Spring 2014

Big Data Analysis Using Neuro-Fuzzy System

Amir Eibagi
San Jose State University

CS298 Report

Big Data Analysis Using Neuro-Fuzzy System

Advisor: Dr. Chris Tseng

Amir Eibagi

May 2014

The Designated Committee Approves the Project Titled

Big Data Analysis Using Neuro Fuzzy System

By

Amir Eibagi

Approved for the Department of Computer Science

San Jose State University

May 2014

Dr. Chris Tseng Department of Computer Science _____

Dr. Thomas Austin Department of Computer Science _____

Mr. Aditya Ramesh MTS, Nutanix _____

Abstract

This project addresses big data classification using hybrid Intelligence Classification System. Hybrid Intelligence classification system is a system that combines at least two intelligent technologies. Specifically, the focus of this project is to apply hybrid Neuro-Fuzzy system to the IBM Watson data and Innocentive Trustworthiness challenge data for prediction and classification. Neural network are low-level computational structure which has ability to learn and performs well on the raw data. On the other hand, fuzzy logic deals with reasoning on higher level using If-then rules and linguistic variables. So combining these two methods can provide us with a very powerful classification system.

List of Figures:

List of Tables:

TABLE OF CONTENTS

1. Project Description

1.1 IBM Great Mind Challenge

The focus of this project is to analyze and predict the data of IBM Watson. Watson is an artificially intelligent computer system capable of answering questions posed in natural language. So in this project, I participated in the IBM Great Mind Challenge. In this 4-week competition, we are presented with a very large dataset of IBM Watson. The data consists of 240000 rows and 320 columns. The 320^{th} column of each row is marked with one of the following labels: true or false. So each row of data indicates a question presented to Watson. So all the numbers from column 1 to 319 of each row will be injected to Watson as feature vector and Watson assigns true if it believes the question has been answered correctly or false otherwise.

So we treat these data as our training data set. Once we have fully developed and trained our artificial intelligent algorithm, we will be given an evaluation dataset. The evaluation dataset is very similar to the training dataset except the very last column. In the evaluation dataset, the last column which indicates true or false labels has been removed. This is our job to use our trained algorithm to predict the outcome of each row of data to either true or false.

I used three different methods for this project. In the first method, I used a fuzzy classifier based on the paper "Generating Weighted Fuzzy Rules from Training Data for Dealing with the Iris Data Classification Problem". In the second method, I used neural network classifier. Lastly, I combined the two previous methods in a Neuro-Fuzzy system. Then I will compare the result obtained from each method.

1.2 Instinct Trustworthiness Challenge

This challenge is very similar to IBM Great mind challenge in terms of the data classification problem. However the format of the training dataset is very different from IBM training data. The training dataset is consists of 415 rows and 115 columns. The first 5 columns in the training data indicates the id of each row. The sixth column corresponds to the class label. In this case, we have four different labels in training dataset; exact amount promised, more than promised, less than promised, and promised not fulfiliable. The rest of the columns are the input features. So each row can be interpreted as a person with many different events and signals (the signals are the input features which in this case is 109). Based on these input features, we can determine where or not trust that person. One thing to note here is that the number of

“Don’t trust” class in training data is much lower compare to the “trust” class. “Don’t trust” class contains 10% of the whole training data. So this constrain makes it harder to predict don’t trust classes in the evaluation dataset. For this challenge, unlike IBM challenge, we are only limited to seven submission per team each week. Also to sign up for trustworthiness competition, we had to form a team. Our team consists of three members where each member uses three different algorithm to obtain better result. The algorithm used are Neuro Fuzzy, genetic decision tree, and neural network classifiers.

1.3 Report Flow

This focus of this report is to analyze the prediction score based on the dataset of the two challenges mentioned above. IBM Great mind challenge happens every academic semester. On the other hand Instinct trustworthiness was introduced to us during spring semester. So we are first going to introduce the methods used against IBM Great mind Challenge and then analyze the results. We then introduce the new method, neuro fuzzy, used this semester to improve the prediction score in IBM challenge. Lastly, we are going to analyze the experiment result using the same method in Great Mind Challenge against Instinct Trustworthiness challenge.

2. Previous Work

2.2.1: Fuzzy Classifier

2.2.1.1: Step1: Fuzzify training data

In this step, we can choose as many fuzzy label/sections for each feature of our data as we desire; this step is called generating membership functions. In this project, I used seven fuzzy membership functions for each feature of data. Since data consists of 319 columns, then we have 319 features; each with 7 equally spaced fuzzy membership functions. As an example, the following picture shows membership function for feature one.

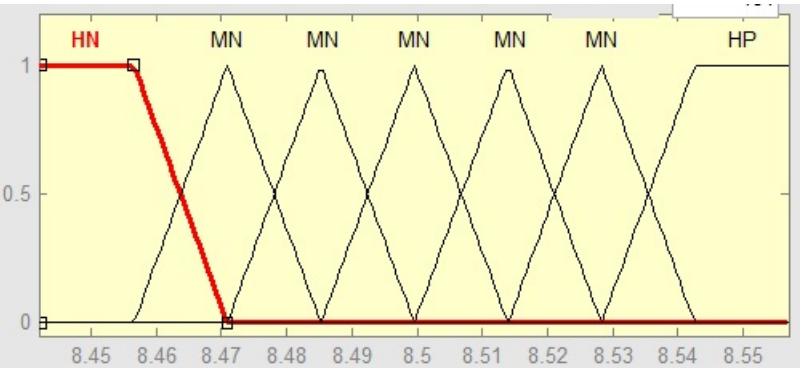


Fig1-Sample membership functions for feature one
X-axis: Range of input value
Y-axis: Fuzzy membership value for input X

Here the Y axis shows a range from 0 to 1. X-axis is the ranges of values for feature one in the data.

So if an input number is 8.46, then this feature 30% belongs to the MN class and %70 belongs to the HN class. Normally, we fuzzify input value to the membership function

with the highest probability. After fuzzifying all the input features, we can move on to the next step.

2.2.1.2: Step2: Generate Reduced Fuzzy Rules

In this step, we use all of the converted training data and generate human readable rules. One solution was is that to convert each row of the data into the fuzzy rules. For example one of the rules could be the following:

If feature1 is MN and feature2 is SN and feature3 is HP and … and feature319 is SN then class is true

The problem with this approach is that we will end up with a very large set of rules which slows down the performance. So an alternate solution is to use reduce fuzzy rule method to shrink the size of the rule set.

Reduce Fuzzy Rule Method:

In this method, we originally start by having each line of the fuzzify training data as our initial rule set. The final reduced rules will be saved into the definitive rule set. We pick a rule from initial rule set and we perform the following checks on it:

1. If Definitive rule set is empty, then we insert the rule into the definitive rule set.
2. If definitive rule set is not empty, we find a rule in the definitive set which has the same class label. We merge these two rules together. Merging of the rules is done as follows:
   Rule1: If feature1 is MN and feature2 is SN then class is True
   Rule2: If feature1 is HP and feature2 is SP then class is True
   Merged rule: iffeature1 is (MN,HP) and feature2 is (SN,SP) then class is true
3. Once we have merged a rule, we should check for collision. This is how we determine if there is a collision: If there exists a rule in the original rule set which has the same features as the merged rule but has different class label, then we have a collision. In case of a collision, merged is not allowed and the rule needs to be added to the definitive rule set.
4. If there is no collision, then we can merge the two rules together and update it in the definitive rule set.

This method allows us to shrink the number of fuzzy rules which will improve our performance.

2.2.1.3: Step3: Defuzzify and classify

This is the final step of the fuzzy classifier method. In this step, we first convert the input data into the fuzzy labels. We can then go through our definitive rule set and check which rule matches the input data. If there is only one match, then we can simply classify the input data with the class label of the matched rule.

However if there is no match or there are two rules matched with different class labels, then we need to generate weights for each fuzzy rule.

Fuzzy rule weight generation:

The degree of the weighted fuzzy rule can be calculated using the following formula:

$$\mathrm{R} = \sum_{i=1}^{n} \varphi_{L_i} * X_i * W_i$$

Where φ_{L_i} is the degree of membership function for the inputX_i,

And $W_i$can be calculated as following:

$$v_i = \frac{|PD|}{|WD|}$$

Where |WD| is the whole domain of the input variable X_i and PD is the set of intervals I_1, $I_{2, \dots}$ which are not overlapping with the domain of the input variable X_i for each type of the classification output.

So $|PD| = |I_1| + |I_2| + |I_3| + \dots$.

And finally

$$W_i = \left(\frac{|PD|}{max(v_1, v_2, \dots, v_n)}\right)^2$$

We finally pick a class label which correspond to the rule with the highest weight in our definitive rule set.

2.2.1.4 Improvement to the fuzzy rule generation method:

Since our data is very large, sometimes we got the same rule weight for two or more fuzzy rules. So in this case, we add up the weights for each classification output in the definitive rule set. For example if we have four rules in our definitive rule set with the following information:

Rule1 with weight 0.4 represent class True

Rule2 with weight 0.2 represent class True

Rule3 with weight 0.4 represent class False

Rule4 with weight 0.1 represent class False

Then if we can add up rule weights for each type of classification, we get the following:

Class true with weight 0.6

Class false with weight 0.5

We can then conclude: we classify data with the true label.

Again, due to the large dataset, this improvement may still result into the duplicate rules with the same weights. In this case, I used the centroid method to perform a better classification. In order to use the centroid method, first we need to determine the membership function for the output. We are only going to have two different types of outputs: true and false. So we only need to have two membership functions. I used the triangular membership function for both true and false labels. To make the calculation easier, I used -1 as false class and used 1 as true class then I used the rule weights along with their class label in the centroid method to find the center of weights in the output membership functions:

$$Result = \frac{\sum_{i=0}^{n=size\ of\ definitive} W_i * Class\ Label_i}{\sum_{i=0}^{n=size\ of\ definitive} W_i}$$

Now by picking a threshold for result we can simply distinguish two class labels.

2.2.1.5 Results (Reduced weighted fuzzy classifier)

The way scores are determined in the IBM Great Mind Challenge is that they count the correct number True questions submitted. We need to submit a simple text file which contains IDs of the questions that we have classified them into the True category. Negative points will be applied for each question incorrectly classified. For example if you submit a file with 200 entries and 50 entries of the entire file is actually belong to the false category while the rest (the other 150) belongs to the true category, then your score will be 150.

So after running the reduced fuzzy classifier, we obtained a low score of 5 on the IBM dataset. We noticed that our reduced rule set contains 53 rules in total. Out of

these rule only 10 rules reflect to the “True” classes. This could mean that we do not have enough true classes’ rule for our reduced fuzzy classifier.

Also the way we are calculating weights may cause having this low score. We are looking for overlap intervals for both True and false classes but due to the low number of true classes, we will not have too many overlaps. Therefore “False” class rules will most likely have higher weights associated with them. Some other rule weight generation can be used and experimented. We also conduct a small test to detect how well our reduced fuzzy classifier can predict and here are the result:

Classes	Correct Classification	Incorrect Classification
True	0.1%	6.9%
False	92%	1%

Table1-Reduced weighted Fuzzy Classifier result on IBM dataset

The table above covers the whole 100% of the data. So 7% of the data are True classes and the rest are all “False” classes.

Since we have such low score for reduced Fuzzy Classifier, we decided to use Neural Network on IBM data set.

2.2.2: Neural Network Classifier

Our brain is made up a lot of tiny unit called neurons. Each neuron is connected to so many other neurons and it communicates with them via electrochemical signal. To simplify things, when a message received to a neuron by other neurons (here we call them input signals), then neuron somehow sums up the inputted signals. If the inputted signal is above some threshold, then neuron generates and fires an output voltage. This is how neurons transmit an action in human brain.

To illustrate the same behavior, we use the idea of artificial neural network. Similarly, neural networks are made of many artificial neurons. Normally, we design neural network layer by layer. If our neural network is only consists of two layers, input and output layers, then neural network is called a perceptron. The following picture shows a perceptron:

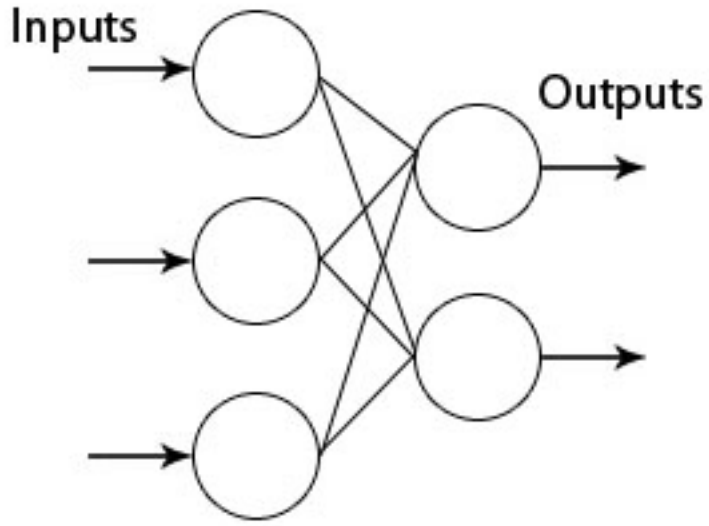


Fig2-Perceptron overview

The problem with perceptron is that it is very limited. It means that if the training data is linearly separable then perceptron converges. Otherwise it fails to classify. So this was the motivation of designing multi-layer neural networks.

In multi-layer neural networks, we have input layer, hidden middle layer, and output layer as shown below:

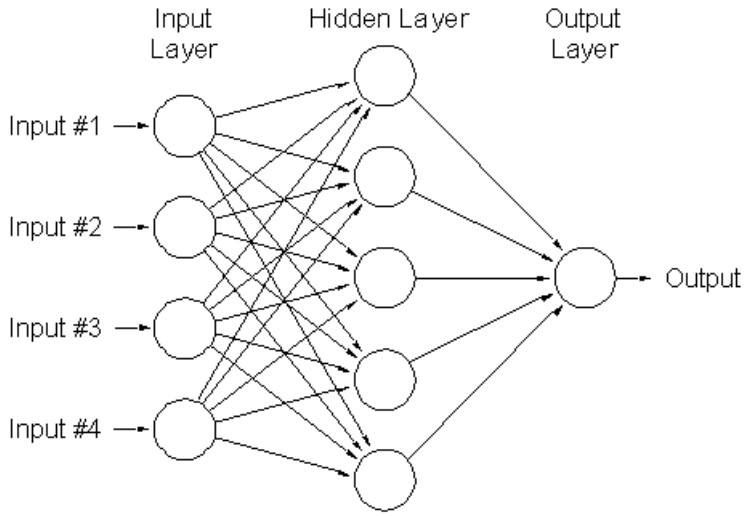


Fig3-Multi-Layer Neural Network

Note that the above picture only shows 3-layer neural network. We could design a neural network with many hidden layers. Each of the neurons in the input layer could be connected to many neurons in the hidden layer. Similarly, each neuron in the hidden layer could be connected to only one or many output neurons depending on number of the outputs. But the question here is how does multi-layer neural network learns and predict the output?

There are many different learning algorithms for neural network such as feed-forward, mini-batch, momentum, adaptive learning rate, rmsprop and many others. To illustrated learning, each neuron is connected to a neuron in the next layer with a weight. Depending on how close neural network predict the output value with the correct value, the weights in each layer gets updated. Once of the most well-known and widely used neural networks is a feed-forward neural network with back propagation learning algorithm. In the feed-forward neural network, there is no cycle; information from neurons either passes forward until it reaches the output. In the back propagation algorithm, the output value is compared to the original value using mean squared error (other comparison method could be used but MSE is the most popular and accurate one). Once error is determined, then error back propagates throughout the network and weights will be updated depending on the error.

2.2.2.1 Neural network Design Overview

Neural networks are very suitable for pattern recognition and classification. Matlab provides an extensive library and toolbox on neural network. I design a neural network using Matlab Neural network toolbox and exercised it by changing different parameters in order to obtain a better performance and result. The basic design of the project is as follows:

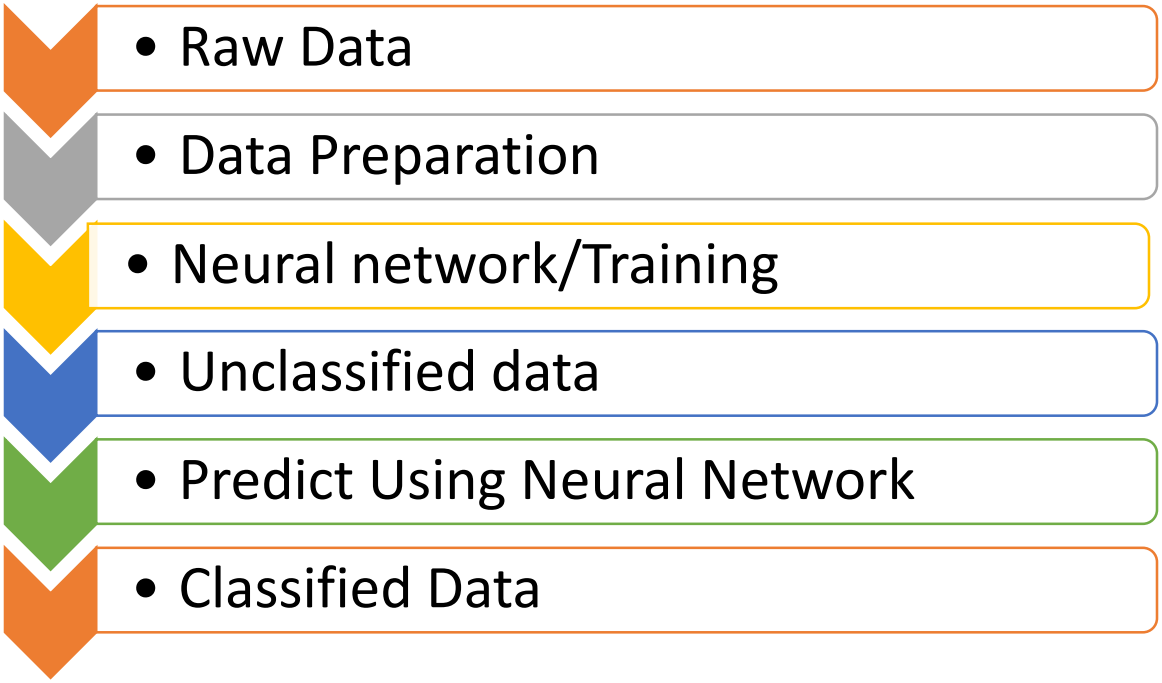


Fig4- Project Design Overview for Neural Network

2.2.2.2 Raw Data

For this project, we are given a very large dataset of IBM Watson. The dataset divides into two main components; training dataset and evaluation dataset. The training data set is a labeled data consists of 239000 rows and 320 columns. Each row and column, except last column, contains some number. The last column in the training dataset is labeled either true or false. As described earlier, each row of the training dataset can be interpreted as a question presented to Watson. Depending on the numbers given, Watson then evaluates the question to be either true or false. On the other hand, evaluation dataset is very similar to the training dataset except the last column in training dataset is removed from the data. Evaluation dataset will be fed into the neural network in order to predict the outcome as either true or false labels.

2.2.2.3 Data preparation

Since the data is very large, it will not be able to fit into the Matlab. Matlab will run out of memory if the whole data is presented all at one. So the training data needs to be broken into several parts. I used CSVChunker to split the training data into 13 different parts. Each part contains 20000 rows of original training dataset.

On the other hand, Matlab neural network require two different file, input and output files, in order to perform classification. Input file will be all the training input from column 1 to column 319. Output file will contain either 0 as false label or 1 as true label.

2.2.2.4 Training algorithms used.

Neural networks can be trained with different training algorithms. Matlab has a very large library in terms of training functions. For this project, we experiment with all different Matlab training functions and recorded the result. We will only show the result of Scaled Conjugate Gradient Descent Method because it gave us the best prediction score. The training functions that Matlab offers are:

1. Levenberg-Marquardt (trainlm)
2. Bayesian Regularization (trainbr)
3. BFGS Quasi-Newton (trainbfg)
4. Resilient Backpropagation (trainrp)
5. Scaled Conjugate Gradient (trainscg)
6. Conjugate Gradient With Powell/Beale Restarts (traincgb)
7. Fletcher-Powell Conjugate Gradient (traincfg)
8. Polak-RiBiere Conjugate Gradient (traincgp)

9. One Step Secant (trainoss)
10. Variable Learning Rate Gradient Descent (traingdx)
11. Gradient Descent with Momentum (traingdm)
12. and Gradient Decent (traingd)

2.2.2.5 Experiment Result

Out of all of the training algorithms mentioned above, Scaled Conjugate Gradient gave the he best prediction score. This training method combine the model-trust region approach used in the Levenberg-Marquardt algorithm, with the conjugate gradient approach. So it doesn't perform line search for finding the direction of gradient descent. Since this method initially, default parameters, gave a better result than all other methods, we decided to use this method and change parameters in order to increase the scoring result. Initially we use 20 hidden neurons and 10 validation checks. For this configuration we had a score about 103 with.

So here are the error, gradient and confusion plots for the setting mentioned above:

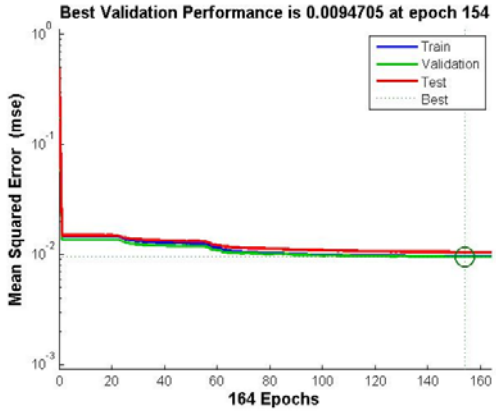


fig5-trainsgc-Error vs Epochs

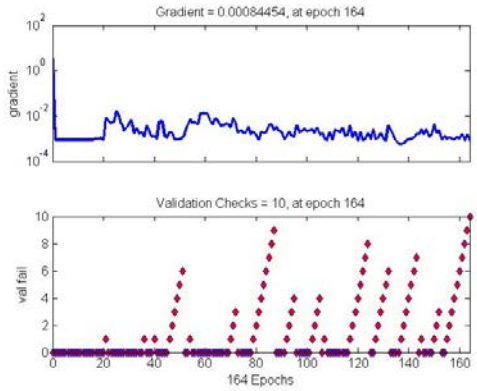


fig-6 tarinscg- Gradient vs Epochs

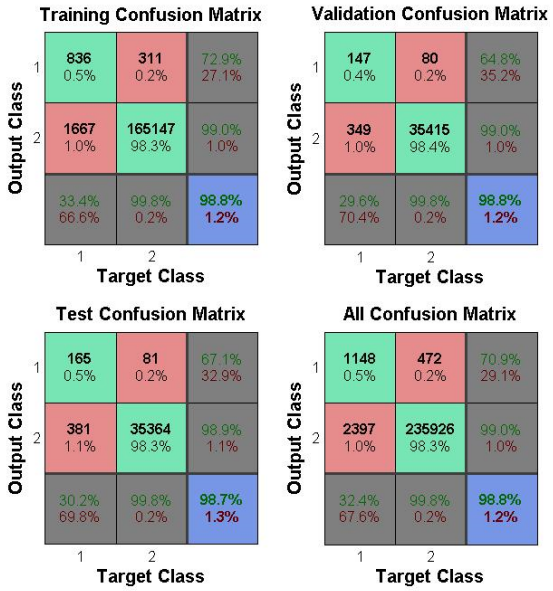


fig7- trainscg- Confusion Matrix (Correct and Incorrect Classifications)

Looking at the MSE graph, we see that error becomes constant after epoch 60 and gradient is not very stable. The result we have obtained here has the lowest error and gradient compare to most of the other methods which explain high prediction score

Now we tried to change some parameters such as number of hidden layers, number of validation checks, and initial seed number to increase the training epoch and decrease the error rate and gradient. Here is the table of different scores with different parameters:

Test	Hidden Layers	Validation Checks	Seed	Score
1	20	6	49122356	103
2	20	10	49122356	100
3	40	10	49122356	106
4	70	10	49122356	74
5	10	10	49122356	91
6	25	10	49122356	106
7	30	10	49122356	117
8	100	10	49122356	100
9	200	10	49122356	87
10	35	10	49122356	118
11	35	10	49121	130
12	52	10	4912183	131
13	55	10	4912183	135

14	55	20	4912183	145
15	55	40	4912183	145

Table2-Comparison of trainscg method with different parameter

Tests 1 through 10 show how changing number of hidden neurons in the hidden layer can affect the scoring result. We then focus on changing the random seed number as well as hidden neurons. After many trials, we decided to choose random seed number as 4912183. Also increasing the number of hidden neurons to 55 gave us a higher score. We then noticed that increasing the validation checks causes neural network to perform more iteration therefore we will get a lower error rate and gradient descent. So here are the final; plots which reflect test 15:

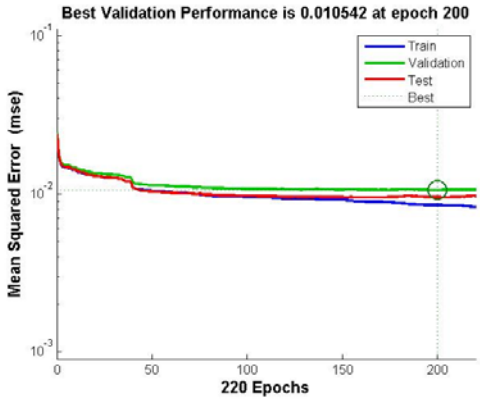


Fig8-Improved trainsgc-Error vs Epochs

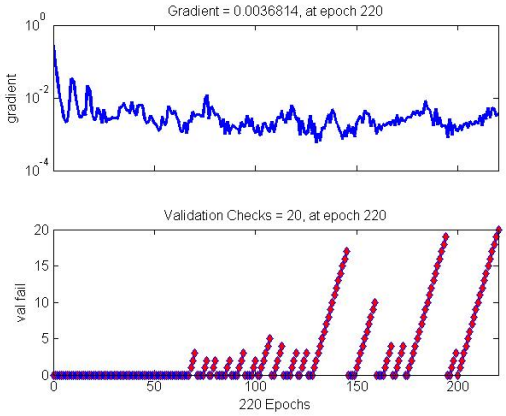


Fig9- Improved tarinscg- Gradient vs Epochs

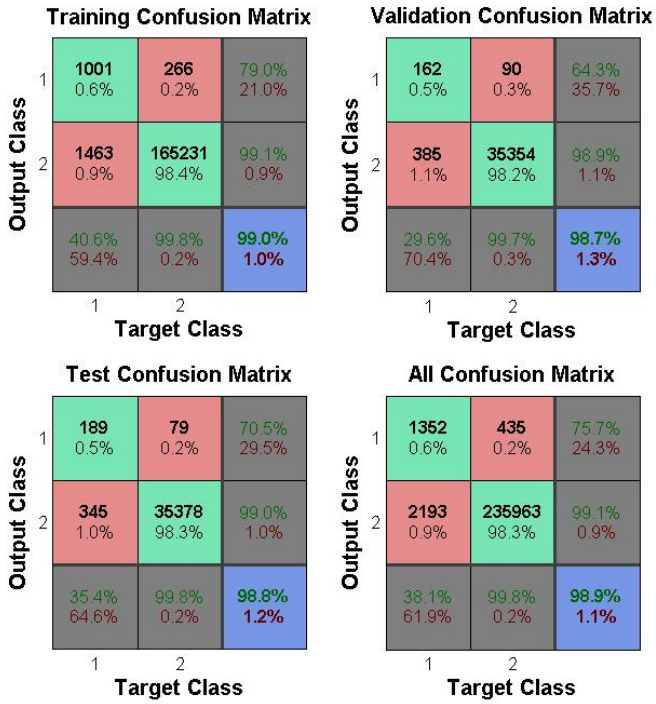


Fig10- improved trainscg- Confusion Matrix (Correct and Incorrect Classifications)

Notice now training error is constantly decreasing. Also we have obtained a much lower gradient. Looking at the confusion matrix, we can see that now we have performed better in terms of misclassification and correct classification of “true” class.

3. New Method (Neuro Fuzzy Classifier)

Hybrid intelligence systems became popular to solve more complex problem in machine learning in terms of space and dimension. Hybrid intelligence system combines two or more artificial intelligence technique and algorithms. Fuzzy-Neuro System is a hybrid intelligence system where it combines both Fuzzy logic and neural network.

	Fuzzy System	Neural Network
Knowledge representation	Yes	No
Uncertainty tolerance	Yes	Yes
Imprecision tolerance	Yes	Yes
Adaptability	Yes	Yes
Learning ability	No	Yes
Explanation ability	Yes	No

Table3-Fuzzy System vs Neural Network

According to the table …., we can see that Fuzzy systems are very suitable for data/knowledge representation (using IF-Then rules) as well as explanation and analysis of data. Even though fuzzy systems are great in presenting and explaining data but they do not have learning capability. On the other hand, neural networks have the many different learning algorithm but they are not so great when it comes to knowledge presentation. We can't just present knowledge by saying a particular weight would give us the best result; the knowledge is distributed throughout the whole network. So this makes neural network like a black box to the user. In addition, neural network can't really explain data which makes it very crucial when it comes into prediction. Neural network only outputs the data from the training it had without any further explanation. So combining these two systems can helps obtain better prediction rate in the IBM great mind Challenge.

3.1 Design Overview

A Neuro Fuzzy system is a homogenous hybrid intelligence system. It takes advantage of a neural network learning ability to design a network which is very similar to the fuzzy inference system in terms of functionality. Such system uses base fuzzy system and expert knowledge to present data and then it uses neural network in order to develop if-then rules and adjust input/output membership function to improve the overall performance of the system. The overall structure of the Neuro Fuzzy system is very alike neural network. Basically the system consists of 5 layers. Similar to neural network, it has an input and output layer as well as 3 middle layer in order to present the fuzzy system. The overall architecture of the Neuro Fuzzy system is shown in the picture below:

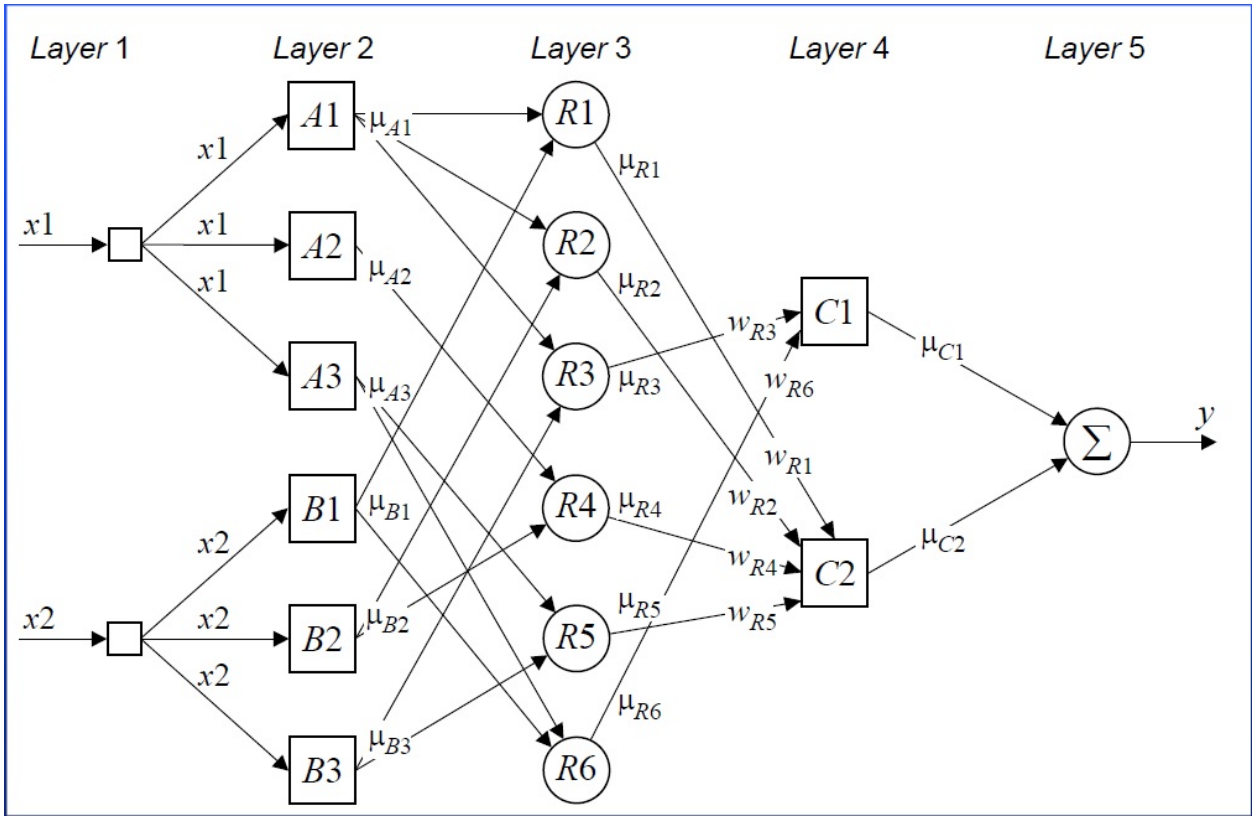


Fig11-Neuro Fuzzy system Structure

a. **Layer 1 (Input layer)**:

In this layer input neurons enter the network. The input neurons here are the crisp values that are in our dataset. So we simply forward these neurons into the next layer.

b. **Layer 2 (Fuzzification Layer)**:

In this layer, input neurons will be fuzzified according to the membership function chosen for the input neurons. Basically this layer is responsible to determine a degree in which a crisp input value belongs to a certain membership function. As mentioned earlier, there are three major type of a membership function (Bell, Triangular and Trapezoid) and each can be a fit into our Neuro Fuzzy system depending on the problem. Normally Bell (Gaussian) membership function gives us a better result since it is covering the other two cases however sometimes we need to be very precise on the fuzzy value. So we can use either trapezoid or triangular membership functions.

c. **Layer 3 (Fuzzy Rule Layer)**

This layer represent the fuzzy rules that we have in our system. Each neuron is mapped into a fuzzy rule. So for example in the picture above, R2 is a fuzzy rule

which receive input s from A1 and B2. Then the intersection of these two input neurons (according to the rule) needs to be computed by using a product operator.

$$Y_i^{(3)} = X_{1i}^{(3)} \times X_{2i}^{(3)} \times \ldots \times X_{ki}^{(3)}$$
$$Y_{R2}^{(3)} = \mu_{A1} \times \mu_{B2} = \mu_{R2}$$

d. Layer 4 (Output fuzzy set)

This layer represents the output neurons processed by fuzzy system. Number of neurons in this layer corresponds to the number of classes we have in our dataset. So the input to this layer is the output of the fuzzy rules Therefore new need to take a union of the inputs in order to defined a fuzzified output neuron.

$$Y_i^{(3)} = X_{1i}^{(4)} \oplus X_{2i}^{(4)} \oplus \ldots \oplus X_{ki}^{(4)}$$

$$Y_{C2}^{(4)} = \mu_{R1} \oplus \mu_{R2} \oplus \mu_{R4} \oplus \mu_{R5} = \mu_{C2}$$

e. Layer 5 (Defuzzification layer)

Finally we need to defuzzify our processed neurons into a crisp output value. In this layer, we use the output membership functions to obtain a value for the input neuron. Depending on where exactly the value will be defuzzified into, we can associate an appropriate class label to that value. There could be many different defuzzification method applied in this layer such as centroid and sum of product. Defuzzification method is explained in detail under "Step3: Defuzzify and Classify" of fuzzy systems

3.2 Neuro Fuzzy in IBM great mind Challenge

We have used neural network and fuzzy classifier in the previous IBM challenge. So we decided to use the new method using Neuro fuzzy classifier to come up with a prediction score and then compare it to neural network and fuzzy system classifiers. To start with, we used a Neuro fuzzy classifier developed in matlab. This tool requires set of fuzzy rules as input as well as network parameter adjustments such as type of fuzzy membership functions, number of membership functions, hidden layers, number of validation checks, and random seed value.

Once again, the goal of this competition is to predict and find the rows in evaluation dataset where we believe they belong to the class labeled "true". The Neuro fuzzy classifier tool in matlab consist of 4 different classifiers, each with slightly different setup. So the initial setup that we used consist of 50 hidden neurons, 7 triangular membership functions, no random seed value, and reduced fuzzy rules (explained in Fuzzy classifier section). We first used the first classifier which uses a simple

backpropagation and a triangular membership function. The score we obtained wasn't very impressive. The total true values detected was around 67.

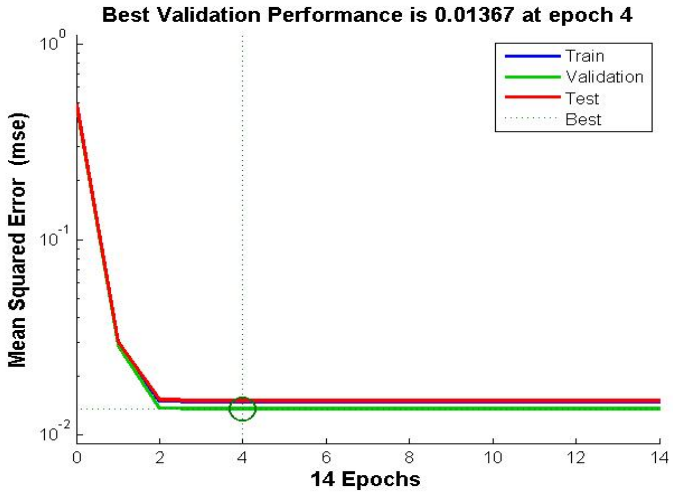


Fig11-BackProp 3 memFunc-Error vs Epochs

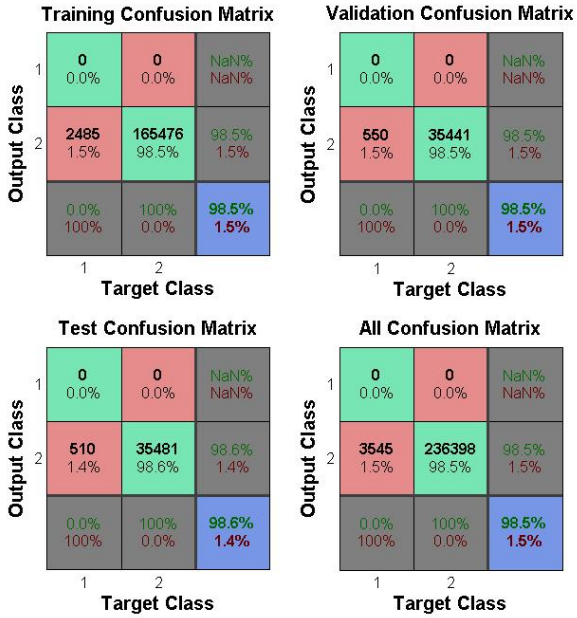


Fig13- BackProp 3 memFunc – Confusion Matrix (Correct and Incorrect Classifications)

The overall error is already low enough but the prediction score is not very satisfying. The reason could be triangular membership function uses a crisp output value. So in the new few sections, we are going to use different methods in order to improve the prediction scores.

3.3 Neuro Fuzzy improvements and result

3.3.1 Removing data redundancy/feature selection

Since our dataset is very large, there is always a chance for data redundancy. Our data consist of 319 columns. Each column can be interpreted as a feature. So feature selection can help us to reduce number of features in our dataset which will simplify our problem. The first step is to find out how many columns in training and evaluation dataset has all zeroes or ones. Furthermore, we need make sure that these features match in both training and evaluation dataset. Looking at the graphs below, we can see that the distribution of zeroes and ones in training and evaluation dataset are very similar therefore we can remove them from dataset.

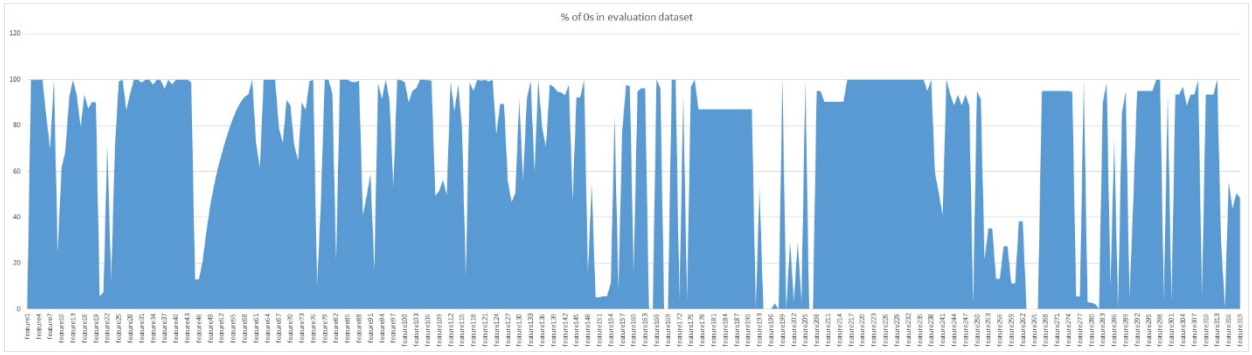


Fig14-Zeroes in training dataset(x-axis: features--------y-axis: percentage of zeroes 0-100)

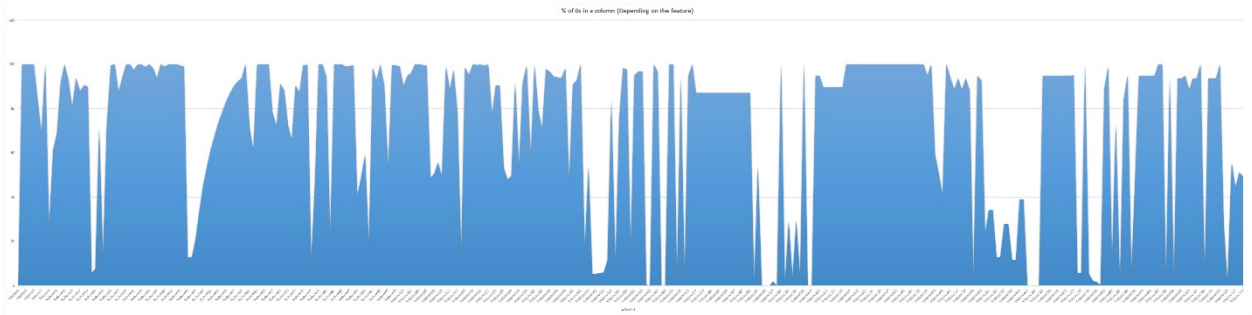


Fig15-Zeroes in evaluation dataset(x-axis: features--------y-axis: percentage of zeroes 0-100)

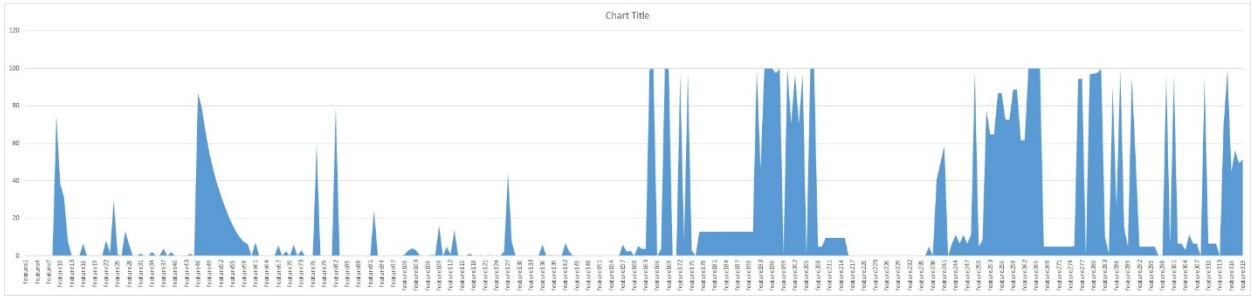


Fig16-Ones in training dataset(x-axis: features--------y-axis: percentage of zeroes 0-100)

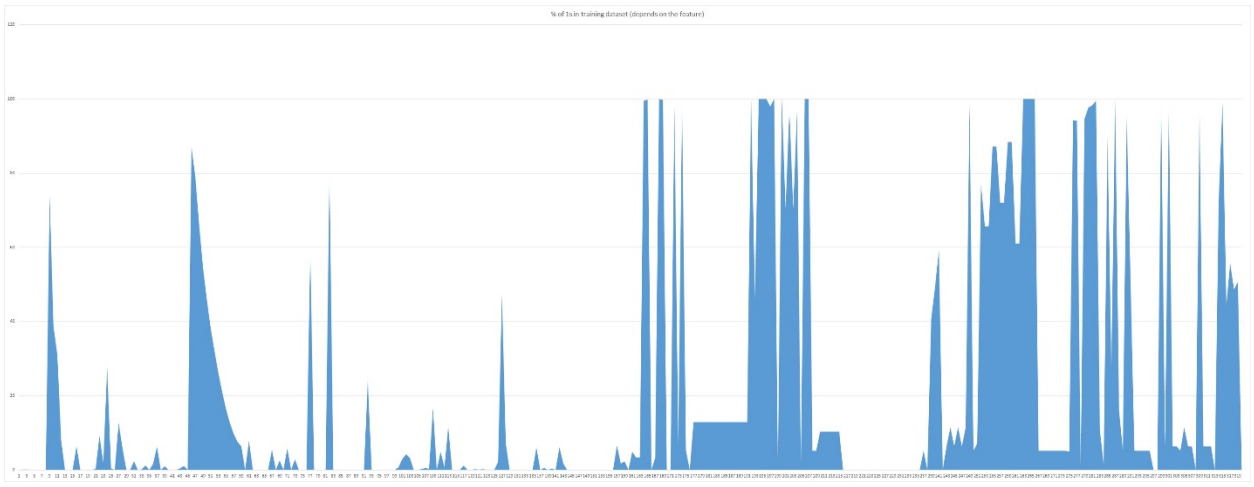
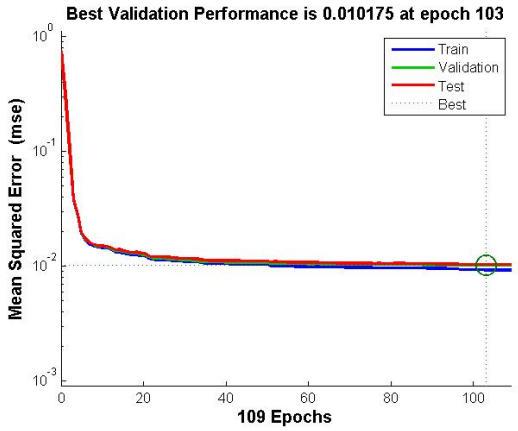


Fig17-Ones in evaluation dataset (x-axis: features--------y-axis: percentage of zeroes 0-100)

We first removed columns that are all zeroes from two dataset. This method reduced number of features to 273 features. This is the decrease of 52 features in our dataset. We then used this new dataset against Neuro Fuzzy classifier (first classifier in Matlab neuro fuzzy classifier tool). The overall process was faster than the original dataset because we have less number of feature. The prediction rate was almost the same as what I had obtained from last semester. The prediction rate was around 90%, and the number of true values I detected in the evaluation dataset was 118. This is a great improvement compare to the original network setup. Graphs below show a decrease in overall MSE and performance of the network which result in a better prediction scores.



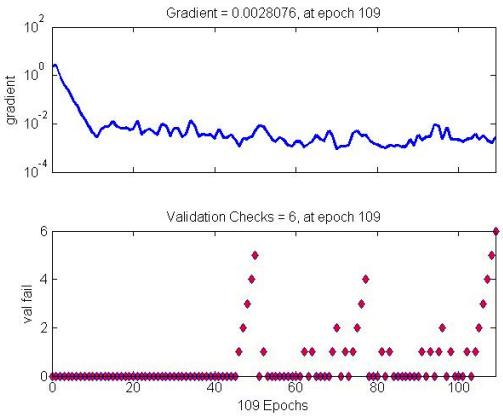


Fig18-only zeroes-Error vs Epochs *Fig19- only zeroes – Gradient vs Epochs*

Fig20- only zeroes – Confusion Matrix (Correct and Incorrect Classifications)

The next step was to remove features where there are all ones from the original dataset. This time the prediction rate dropped to 85% and the total number of true values detected was 104. This method reduces features to 300 features.

The last feature selection test was to remove features where there are all ones and zeroes. This method reduced number of features to 267. This significantly improved the computation time as well as prediction score. I was able to improve the prediction to 94% . The total number of true values detected was 125. The below graph shows that prediction error for training, validation, and test are very similar with each other. Looking at the confusion matrix, we can see that prediction has been improved to 0.4% true values predicted.

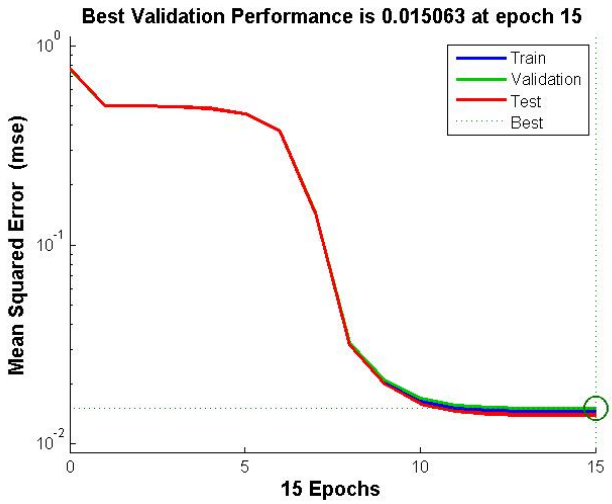


Fig21-combined 0s and 1s-Error vs Epochs

Fig22- combined 02 and 1s – Confusion Matrix (Correct and Incorrect Classifications)

We also tried to only remove the features where the percentage of zeroes in a feature is greater than 98%. But this method didn't work well as expected. The prediction rate was very low (around 50%) and 76 true values were predicted.

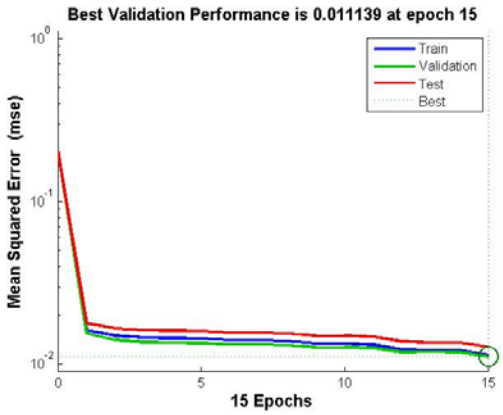


Fig18-0s columns over 98% -Error vs Epochs

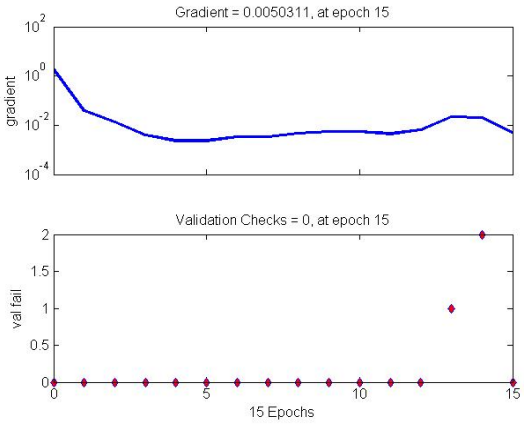


Fig19- 0s columns over 98% - Gradient vs Epochs

Fig25- 0s columns over 98% - Confusion Matrix (Correct and Incorrect Classifications)

Here is the summary of the results using feature selection:

	Prediction rate (training set)	Final Score
No feature selection	43%	67
Remove ones only	85%	104
Remove zeroes only	90%	118
Remove zeroes if > 98%	50%	76
Remove ones and zeroes	94%	125

Table4-feature Selection result

3.3.2 Row selection

Since we have lots of rows in our training dataset, it will make our problem much simpler if we can remove some of the rows from our dataset. So initial approach was similar to what we had done during feature selection. We tried to remove rows where there is all zeroes or ones. Unfortunately there wasn't a single row which satisfy this condition. We then defined a threshold. Initially we removed rows where the threshold of having zeroes is greater than 98%. The result wasn't really impressive. Only 20 true values were predicted from evaluation dataset. Then we tried the same approach to remove rows numbers of ones in the row is greater than 98%. Again the true value detection wasn't very good; it was around 17.

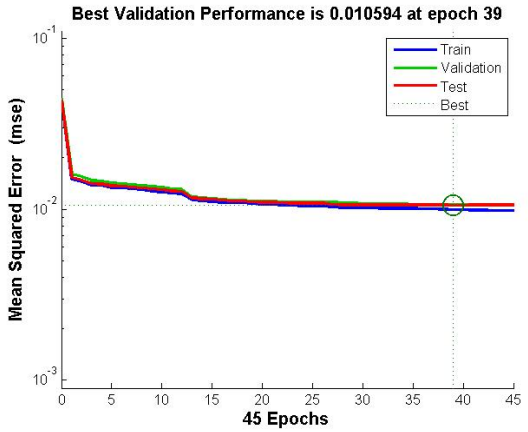


Fig26-row selection –Error vs Epochs

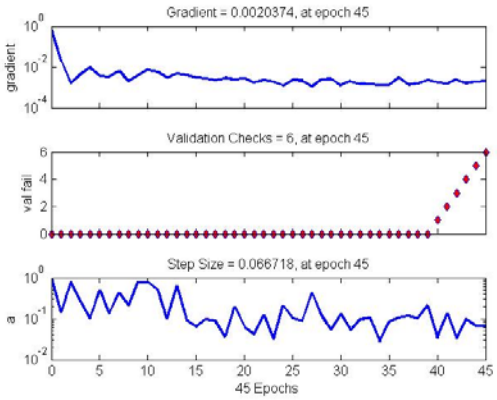


Fig27- row selection – Gradient vs Epochs

Fig28- row selection – Confusion Matrix (Correct and Incorrect Classifications)

So we decided to combine the two methods mentioned above, remove rows where number of ones exceeds 98% or number of zeroes exceeds 98%, in order to produce a better result just like the feature selection which we used in previous improvement method. Indeed, the result was higher than the two methods mentioned above but it was not as good as feature selection result. With this combined method, 30 true values were predicted which compare to 125 obtained from feature selection is not very impressive.

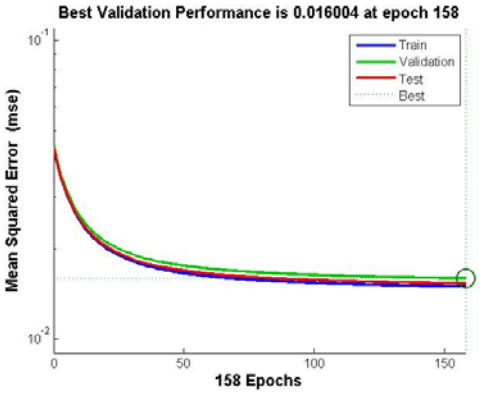


Fig29-row & column cobined –Error vs Epochs

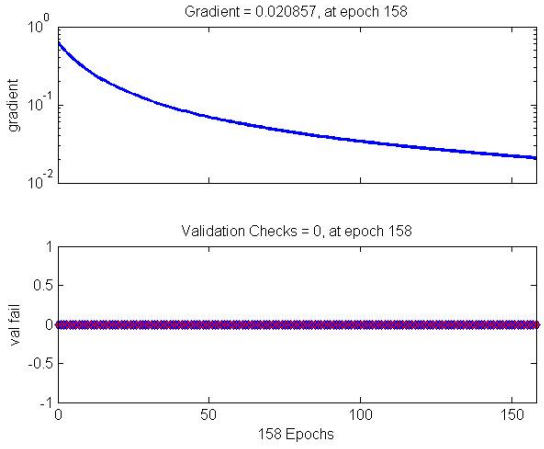


Fig27- row & column combined - Gradient vs Epochs

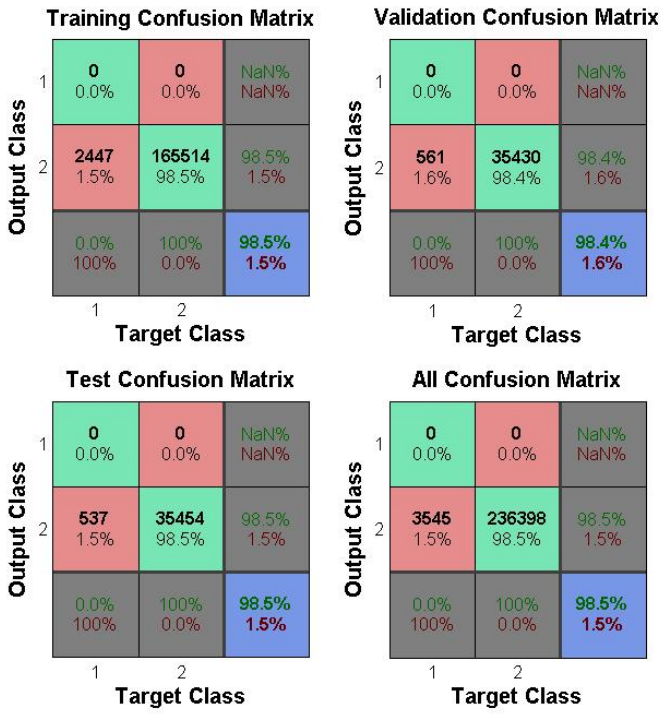


Fig31- row & column combined – Confusion Matrix (Correct and Incorrect Classifications)

3.3.3 Change learning algorithm\mebership function type

Here we tried to use some different learning algorithm as well a membership functions in order to obtain a better prediction score for our Neuro Fuzzy classifier. As mentioned earlier, the first Neuro Fuzzy classifier, used 50 hidden neurons with 5 triangular member ship functions and reduced fuzzy rules as its input. Now in the second Neuro fuzzy classifier, we used uses trapezoid membership functions with momentum as its learning method. We trained our network on training data set after applying feature selection method as mentioned above. Increasing the number of hidden neurons to 79 gave us the best result. The prediction score with 79 hidden neurons and 7 trapezoid membership function was 137. Here is the summary table of using trapezoid membership functions with momentum learning method

Hidden layer	memFunc type	#of memFunc	Prediction rate	MSE	Final score
50	Trapezoid	5	43%	0.83	67
55	Trapezoid	5	52%	0.54	77
67	Trapezoid	7	71%	0.33	112
79	Trapezoid	7	94%	0.034	137
92	Trapezoid	7	86%	0.076	121
100	Trapezoid	7	73%	0.103	116

Table5-Momentum learning with trapezoid membership functions

The next classifier we tried, it uses SCG as its learning method. When we used neural network for classification, SCG learning method gave us better result compare to momentum method. Similarly, when we ran the classifier, with 7 membership function (Trapezoid membership function), the result wasn't really impressive. We were only able to detect 107, true values. So next step was to change the actual shape of the membership function to see if there will be any improvement. When we change the shape of membership function from trapezoid to Gaussian, we saw an increase in prediction; 118 true values were detected. Next, we reduced the number of membership functions to 5 and we obtained a much better result. With this setup, 132 true values were predicted. The last classifier, is just an improvement to the previous classifier. The last classifier is much more efficient in terms of computation using SCG learning algorithm. This classifier uses batch method to update network's weights. However, the result were very similar to the previous classifier. Here is the summary of the 4 different classifier result with different configuration:

	Learning Alg	Hidden layers	MemF type	# of MemF	MSE	Score
Classifier 1	backpropagation	50	Triangular	7	0.082	87
Classifier 2	momentum	50	Triangular	7	0.062	96
Classifier 3	SCG	50	Triangular	7	0.021	114
Classifier 4	SCG batch	50	Triangular	7	0.034	109

Table6-classifier comparison using triangular membership functions

	Learning Alg	Hidden layers	MemF type	# of MemF	MSE	Score
Classifier 1	backpropagation	67	Trapezoid	7	0.052	102
Classifier 2	momentum	67	Trapezoid	7	0.032	108
Classifier 3	SCG	67	Trapezoid	7	0.0110	121
Classifier 4	SCG batch	67	Trapezoid	7	0.0124	118

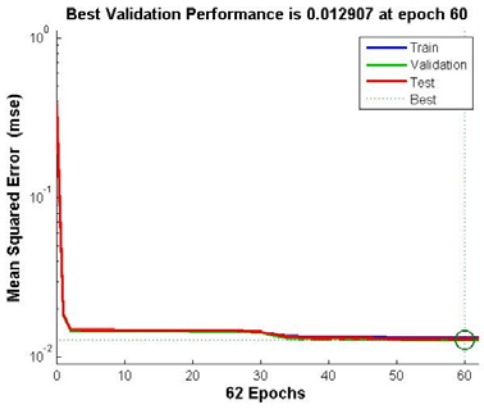
Table7-classifier comparison using trapezoid membership functions

	Learning Alg	Hidden layers	MemF type	# of MemF	MSE	Score
Classifier 1	backpropagation	79	Gaussian	5	0.01297	102
Classifier 2	momentum	79	Gaussian	5	0.01055	108
Classifier 3	SCG	79	Gaussian	5	0.010126	148
Classifier 4	SCG batch	79	Gaussian	5	0.010197	132

Table8-classifier comparison using Gaussian membership functions

Here are performance, gradient, and confusion matrices of all 4 classifier explained above.

Classifier 1:



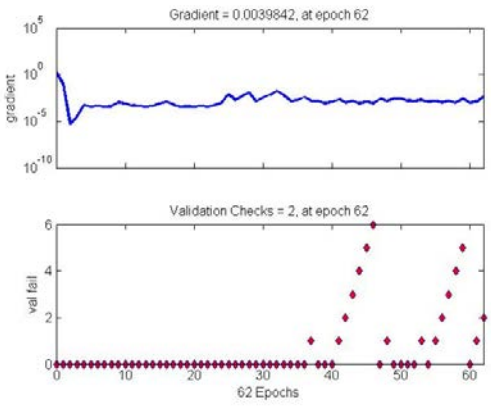


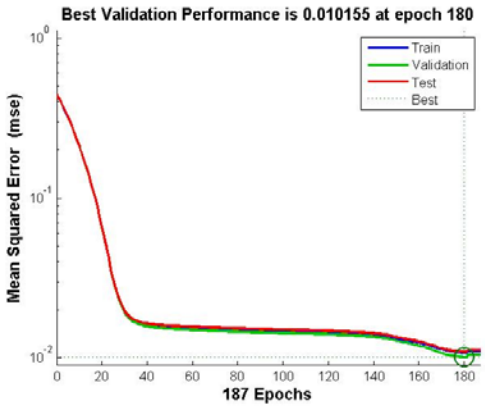
Fig32-IBM classifier 1 -Error vs Epochs

Fig33- IBM classifier 1 - Gradient vs Epochs

Fig34- IBM classifier 1 - Confusion Matrix (Correct and Incorrect Classifications)

Even though the performance of overall networks looks pretty reasonable but prediction of the True values is still very low (Confusion matrix shows a lot of false positive). Also it seems that we have over trained the network since there is an increase in the gradient of the network.

Classifier 2:



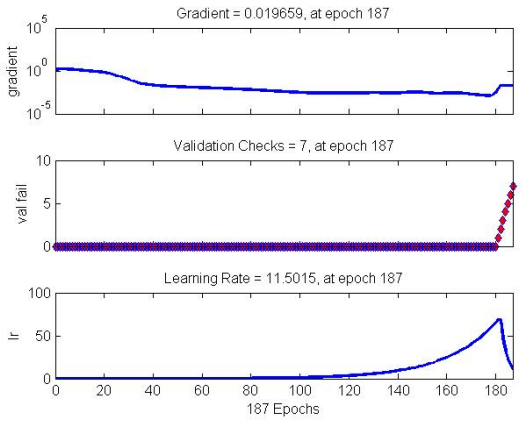


Fig35-IBM classifier 2 -Error vs Epochs

Fig36- IBM classifier 2 - Gradient vs Epochs

In this classifier Gradient looks more stable than the previous classifier, but the momentum methods perform slightly as good as the previous backgropagation classifier.

Classifier 3:

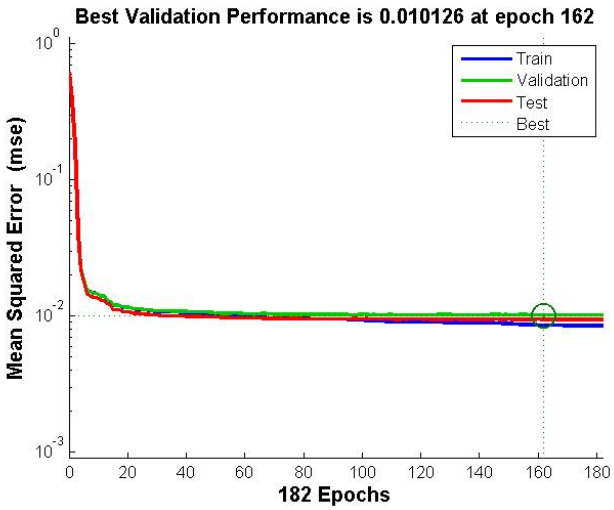


Fig37-IBM classifier 3 -Error vs Epochs

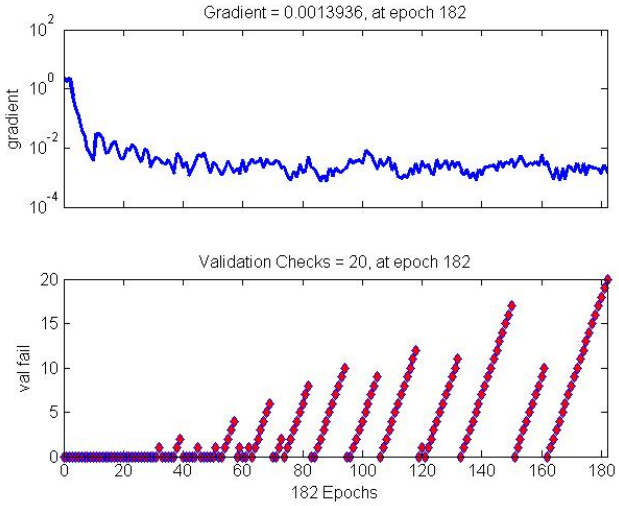


Fig38- IBM classifier 3 - Gradient vs Epochs

Fig39- IBM classifier 1 - Confusion Matrix (Correct and Incorrect Classifications)

This classifier uses “trainscg” which significantly improves the prediction score. The graphs above proves it. The prediction of false positive has decreased as shown in the confusion matrix and gradient has reached its minimum value compare to all other classifiers.

Next classifier uses a batch training of the third classifier. The result of the fourth classifier is slightly lower to the third classifier. We have concluded that online training works the best for IBM Great Mind Challenge.

Classifier 4:

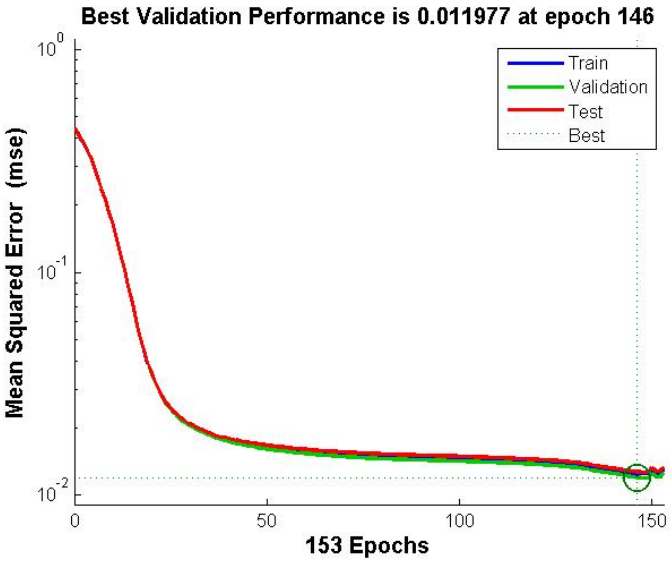


Fig40-IBM classifier 4 -Error vs Epochs

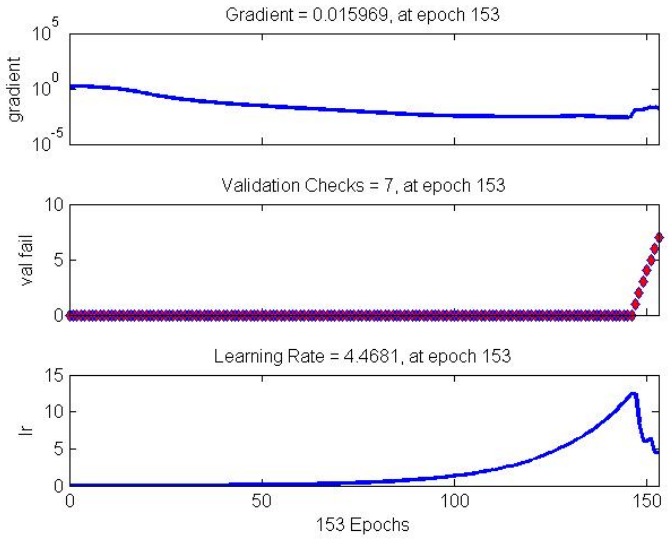


Fig41- IBM classifier 4- Gradient vs Epochs

Finally IBM great mind challenge provided us with a completely new dataset and now this time we are given only one chance to submit. So we decided to use the neuro-fuzzy with 5 Gaussian membership functions and SCG learning. When Neuro fuzzy classifier was ran against the final evaluation dataset, 140 true values were predicted which is still close to the 148 true values that we detected using old evaluation dataset. This result put our team in the fourth place of the competition after they announced the top three winners. Here is the screen shot of the top three winners of the completion. So the score we obtained is only three points away from the third place team.

3.4 Neuro Fuzzy in the IARPA Trustworthiness Challenge

3.4.1 Data Preparation

We divided the four class labels in our training data into two classes as follows:

Promised belongs to the “trust” class

Promised not fulfiliable belongs to the “don't trust” class.

More than promised belongs to “trust” class

Less than promised belongs to the “don't trues” class

Furthermore “trust” class is represented with number 1 and “don’t trust” class is 0 in our dataset. There is also a feature B-ALS which is presented by labels in the training dataset. The labels are low, medium and high. This feature is the signal to identify if the person is at risk. So we converted these three labels into 0, 50 and 100 respectively. Now the training dataset is ready to be fitted into the Neuro Fuzzy classifier. In the next section, we are going to run the classifier against the evaluation dataset and analyze the result.

3.4.3 Trustworthiness challenge improvements and results

3.4.3.1 Applying Neuro Fuzzy classifier on evaluation set

So when we ran the fuzz-neural network against training dataset, the score obtained was -0.03. This score is based on how well we predicted the trust and don’t trust label in the evaluation dataset. Basically, we need to run our trained network against evaluation dataset (new dataset, separate than training dataset) and for each row in our evaluation dataset, we need to predict whether it belongs to “trust” or “don’t trust” class. I used neuro fuzzy system with 3 triangular membership functions and 57 hidden neurons. The network error is increasing as we train the network. This explain why we have such low score in our prediction.

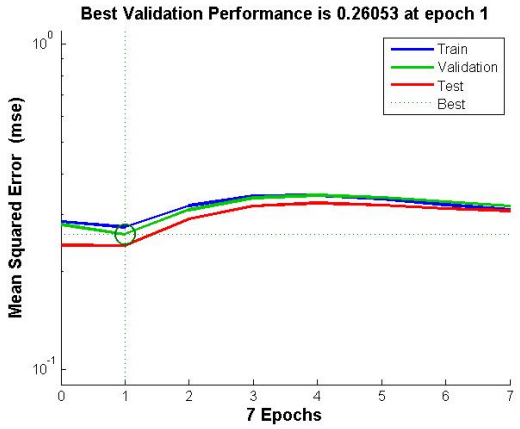


Fig42-Neuro Fuzzy 3 memFunc -Error vs Epochs

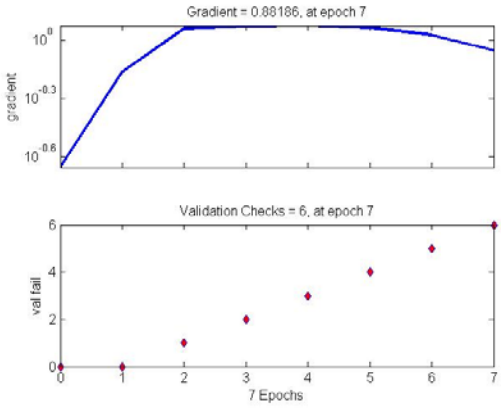


Fig43- Neuro Fuzzy 3 memFunc - Gradient vs Epochs

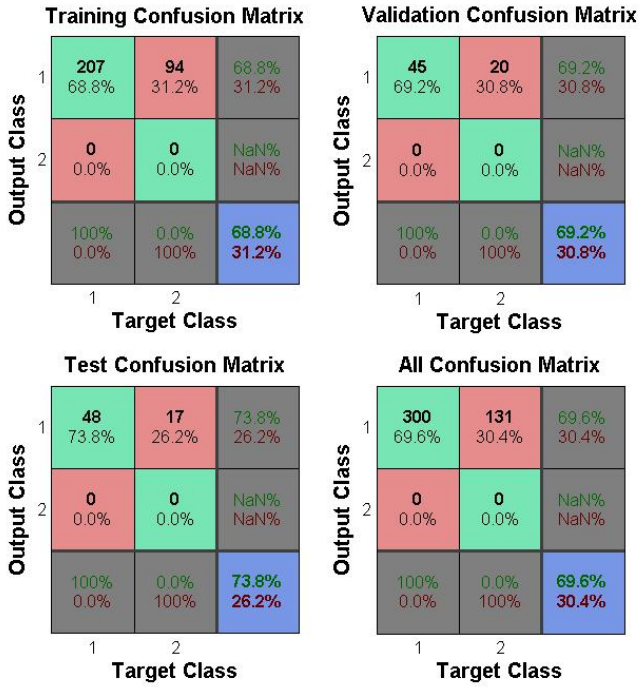


Fig44- IBM classifier 1 - Confusion Matrix (Correct and Incorrect Classifications)

3.4.3.2 Change number of classes

Previously, in our training dataset, we had four different vales to determine the classes; exact amount promised, more than promised, less than promised, promise not

fulfillable. In the previous approach, we decided to associate “exact amount promised” and “more than promised” to the class “trust” by replacing it with value 1 in our training dataset. Similarly we associated class “don’t trust” to “less than promised” and “promise not fulfillable” by replacing them with 0 in our tarring dataset.

Now in the new approach, we give each of these four cases a special number which represent a new class in our training. So now our training dataset contains 4 distinct classes. We give value 1.5 to more than promised, 1 to exact amount promised, -1 to less than promised and -1.5 to promised not fulfillable. We trained our Neuro Fuzzy network on these 4 classes. In order to classify after we ran our network on evaluation dataset, we need to map “more than promised” and “exact amount promised” to “Trust” class. Similarly we mapped “promised not fulfillable” and “less than promised” to “Don’t trust” labels.

We tested our Neuro Fuzzy network against this new approach. Since we are only limited to 7 submission pre week, we need to submit the solution which we believe it would give us the best solution. In order to determine the best solution, we heavily rely on the MSE of our network. We submitted a solution where our network had a very low MSE; around 0.13896. The score of the submission was 0.02 which was higher than what we had obtained previously.

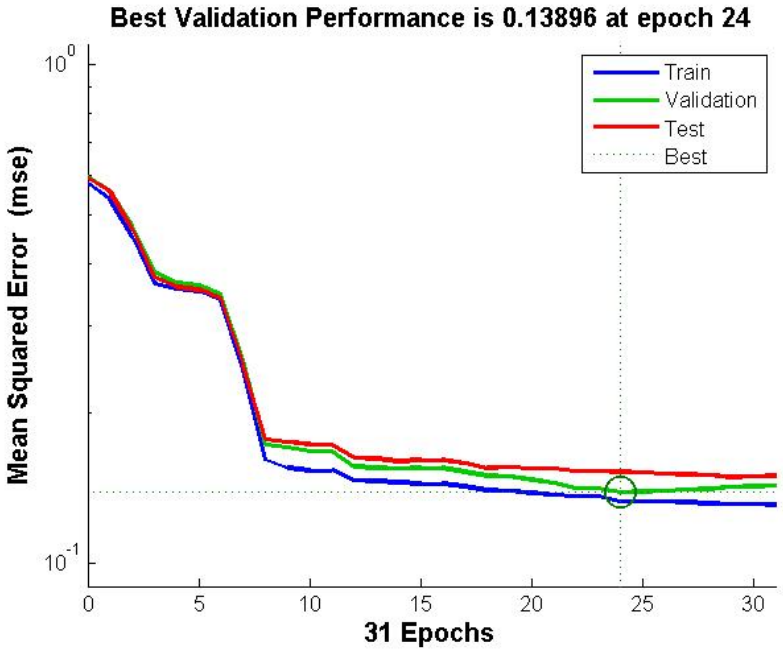


Fig44-4 classes dataset -Error vs Epochs

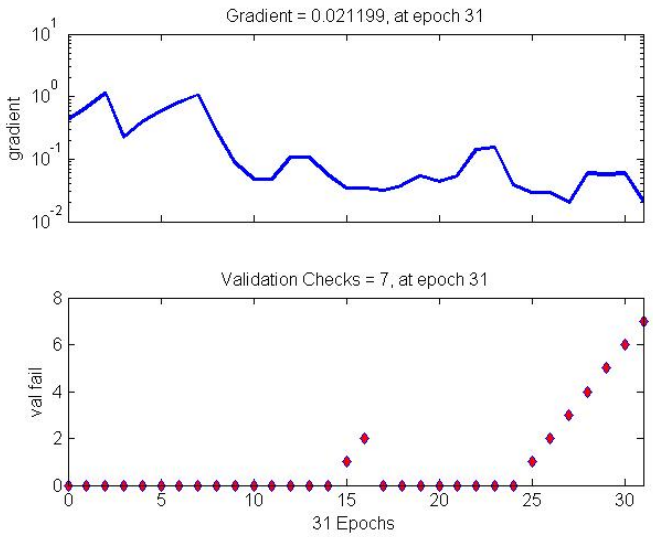


Fig45- 4 classes dataset -Gradient vs Epochs

Fig46- 4 classes dataset - Confusion Matrix (Correct and Incorrect Classifications)

Even though, nework error and gradient are decreasing, but we have completely failed to detect one classe in our confusion matrix which s “promise not fulfilable.”

3.4.3.3 Change network parameters

Originally we used 3 triangular membership functions. That setup didn’t give us a good score; in fact it generated a negative score. So we decided to change the membership function into 3 trapezoid functions. It slightly improved the score. The score wasn’t negative anymore but the overall prediction score was around 0.03. We then focused more on the neural network side of the neuro-fuzzy system. We tried to change some parameters such as evaluation checks, random seed value, number of hidden layers and number of iterations. The score improved when number of hidden layers are between 50 to 60 and validation checks are at minimal. The score was improved to 0.14. Here are some tables which summarizes the result of changing some network parameters.

	Learning Alg	Hidden layers	MemF type	# of MemF	MSE	Score
Classifier 1	backpropagation	79	Gaussian	5	0.793	0.02
Classifier 2	momentum	79	Gaussian	5	0.701	0.07
Classifier 3	SCG	79	Gaussian	5	0.473	0.12
Classifier 4	SCG batch	79	Gaussian	5	0.514	0.09

Table8-classifier comparison using Gaussian membership functions on Trust datatset-79 hidden

	Learning Alg	Hidden layers	MemF type	# of MemF	MSE	Score
Classifier 1	backpropagation	100	Gaussian	5	1.04	-0.02
Classifier 2	momentum	100	Gaussian	5	0.957	0.013
Classifier 3	SCG	100	Gaussian	5	0.831	0.062
Classifier 4	SCG batch	100	Gaussian	5	0.846	0.058

Table9-classifier comparison using Gaussian membership functions on Trust datatset-100 hidden

	Learning Alg	Hidden layers	MemF type	# of MemF	MSE	Score
Classifier 1	backpropagation	56	Gaussian	5	0.320	0.07
Classifier 2	momentum	56	Gaussian	5	0.199	0.14
Classifier 3	SCG	56	Gaussian	5	0.199	0.36
Classifier 4	SCG batch	56	Gaussian	5	0.224	0.21

Table10-classifier comparison using Gaussian membership functions on Trust datatset-56 hidden

Classifier 1:

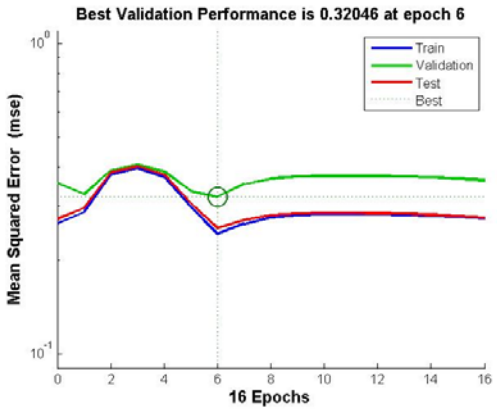


Fig47-Trust Classifier 1 -Error vs Epochs

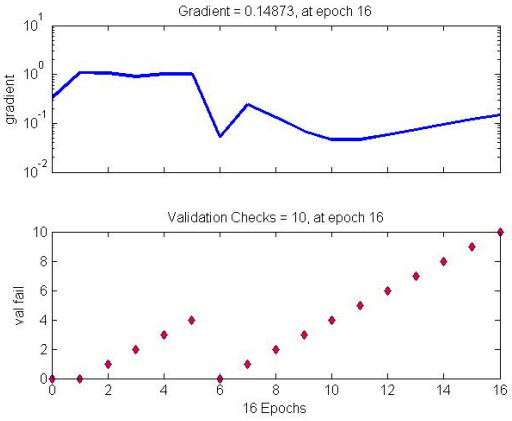


Fig48- Trust Classifier 1 - Gradient vs Epochs

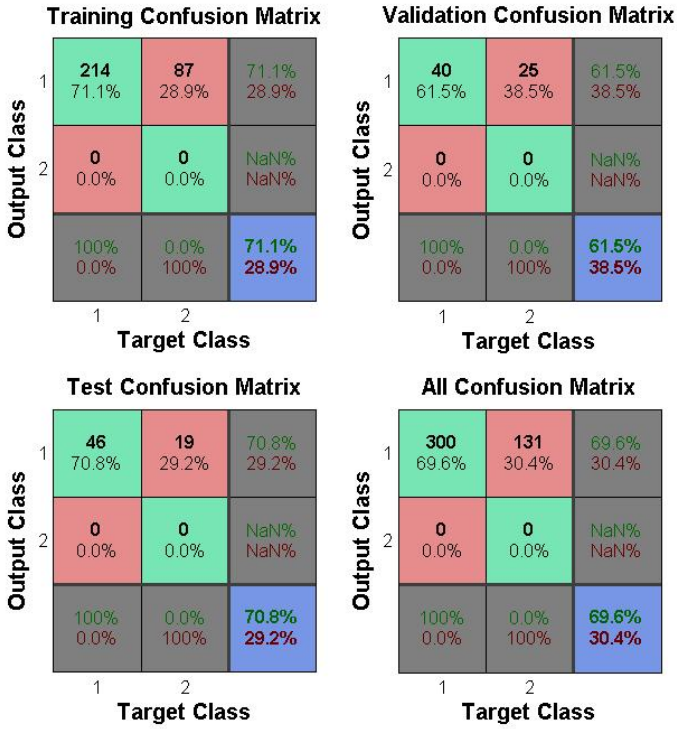


Fig49- Trust classifier 1 - Confusion Matrix (Correct and Incorrect Classifications)

According to the confusion matrix, we have failed to detect “Don’t trust” class in our training, test, and validation dataset. Notice that how network performance, and gradient increase over network training time. This explain low prediction score that we have for this method.

Classifier 2:

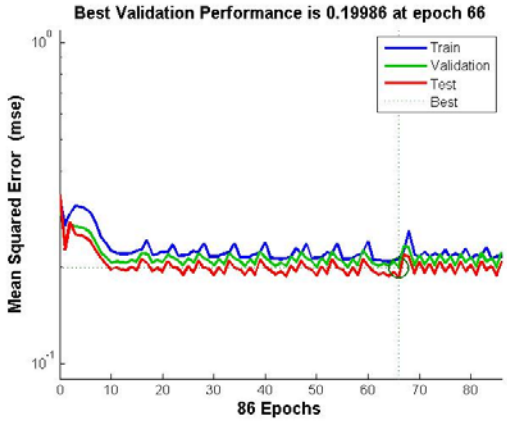


Fig50-Trust Classifier 2 -Error vs Epochs

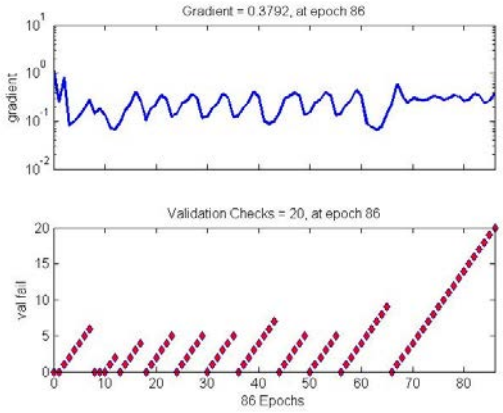


Fig51- Trust Classifier 2 - Gradient vs Epochs

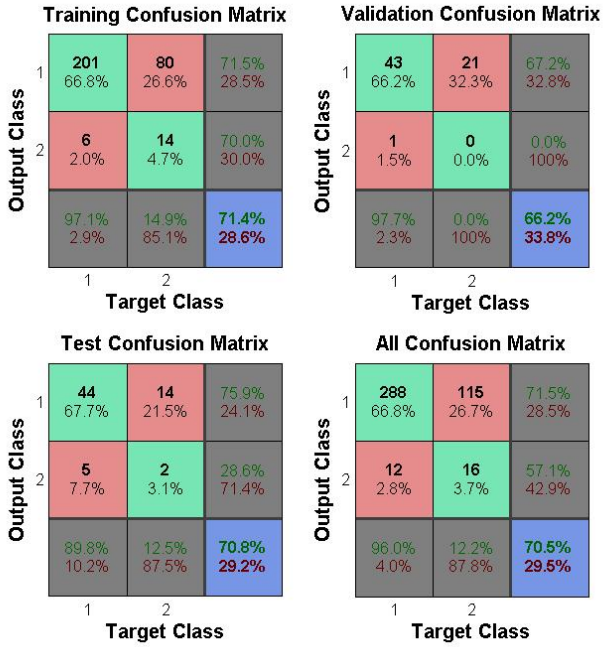


Fig52- Trust classifier 1 - Confusion Matrix (Correct and Incorrect Classifications)

Here we have improved our prediction over “Don’t trust” class. We have also almost stable network performance and gradient. So this momentum method slightly improved our prediction score.

Classifier 3:

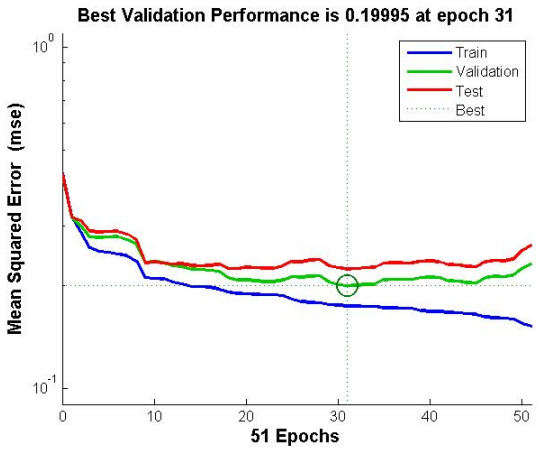


Fig53-Trust Classifier 3 -Error vs Epochs

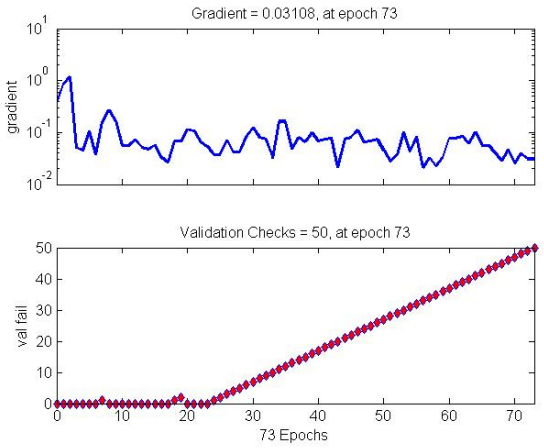


Fig54- Trust Classifier 3 - Gradient vs Epochs

Fig55- Trust classifier 3 - Confusion Matrix (Correct and Incorrect Classifications)

Once again, “trainscg” gave us the best prediction score compare to other classifiers used against this dataset. Overall 7% error rate in “don’t trust” classes is better compare to the overall error (false positive) in other classifiers.

Classifier 4:

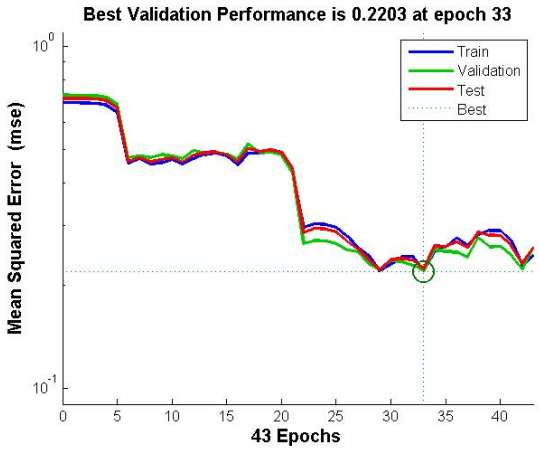


Fig56-Trust Classifier 4 -Error vs Epochs

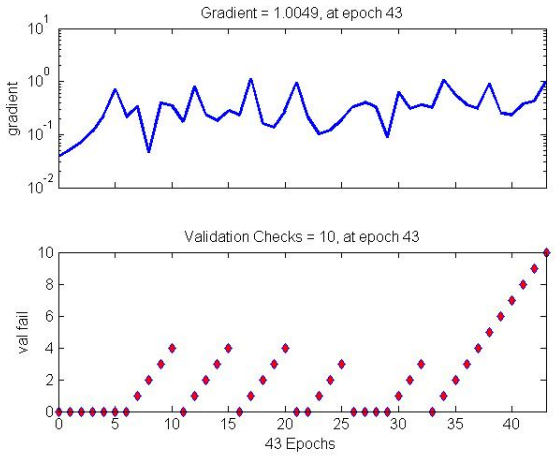


Fig57- Trust Classifier 4 - Gradient vs Epochs

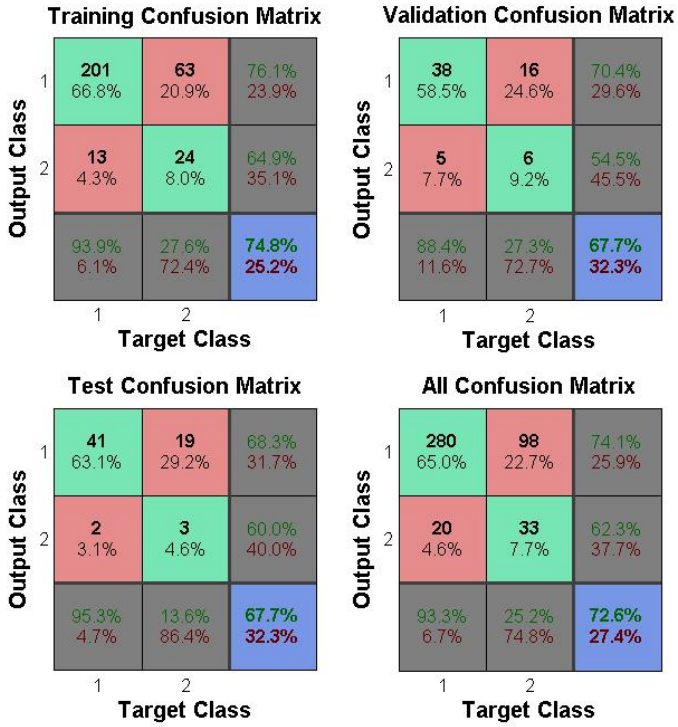


Fig58- Trust classifier 4 - Confusion Matrix (Correct and Incorrect Classifications)

3.4.3.4 Retrain the network on its weakness

Since we were not improving by changing many different parameters as mentioned in the previous improvements, we started focusing on the root cause of this issue. In order to find the issue, we tried to analyze the training dataset. In other words, we wanted to see how well we can predict the entire training set. So the goal is to train

the Neuro Fuzzy classifier on the entire training dataset, and then use the same training set in order to predict the outcome. We noticed that, we were not able to fully predict the training set. Specifically, the prediction of “don’t trust” class was extremely poor; around 40%. This explains the low prediction score that we have on the evaluation dataset.

So we tried to improve the prediction rate on the training dataset by using different learning algorithms. Only SCG learning algorithm gave us the highest prediction score. Also changing the fuzzy parameters in the network didn’t really help much. So the last option we tried was to retrain the network on the set of data were we predicted wrong class labels. We noticed that 121 rows in our training dataset will always misclassified. So we retrain our network on these 121 rows.

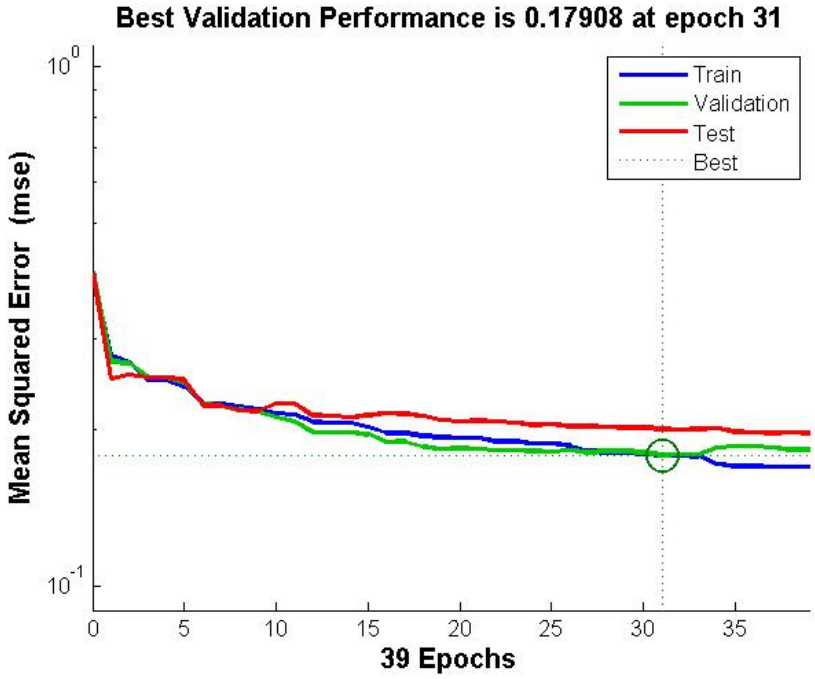


Fig59-Retarain Classifier -Error vs Epochs

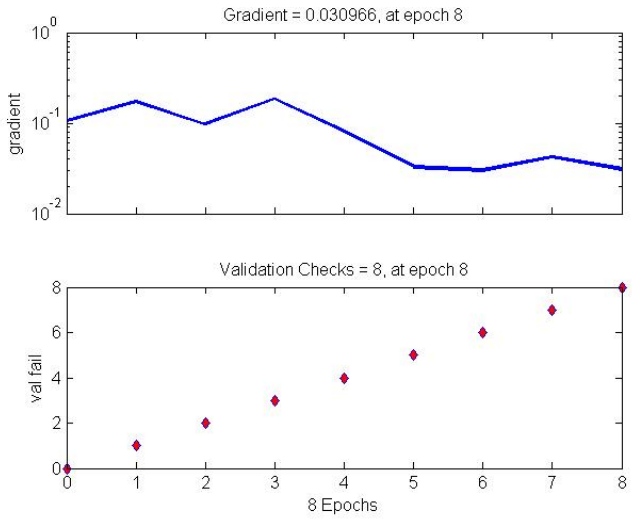


Fig60- Retrain Classifier - Gradient v Epochs

Fig61- Retrain classifier- Confusion Matrix (Correct and Incorrect Classifications)

Here we can see that MSE, performance and gradient of the network is decreasing as we train the network. We retrain the network on the data where it failed to predict using the same learning algorithm (trainscg) and same network parameters. This shows an impressive in the network prediction score as well as total number of errors for

detecting “Don’t trust” labels (only 2%). The prediction score was at 0.21. The prediction is still not very high and this could be due to our network become bias toward the 121 rows that we retrained on.

3.4.3.5 Changing the threshold

The last improvement was to find an optimal threshold for our classifier. We noticed that some output values of our network is not very close to 0 or 1 which indicate “trust” and “don’t’ trust” class respectively. Some values falls between 0 and 1. So we need a threshold to determine a boundary between “trust” and “don’t trust” class.

The following table shows the different threshold used in order to decide the best boundary for separating trust and don’t trust classes.

Threshold	Score
0.5	0.18
0.623	0.36
0.7	0.19
0.85	0.13
0.65	0.26
0.59	0.33
0.61	0.56
0.65	0.28

So according to this table threshold of 0.61 gave us the best prediction score of 0.59.

We have entered in this competition as a team. So our team score for this competition is 0.72. The top 25 teams which have score greater than 0.70 are eligible to participate in the final round. At this stage, teams are required to submit their solutions along with their code and their report. There are awards for the top three teams of this competition.

4. Conclusion:

In this paper, we presented three classifier on IBM Great Mind Challenge Dataset and Innocentive Trustworthiness dataset; Reduced Weighted Fuzzy Classifier, Neural Network Classifier, and Neuro Fuzzy Classifier. As we discussed, we obtained a relatively low score using reduced fuzzy classifier. The problem were mostly involved in the way fuzzy rules were generated and how weights are associated to the fuzzy rules. We then decided to use neural network to compare the result with our fuzzy classifier. Many different algorithms on Matlab Neural Network were put into practice. The result of each learning method were recorded for further analysis. We have concluded that Neural Network will give us much higher score compare to Fuzzy Classifier. Fuzzy logic cannot learn but it is really powerful in making decision based on imprecise and ambiguous data. On the other hand, Neural Network have the learning ability but are not as powerful as fuzzy logic in terms of making decision. So combining these two method could give us a very powerful classifier. A homogeneous artificial intelligence algorithm that uses both Neural Network and Fuzzy Logic is called Neuro-fuzzy system. As we saw in experiment result, We have improved our prediction score by using Neuro Fuzzy System. In both IBM Great Mind Challenge and Innocentive Trustworthiness challenge, we wre eable to finish in the top ten. IN IBM Great Mind Challenge, out of 55 participant teams, we ranked 4^{th}. In Innocentive Challenge, out of 453 teams, we ranked 8^{th} in the overall competition.

5. References:

[1] *Yung-Chou Chena., Li-Hui Wangb., Shyi-Ming Chen.: Generating Weighted Fuzzy Rules from Training Data for Dealing with the Iris Data Classification Problem (2006).*

[2] *Castro, J. L., Castro-Schez, J. J., and Zurita, J. M. 1999. Learning maximal structure rules in fuzzy logic for knowledge acquisition in expert systems. Fuzzy Sets and Systems, 101, 3: 331-342. (1999)*

[3] *Castro, J. L. and Zurita, J. M. 1997. An inductive learning algorithm in fuzzy systems. Fuzzy Sets and Systems, 89, 2:193-203. (2001)*

[4] *Bulbul, A. A Short Fuzzy Logic Tutorial. (2010)*
http://www.calvin.edu/~pribeiro/othrlnks/Fuzzy/home.htm

[5] Sun, C.-T., Jang, j.-s.*: A neuro-fuzzy classifier and its applications (1993)*

[6] *Gabrys, b.: Combining neuro-fuzzy classifiers for improved generalisation and reliability (2002)*

[7] *gabrys, b.: Learning hybrid neuro-fuzzy classifier models from data: to combine or not to combine (2001)*

[8] *Fuller, b.: Neuro-Fuzzy Methods. (2001)*
http://uni-obuda.hu/users/fuller.robert/dam.pdf

[9] *Reby, D, Lek, S, Dimopoulos, I, Joachim, J, Lauga, J, Aulagnier, S.: Artificial neural networks as a classification method in the behavioural sciences*

[11] *Zhou, Z., Chen S., Chen Z.: FANNC: A Fast Adaptive Neural Network Classifier (2000)*

[12] *Hybrid Intelligence Systems. (2004)*
www.computing.surrey.ac.uk

[13] *Negnevitsky, M. Hybrid Neuro-Fuzzy Systems. (2013)*

http://www.iaria.org/conferences2013/filesINTELLI13/INTELLI%20Keynote_Negnevitsky.pdf

[14] *Negoita, M., Neagu, D., Palade, V.: Neuro-Fuzzy Integration in Hybrid Intelligent Systems (2005)*