

# **Classification Project**

## **Project I**

### **CSE 574 - Machine Learning**

**Achint Oommen Thomas**  
**Dept. of Computer Science and Engineering**  
**State University of New York at Buffalo**

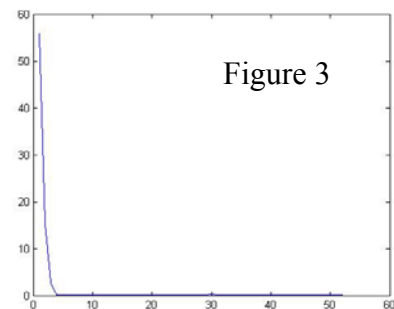
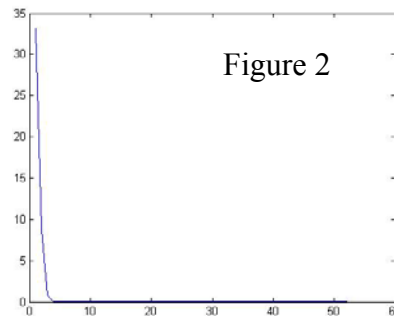
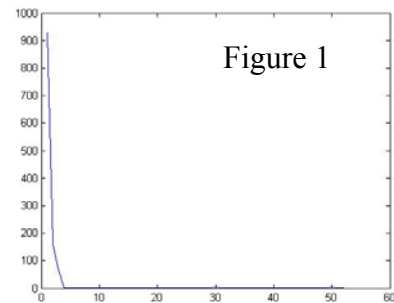
## Introduction

This project involves a classification task on a 2-class problem. The objective was to build the best possible classifier to classify the data with highest accuracy on an unseen test dataset. This report is divided into the following sections. Section I deals with an analysis of the data and on preprocessing applied. Section II gives a discussion on what model was chosen and the reason that choice was made. Section III gives a detailed description of the chosen model and the model parameters. Section IV gives a discussion on the observations made during the implementation of this project. Section V is a list of references used during the course of this project.

## Section I: Analysis of the Dataset

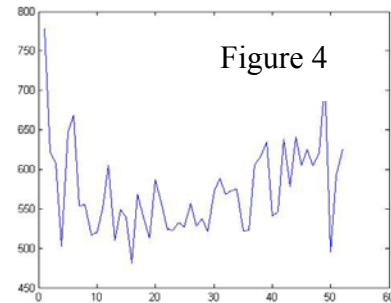
The dataset is 1000 rows by 53 columns. The rows represent each separate datapoint and the columns give the different dimensions. Dimension number 53 is the classification label that may be used in case a supervised learning approach is used. The labels have only 2 distinct values (0 and 1) making this a 2-class classification problem. The fact that we have labels provided, suggests that we use a supervised learning algorithm like Decision Trees or Neural Networks.

Figure 1 shows the maximum of the values for each dimension. On analysis, it was found that the minimum value in each dimension is 0. We see that the data has large ranges in certain dimensions as compared to others. Figure 2 gives of the mean of the data over each dimension. Figure 3 gives the standard deviation. We again see that these quantities vary across the dimensions and as such the data is not centered about a common point or constrained to have a particular standard deviation. We can therefore think about performing some form of normalization on the data. The method of normalization chosen was to constrain the data to have zero mean and unit standard deviation [1]. Figure 4 shows the number of unique values present in each dimension of the dataset. We can make a decision on model selection using this information. This will be seen in Section 2.



## Section II: Model Selection

We see that the dataset given to us is in the form of training examples with classification labels. Since we have the labels, we can think of using a supervised learning technique. Unsupervised learning techniques like Clustering (K-Means, Fuzzy C-Means) and K-Nearest Neighbour methods usually give lower accuracies and are used primarily in the case when we do not have labeled data. Hence, we can decide on using either Decision Trees or Neural Networks as classifiers.



If we observe Figure 4, we see that each dimension has a large number of unique values. If we choose to use a Decision Tree as a classifier we will have to decide on a method of discretising the data so as to maximize an information gain measure that we specify. It is difficult to achieve this discretization since we do not know what physical attributes the dimensions represent. On the other hand, if we choose to use a Neural Network and use the Back Propagation Algorithm, we do not have to worry about such factors. Thus, a Neural Network implementing Back Propagation Algorithm, classifier model was chosen.

## Section III: Description of Chosen Method

When using a Neural Network as a classifier, a number of different model parameters need to be decided upon. These include the number of hidden layers, number of neurons in each hidden layer, the learning rate, the transfer functions for the neurons and the split up of the training data into training and validation sets. The number of neurons in the input and output layers are dictated, in part, by the nature of the problem.

### Algorithm and Network Architecture

The Neural Network I have designed for this project is a fully connected Feed Forward Network. It consists of 1 hidden layer in addition to 1 input layer and 1 output layer. The input layer consists of 52 neurons, the hidden layer has 5 neurons and the output layer has 1 neuron. The reason for this is described in the Model parameters section below.

Back Propagation Algorithm has been used as the learning algorithm for this network. The algorithm consists of the following steps

- Weight Initialization
  - o The weights for each neuron in the network have to be initialized to small values. This is because, large initial weights will drive the neurons into saturation and the learning process will slow down [1].
  - o I have used the Matlab function `rand()` to perform the weight initialization. This function is specially written to initialize the weights

symmetrically for use by neural networks. The bias weights are also initialized using this function.

- Perform Forward Pass of Algorithm
  - o Calculate the activation to each neuron based on its weights and inputs
  - o Calculate the outputs of each neuron based on the transfer function used for that neuron. This will be elaborated below in the Model Parameters section.
  - o Round the outputs to make it have value either 0 or 1.
- Calculating the objective function  $G(w)$  and errors
  - o Since I have used the standard sigmoidal transfer function for the output layer neurons, the objective function is defined by
    - $G(w) = -\sum \sum (y_n^{(i)} \log(t_n^{(i)}) + (1 - y_n^{(i)}) \log(1 - t_n^{(i)}))$  ----- (3)
  - o The error is calculated by
    - $E_n = y_n - t_n$  ----- (4)
- Calculating Weight Updates
  - o The weight update for the hidden to output layer weights is
    - $\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}}$  and
    - $\frac{\partial E}{\partial W_{ij}} = \varepsilon \delta_{(i)} y^{(i)}$  ;  $\delta_{(i)} = \partial \Psi(x) / \partial x$
    - $\varepsilon$  is the error between the outputs and the targets
    - $\Psi(x)$  is the transfer function used in neuron  $j$
  - o The weight update for the input to hidden layer weights is
    - $\Delta W_{ik} = -\eta \frac{\partial E}{\partial W_{ik}}$  and
    - $\frac{\partial E}{\partial W_{ik}} = \delta_{(j)} x^{(k)}$  ;  $\delta_{(j)} = -\partial \Psi(x) / \partial x \bullet \sum_{i=1}^I \delta_i W_{ij}$
  - o These updates are calculated in the following code in the project
 

```
dEbydoutsum = outputs-ttargets;
dEbydhidvals = dEbydoutsum*(hidout');
dEbydhidsum = dEbydhidvals.*hidvals.*(1-hidvals);
dEbydoutbiases = sum(dEbydoutsum);
dEbydhidout = hidvals'*dEbydoutsum;
dEbydhidbiases = sum(dEbydhidsum);
dEbydinhid = tdata'*dEbydhidsum;
```
- Performing Backward Pass of Algorithm
  - o The weight updates calculated in the previous step are used to update the weights for the network. These lines of code show how.
 

```
inhid = inhid - epsilon*dEbydinhid;
hidout = hidout - epsilon*dEbydhidout;
hidbiases = hidbiases - epsilon*dEbydhidbiases;
outbiases = outbiases - epsilon*dEbydoutbiases;
```

Reference material [1] and [5].

## Model Parameters

- The number of input neurons is equal to the number of inputs from the dataset that we are presenting the network. Since no dimensionality reduction has been performed on the data, the number of inputs is 52 and subsequently, the number of input neurons is also 52.
- The number of outputs depends on the number of classes we wish to predict. For this dataset, we are dealing with a 2-class problem. For a binary classification problem, we need just 1 output neuron. High output from the neuron denotes one class and low outputs denote the opposite class.
- The number of hidden layers is related to the classification accuracy of the network. More hidden layers capture more intricacies in the data. However, the weight update rules get more complicated to derive and implement as the number of hidden layers increase. As the network was built and tested using a single layer of hidden neurons, it was observed that respectable classification accuracies were being achieved. This suggested that a single hidden layer was sufficient to capture the nature of the input – output mappings and generalize over the unseen validation data. Occam's Razor mentioned in [2] suggests that we choose the simplest hypothesis that fits the problem. I did.
- The number of hidden neurons in the hidden layer is another parameter choice that affects the classification accuracy. Too low a number of neurons will not be able to capture the nature of the data. Too high a number will overfit the learned boundary on the training data and the generalization capability of the network will be low. It is desirable to select a value that is optimal. After running tests on the dataset with a few different values, I decided to use 5 hidden neurons since this number gives the highest accuracies averaged over a set of 5 runs.
- The learning rate affects the rate at which the network converges to a set of optimal weights. If the learning rate is set too high, the gradient descent in the weight space happens at too fast a pace and we may have a condition when the network oscillates between different signs for the gradients as it jumps around in the weight space. If the learning rate is too low, the network learns too slowly and may take a long time to converge to an optimal set of weights. For this network and dataset, I used a learning rate value of 0.00005. This was chosen as it provided a condition where the log likelihood keeps decreasing steadily at a constant rate and the network stabilizes to give high classification accuracies over the validation set.
- There are a number of transfer functions that can be used in a neuron to compute its output. One property that it must fulfill is of being differentiable everywhere [1]. Sigmoid transfer functions are widely used in the construction of neural networks. I have chosen to use the anti-symmetric sigmoid transfer function

$$\alpha * \tanh(\beta * act_j) \quad \text{-----} \quad (1)$$

where  $\alpha$  and  $\beta$  are parameters, values 1.7159 and 0.66667 [1]

$act_j$  is the activation of the  $j^{\text{th}}$  hidden neuron given by  $\sum w_{ij}^T x_{ij}$  where  $w_{ij}$  are the weights for the  $j^{\text{th}}$  hidden neuron and  $x_{ij}$  are its inputs

for the hidden layer neurons since it may result in the network learning faster [3]. For the output layer, I have chosen the normal sigmoid

$$\frac{1}{1 + e^{-act_j}} \quad \text{-----} \quad (2)$$

where  $act_j$  is the activation of the  $j^{\text{th}}$  hidden neuron given by  $\sum w_{ij}^T x_{ij}$  where  $w_{ij}$  are the weights for the  $j^{\text{th}}$  hidden neuron and  $x_{ij}$  are its inputs

as it compresses its output between 0 and 1. This is a convenient range since we can consider the output as a probability. Also, since the class labels in the validation set are 0 and 1, we can use the output value as a proximity measure to the class that datapoint must be assigned to.

I have used two separate algorithms, one with the tanh() transfer function in the hidden layer and another with the standard sigmoid in the hidden layer. I have saved 3 sets of weights. Two sets have been obtained by training using the tanh() algorithm and the third set has been obtained using the standard sigmoid algorithm. The final prediction script uses these 3 sets of weights and takes the maximum vote to decide on a classification.

- The number of iterations (epochs) that we allow the network to run, determines how the network weights converge. If we allow the network to run for only a few iterations, the weights may not converge to an optimal value. The epochs can be controlled by setting a limit on the iterations a network must complete or by stopping the algorithm based on the value of the validation or training error or other methods. I have chosen to stop the iterative procedure when the value of the objective function  $G(w)$  starts to increase. Normally, as training proceeds, the value of the objective function keeps decreasing and the network makes fewer errors on the training and validation sets. As the objective function value starts to increase however, the errors also start increasing. I have used 200000 as an upper bound on the epochs.
- Selection of training and validation sets is another issue that must be considered. The problem arises since we were given just a single dataset. We need to split up this dataset. Too many training examples may provide sufficient data to the network for it to learn optimal weights. However, this decreases the number of validation examples we will have to ascertain if the network can generalize well over data it has not seen. Too few training examples will mean the network will not have enough data to learn the classification boundary. We should also be careful not to use the same training

and validation sets in different runs of the network while searching for optimal weights. Randomizing the sets will give us a more unbiased estimate of the actual error rates. I have therefore, randomized the split every time. I have used 600 examples for the training set and 400 examples for the test set.

## Section IV: Discussion

During the course of this project implementation, the following observations were made

- Given the same set of inputs to the network and for the same architecture and model parameters like learning rate and epochs, accuracies vary over different trials. This can be attributed to the fact that the weight initialization is done in a random fashion for each run. Hence, we have the network moving in slightly paths in the weight space as it searches for an optimal set of weights.
- The learning rate is a sensitive parameter in designing a good network. Sometimes, it takes a lot of searching to finalize on good learning rate values.
- It was noted in [1] that the network could be made to perform better if the learning rate was made a dynamic rather than a static parameter. Hence, as one approach, we could vary the learning rate based on the value of the objective function. As we find that the network is converging and the jumps between the objective functions get smaller, we could lower the learning rate to avoid performing very large jumps and halting the iterative process. Another technique is to use different learning rates for different neurons. Since the output layer has higher local gradients than other layers, we could use lower rates here. Neurons with many inputs could have lower rates so as to maintain a similar learning rate overall.
- The survey in [3] gives some interesting results when neural networks are constructed in an optimal fashion basing the architecture and transfer function selection on the problem type. This approach could have been tried if time was available.

## Section V: References

- [1] Simon Haykin, "Neural Networks: A Comprehensive Foundation", Second Edition; Pearson Education, Inc. 1999; ISBN 81-7808-300-0.
- [2] Tom M. Mitchell, "Machine Learning"; McGraw Hill Companies, Inc. 1997; ISBN 0-07-115467-1.
- [3] Norbert Jankowski and Włodzisław Duch, "Optimal Transfer Function Neural Networks"
- [4] Richard O. Duda, Peter E. Hart and Dacid G. Stork, "Pattern Classification", Second Edition; John Wiley and Sons, Inc. 2001; ISBN 9814-12-602-0.
- [5] Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning Internal Representations by Error Propagation"; In Parallel Distributed Processing: Explorations in the Microstructure of Cognition; MIT Press, Cambridge, MA.