, with 10 neurons in each and 2 neurons at the output layer. Encoder layers used sigmoid function as an activation function and output layer used the Softmax activation function. Network was trained with Scaled Conjugate Gradient Backpropagation method with the cross-entropy as a performance function using the 70% of the dataset as the training data, 15% as the validation data and 15% as testing data. This network output resulted in highest $F_{0,5}$ score of 0,116 at the discrimination threshold value of 0,906, highest F_1 score of 0,1214 at the threshold value of 0,938 and highest F_2 score of 0,1977 at the threshold value of 0,958.

SUMMARY

The main motivation for this study came from the insurance company and its need classify their vehicle insurance customers by the probability of them getting into an accident and filling the claim. Correct classification can help the insurance company to give fairer prices for the customers and make better forecasts about the risks associated with vehicle insurance contracts. Previous approaches did not use any type of intelligent or adaptive classifiers for achieving this and relied completely on just using the customer driving history as a metric. Out of this came the idea that classification can be improved by using the data company has stored throughout the period in their database. This database holds different types of information about customer and about the vehicles used and their accident history.

The dataset consisting of half a million insurance policy samples was provided for further studies and analysis. In order to extract meaningful and beneficial knowledge from the database the approach of artificial neural networks was chosen as theoretically it can make good classification models when the data is high dimensional, nonlinear and noisy. This approach also have been backed up with number of studies, giving neural networks a promising performance possibilities.

First step before training the neural network was to study and prepare the dataset. Firstly the features, which could be used for training and feature, which would be used as a target data was separated. One of the fields in the database, called “number of claims”, was chosen as a measure of risk for each customer. For the majority of the samples, around 97%, there were no instances of filled claims making them safe, or desirable customer. For the rest of the samples the values varied from 1 to 4. This was changed to a binary feature, storing information whether given customer had an accident or he did not. For the training dataset numerical features were normalized and scaled, since this improves the convergence rate for the network training algorithm and results in a better classifier. Categorical features were represented in the database as identification numbers which had no particular ordering and meaning, so they were replaced with the risk factor, a positive real number between 0 and 1 showing how high is the risk associated with the particular categorical feature. Next step was to conduct a feature selection. This step was done in order to get rid of redundant features, which provided no benefit for classification performance. Only 11 highest correlative features were finally extracted and used as the training, validation and testing sets during the training.

On the next step neural network architectures were considered. The best architectures were considered by studying their properties and two architectures were chosen to implement and compare. Feed-forward neural network with hidden layers and Autoencoder neural network. By using best practices, hyperbolic tangent and sigmoid transfer functions were chosen for hidden layers of the networks. Softmax activation function was chosen as an output layer activation function. By conducting performance tests, optimal number of layers and number of neurons in each of them were chosen. The resulted setup for feed-forward network was to use 11 neurons on the input layer, 12 neurons at the hidden layer and 2 neurons at the output. The last layer would output the probabilities of the input belonging to safe or to dangerous customer class. As for the Autoencoder network an optimal architecture was to use 11 neurons at the input layer, then 2 encoder layers, with 10 neurons in each, trained separately and then stacked in sequence together with an output layer consisting of 2 neurons. Also optimal network initialization techniques were studied and chosen in order to get best training and generalization performance.

For the next step the training methods were selected and implemented. Backpropagation algorithm was chosen for training the networks. Several approaches were considered in order to improve the backpropagation algorithm convergence speed and generalization abilities. These include training functions: Bayesian Regularization, Scaled Conjugate Gradient, Conjugate Gradient with Powell-Beale restarts and Resilient Backpropagation. The training was done with two performance functions: Mean Squared Error and Cross-entropy, since each of them perform best with different training functions. Also in order to control the network bias and variance balance and to prevent it from overfitting the weight decay regularization term was added to the performance functions. The computational requirements and parallelization possibilities for these training functions were also considered and compared, in order to get fastest performance possible.

Trained networks were then evaluated on the next step. Each of them underwent series of evaluation metrics used for binary classifiers like: precision, recall, f score and Receiver Operating Characteristic. This test provided insight into which architecture and training approach has resulted in best classification performance and how they compared to an existing, non-neural approach. Tests showed improvement compared to an old method and also showed that feed-forward neural network with Bayesian regularization and Autoencoder with scaled conjugate gradient training have shown the best performance out of all. The evaluation metrics also helped in choosing optimal discrimination threshold values, which can result in optimal

performance of the classifier. Several of this thresholds were calculated and they can be applied based on the business requirements of the insurance company, since they regulate the "strictness" of the classification, which means getting higher recall values by the price of diminished precision.

The best performing networks were deployed as functions and can be used further for implementing them on any desktop, mobile or web application. This function holds the weight and bias matrixes of the network and their implementation does not require any additional neural network libraries other than implementing basic matrix operations.

For the further development of this approach, as it was shown during the feature selection step of the study, getting new and informative features is the best way to improve the performance of the model. Final results heavily depends on the subset of features used during training and getting new data, which was not available during this study can even further improve the classification and forecasting possibilities of the artificial neural network.

REFERENCES

[1] George E. Rejda, *Principles of Risk Management and Insurance*, 10th ed. New York: NY: HarperCollins, 2004.

[2] C. Dugas, Y. Bengio, N. Chapados, P. Vincent, G. Denoncourt, and C. Fournier, "Statistical Learning Algorithms Applied to Automobile Insurance Ratemaking," *CAS Forum*, vol. 1, no. 1, pp. 179–214, 2003.

[3] F. L. Kitchens, "Artificial Neural Networks Used in Automobile Insurance Underwriting," pp. 168–170, 2005.

[4] R. Holtom, *Underwriting, principles & practices*, 3rd ed. The National Underwriter, 1988.

[5] J. Lemaire, *Automobile Insurance: Actuarial Models*. Philadelphia: University of Pennsylvania, 1985.

[6] James Cooper Rose, "An expert system model of commercial automobile insurance underwriting," The Ohio State University, 1986.

[7] P. Mulquiney, "Artificial Neural Networks in Insurance Loss Reserving," no. September, pp. 1–4, 2003.

[8] B. Widrow, D. E. Rumelhart, and M. a. Lehr, "Neural networks: applications in industry, business and science," *Commun. ACM*, vol. 37, no. 3, pp. 93–105, 1994.

[9] O. Maimon and L. Rokach, *Data Mining and Knowledge Discovery Handbook*. 2005.

[10] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas, "Data preprocessing for supervised learning," *Int. J. Comput. Sci.*, vol. 1, no. 2, pp. 111–117, 2006.

[11] N. V Chawla, N. Japkowicz, and P. Drive, "Editorial : Special Issue on Learning from Imbalanced Data Sets," *ACM SIGKDD Explor. Newsl.*, vol. 6, no. 1, pp. 1–6, 2004.

[12] F. Provost, "Machine learning from imbalanced data sets 101," *Proc. AAAI'2000 Work. ...*, p. 3, 2000.

[13] I. Press, L. Shafer, G. W. Arnold, and D. Jacobson, "Imbalanced Learning," *Inst. Electr. Electron. Eng. Inc. Publ.*, 2013.

[14] M. H. Beale, M. T. Hagan, and H. B. Demuth, "Neural Network Toolbox™ User's Guide," 2015.

[15] Y. A. LeCun, L. Bottou, G. B. Orr, and K. R. Muller, "Efficient backprop," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7700 LECTU, pp. 9–48, 2012.

[16] I. Bin Mohamad and D. Usman, "Standardization and its effects on K-means clustering algorithm," *Res. J. Appl. Sci. Eng. Technol.*, vol. 6, no. 17, pp. 3299–3303, 2013.

[17] E. Fitkov-Norris, S. Vahid, and C. Hand, "Evaluating the Impact of Categorical Data Encoding and Scaling on Neural Network Classification Performance: The Case of Repeat Consumption of Identical Cultural Goods," *Commun. Comput. Inf. Sci.*, vol. 311, pp. 343–0352, 2012.

[18] M. a Hall, "Correlation-based Feature Selection for Machine Learning," *Methodology*, vol. 21i195–i20, no. April, pp. 1–5, 1999.

[19] R. Kohavi, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," *Int. Jt. Conf. Artif. Intell.*, vol. 14, no. 12, pp. 1137–1143, 1995.

[20] Z. Reitermanov, "Data Splitting," *WDS'10 Proc. Contrib. Pap.*, pp. 31–36, 2010.

[21] M. T. Hagan, H. B. Demuth, and M. H. Beale, "Neural Network Design," pp. 1–1012, 1995.

[22] S. Hawkins, H. He, G. Williams, and R. Baxter, "Outlier Detection Using Replicator Neural Networks," *Data Warehous. …*, pp. 170–180, 2002.

[23] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd. ed. Prentice Hall, 1998.

[24] C. M. Bishop, *Pattern Recognition*. New York: Springer-Verlag Inc., 2006.

[25] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[26] Y. Bengio, "Learning Deep Architectures for AI," *Found. Trends® Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009.

[27] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," no. Ml, pp. 1–14, 2013.

[28] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *Ann. Math. Stat.*, vol. 22, no. 1, pp. 79–86, Mar. 1951.

[29] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks

by choosing initial values of the adaptive weights," *IJCNN Int. Jt. Conf. Neural Networks*, vol. 13, p. C21, 1990.

[30] C. M. Bishop, "Neural networks for pattern recognition," *J. Am. Stat. Assoc.*, vol. 92, p. 482, 1995.

[31] G. E. Hinton, "Learning Translation Invariant Recognition in Massively Parallel Networks," *Vol. I Parallel Archit. PARLE Parallel Archit. Lang. Eur.*, pp. 1–13, 1987.

[32] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6, pp. 525 – 533, 1993.

[33] M. J. D. Powell, "Restart procedures for the conjugate gradient method," *Math. Program.*, vol. 12, no. 1, pp. 241–254, 1977.

[34] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," *IEEE Int. Conf. Neural Networks - Conf. Proc.*, vol. 1993-Janua, pp. 586–591, 1993.

[35] D. J. C. MacKay, "A Practical Bayesian Framework for Backpropagation Networks," *Neural Comput.*, vol. 4, no. 3, pp. 448–472, 1992.

[36] D. J. C. MacKay, "Bayesian methods for adaptive models," *PhD Thesis*, vol. 1992, 1991.

[37] F. D. Foresee and M. T. Hagan, "GAUSS-NEWTON APPROXIMATION TO BAYESIAN LEARNING ** School of Electrical and Computer Engineering," *Network*, pp. 1930–1935, 1930.

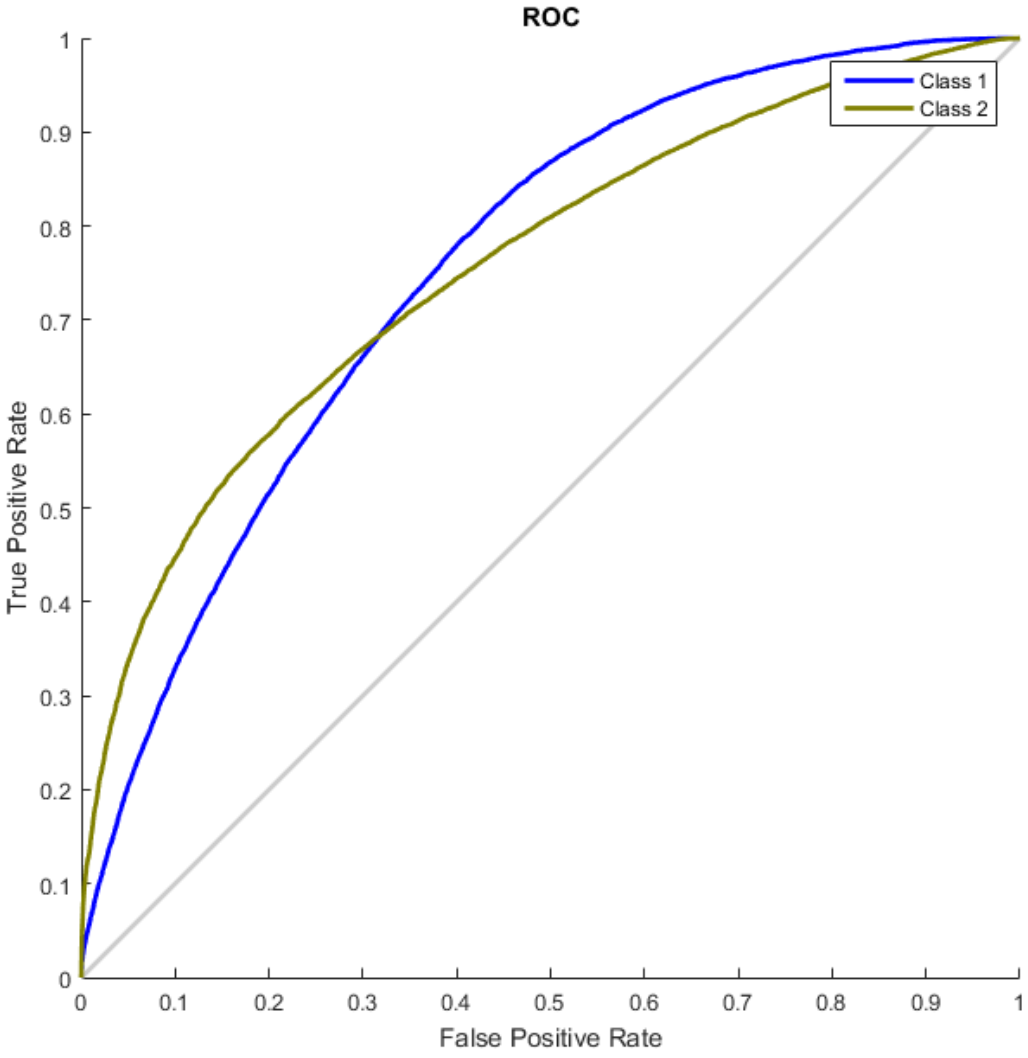
[38] N. V Chawla, "Data Mining for Imbalanced Datasets: An Overview," *Data Min. Knowl. Discov. Handb.*, pp. 853–867, 2005.

[39] C. J. . Van Rijsbergen, "Evaluation," University of Glasgow, 1979.

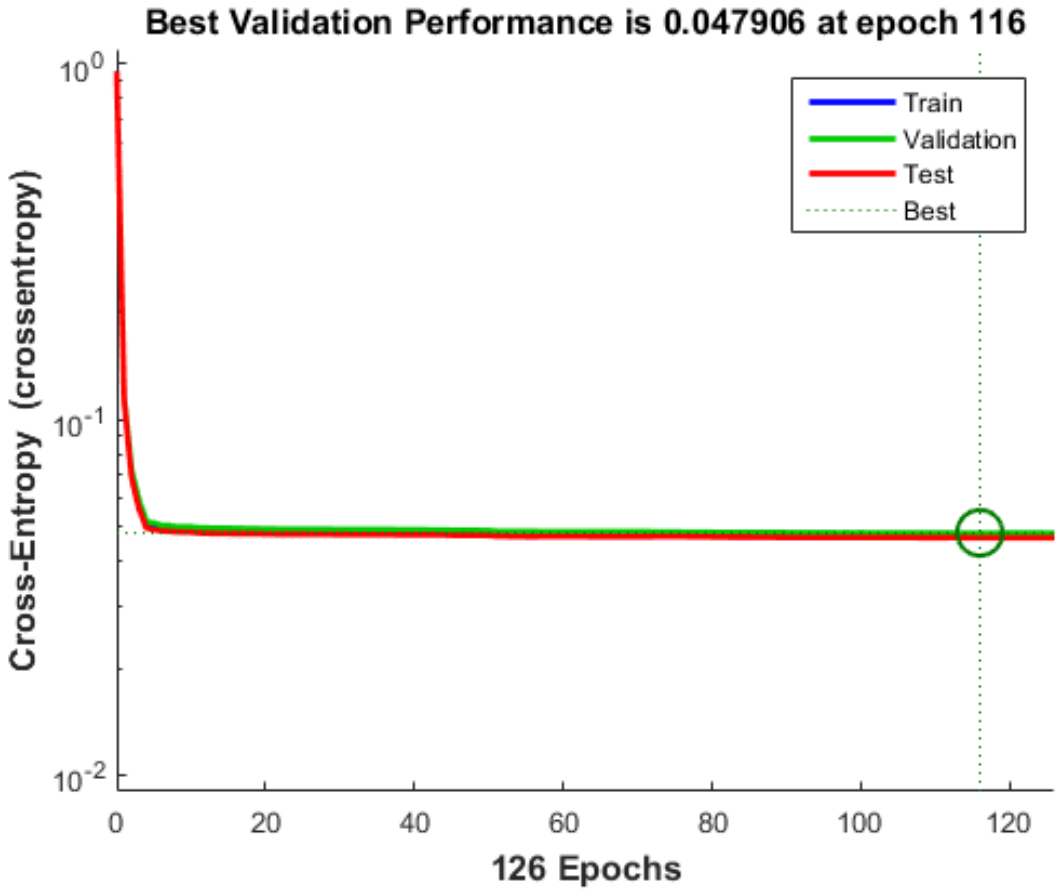
[40] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognit.*, vol. 30, no. 7, pp. 1145–1159, 1997.

APPENDICES

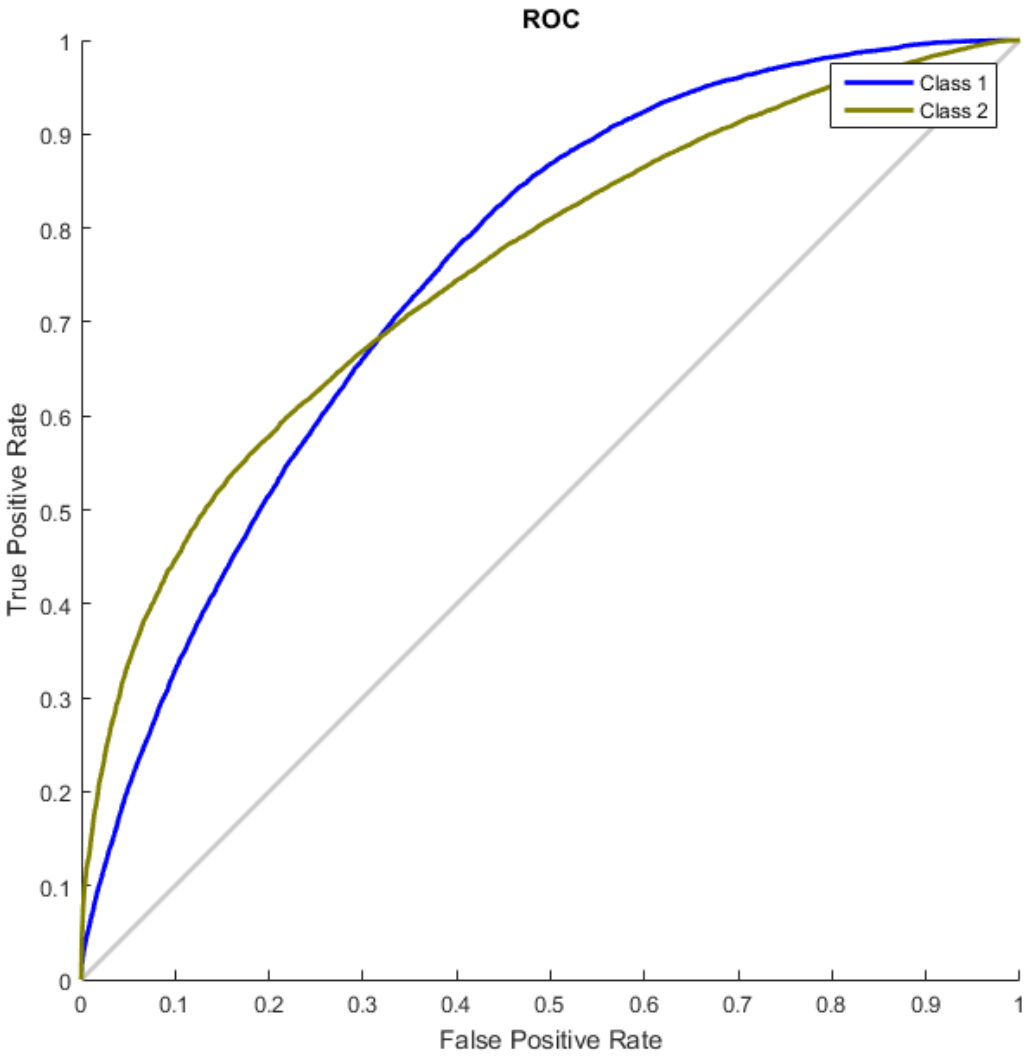
Appendix 1. Receiver Operating Characteristic for Scaled Conjugate Gradient method



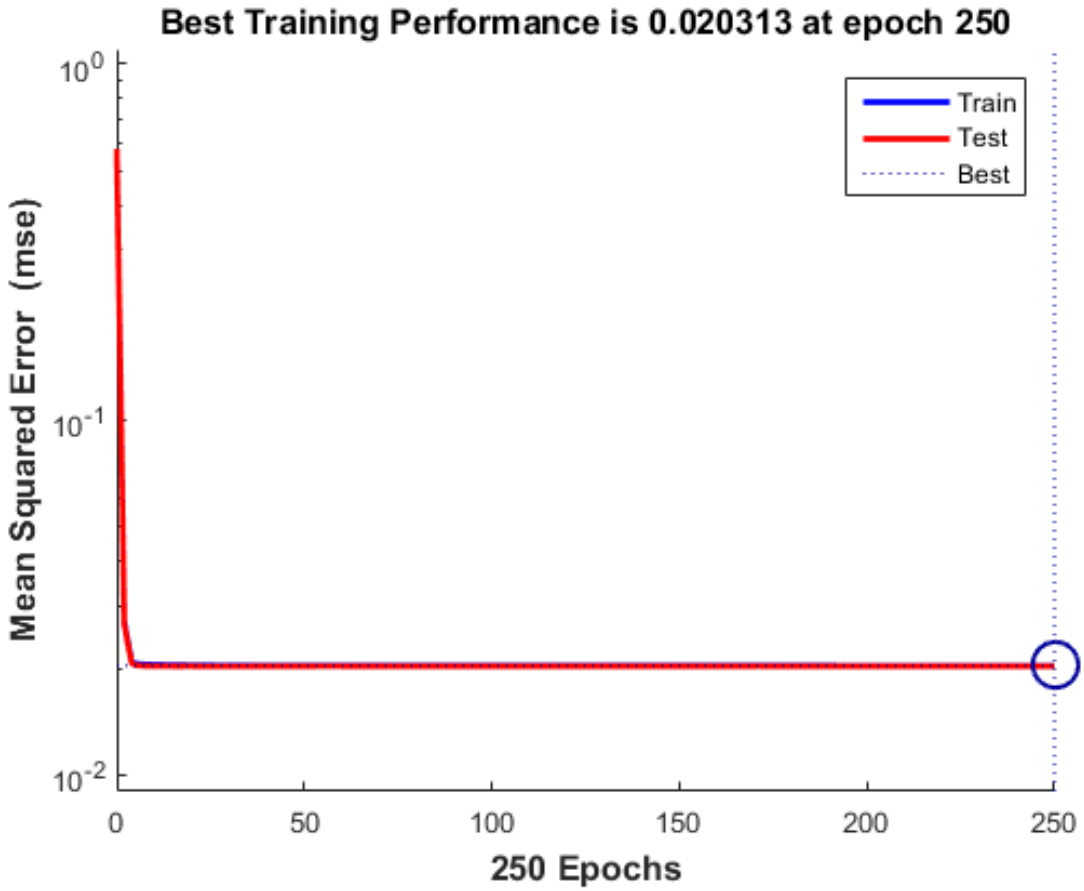
Appendix 2. Training performance plot for Scaled Conjugate Gradient method



Appendix 3. Receiver Operating Characteristic for Bayesian Regularization method



Appendix 4. Training performance plot for Bayesian Regularization method



Appendix 5. Deployed function code snippet for Autoencoder network

```
function [y1] = AutoencoderNeuralNetworkFunction(x1)
%AUTOENCODERNEURALNETWORKFUNCTION neural network simulation function.
%
% Generated by Neural Network Toolbox function genFunction, 01-May-2016 19:33:27.
%
% [y1] = AutoencoderNeuralNetworkFunction(x1) takes these arguments:
%   x = 11xQ matrix, input #1
% and returns:
%   y = 2xQ matrix, output #1
% where Q is the number of samples.

%#ok<*RPMT0>

% ===== NEURAL NETWORK CONSTANTS =====

% Input 1
x1_step1_xoffset = [0.00653968963238101;0;0;0;0;0;0;0;0.0105763089468793;0];
x1_step1_gain = [167.211047863039;2;86.9316770186336;2;89.5;2;2;2;2;46.9451810601615;31.6666666666667];
x1_step1_ymin = -1;

% Layer 1
b1 = [-2.3122852307384076;2.4278681738642254;2.6987279666580366;2.1978766956996099;1.8257297720587156;-0.55
IW1_1 = [5.4020229470146539 0.46523479214396557 0.78517011572763307 -1.2200695783129183 6.6092858856271413

% Layer 2
b2 = [-2.9172325597170654;2.9167274490819302];
LW2_1 = [-0.68809591381188584 -1.4607728610405701 -1.6238488476606483 6.9788788054694297 -0.619762547460930

% ===== SIMULATION ========

% Dimensions
Q = size(x1,2); % samples

% Input 1
xp1 = mapminmax_apply(x1,x1_step1_gain,x1_step1_xoffset,x1_step1_ymin);

% Layer 1
a1 = tansig_apply(repmat(b1,1,Q) + IW1_1*xp1);

% Layer 2
a2 = softmax_apply(repmat(b2,1,Q) + LW2_1*a1);

% Output 0
y1 = a2;
end

```

Appendix 6. Deployed function code snippet for Feed-forward network

```
function [y1] = BayesianNeuralNetworkFunction(x1)
%BAYESIANNEURALNETWORKFUNCTION neural network simulation function.
%
% Generated by Neural Network Toolbox function genFunction, 01-May-2016 19:26:58.
%
% [y1] = BayesianNeuralNetworkFunction(x1) takes these arguments:
%   x = 11xQ matrix, input #1
% and returns:
%   y = 2xQ matrix, output #1
% where Q is the number of samples.

%#ok<*RPMT0>

% ===== NEURAL NETWORK CONSTANTS =====

% Input 1
x1_step1_xoffset = [0.00653968963238101;0;0;0;0;0;0;0;0.0105763089468793;0];
x1_step1_gain = [167.211047863039;2;86.9316770186336;2;89.5;2;2;2;2;46.9451810601615;31.6666666666667];
x1_step1_ymin = -1;

% Layer 1
b1 = [-2.3122852307384076;2.4278681738642254;2.6987279666580366;2.1978766956996099;1.8257297720587156;-0.
IW1_1 = [5.4020229470146539 0.46523479214396557 0.78517011572763307 -1.2200695783129183 6.609285885627141

% Layer 2
b2 = [-2.9172325597170654;2.9167274490819302];
LW2_1 = [-0.68809591381188584 -1.4607728610405701 -1.6238488476606483 6.9788788054694297 -0.619762547460

% ===== SIMULATION ========

% Dimensions
Q = size(x1,2); % samples

% Input 1
xp1 = mapminmax_apply(x1,x1_step1_gain,x1_step1_xoffset,x1_step1_ymin);

% Layer 1
a1 = tansig_apply(repmat(b1,1,Q) + IW1_1*xp1);

% Layer 2
a2 = softmax_apply(repmat(b2,1,Q) + LW2_1*a1);

% Output 1
y1 = a2;
end

```