Project 4: Networks for Classification and Regression

Sarah Wilson SWI1S117@JHU.EDU
303-921-7225
Engineering Professionals Computer Science
Johns Hopkins University
Baltimore, MD 21218, USA

1. Introduction

Neural Networks are collection of algorithms that are taught the underlying patterns in a data set through a process of supervised learning. Neural Networks operate using a process that is similar to the human brain, they contain simple processing nodes, called neurons that are connected in layers. These layers contain weights that determine an output based on a given input. The basis of an understanding of Neural Networks actually starts with Linear Regression.

Linear Regression is a supervised Machine Learning algorithm that can be applied to Regression problems. Linear Regression assumes there is some sort of linear relationship between the input data and the output value that can be represented by a line or plane of best fit between the inputs and the output. Linear Regression is meant for continuous outputs, when Classification tasks are required (tasks that have outputs with class labels), then the concepts of linear regression can be applied, but instead the Sigmoid function is used, hence the name Logistic Regression. The sigmoid function is able to take classifiers that are not numeric and map them to values between 0 and 1. Linear and Logistic regression are the basic concepts that build the foundation for Neural Networks. Neural Networks can be thought of as a series of Logisitc regression tasks that happen across many layers and nodes. Neural Networks can also be augmented through the use of developing an Auto Encoder. The Auto Encoders are trained such that an output pattern to a network always matches the input pattern. The Auto Encoder process can be thought of a method to fine tune the weights feeding the prediction output from a Neural Network.

The problem statement presented in this paper is to understand an implement three major categories of networks; Linear, Feed-Forward Back Propagation and Feed-Forward Back Propagation with an Auto Encoder. For the Linear Network: a Linear Regression will be applied to Regression problems while a Logistic Regression will be applied to the Classification Problems.

The hypothesis of this report is that the Feed-Forward Network with an Auto Encoder will perform better than the Feed-Forward Back Propagation and Linear regression approaches on both the Classification and Regression problems. The reason for this hypothesis is that the Auto Encoder builds upon the Linear Regression and basic Neural Regression. Both Linear Regression and Neural Networks rely on updating weight based on the error between the predicted output and the actual output of a training set. The Auto Encoder improves upon this process by trying to learn and mimic the input pattern of the data and apply that to the output or predicted value. This extra encoding and decoding steps, serve to fine tune the weights feeding the neural network. The hypothesis is based

around the concept that fine tuned weights should out preform coarsely tuned weights in making predictions based on new input data to the network. The results from experiments ran on the provided data sets will be discussed and presented against the outlined hypothesis.

Section 1 has provided the introduction, problem statement and hypothesis in regards to three major learning algorithms that will be explored. Section 2 will provide an in-depth explanation of the algorithms. Section 3 will present the results obtained by the different algorithms. Section 4 will discuss the results that were obtained and compare them to the hypothesis that was outlined in the introduction. This report will conclude in Section 5 with a discussion of lessons learned and areas of possible future work.

2. Algorithms and Experimental Methods

Linear and Logistic Regression

Regression is an a problem solving approach in Machine Learning that models a target predictor based on independent variables or values. Linear Regression and Logistic Regression both are discriminant-based approaches that do not care about estimating the densities inside a particular class region, instead the discriminant-based approach aims to estimate the boundaries between the class regression. The output of Linear and Logistic regression models is a weighted sum of the input attributes or features. The magnitude of the weight applied against each feature, shows the importance of that feature towards the overall prediction. The weights are learned by the model during training. Linear Regression is used on Regression problems because the data that is being predicted is continuous. This Linear model will not work for Classification problems, as the output of the linear model would be a real value, when in Classification a class label is desired. Logistic Regression is used for Classification problems, as it maps the output of the weight sum presented in Linear Regression and maps it to a non-linear function that ensures the outputs are between 0 and 1. The function used to achieve this is the Sigmoid or Logistic function.

The objective of training in both Linear and Logistic Regression is to determine the weights that will be used in the final model. The weight coefficients can be determined using the process of Gradient Descent. The general procedure for applying Gradient Descent is to first calculate the prediction based on the current weight coefficients and then calculate new coefficients based off the error in the prediction. This process is repeated until the coefficients are deemed accurate enough. The weight convergence process used in this paper to determine if the prediction is accurate enough will be to a set number of iterations. After the set number of iterations through the weight update process, the weights will be deemed good enough. At this point the model can be considered trained and will be evaluated for performance on the test data set.

Linear and Logistic Regression Algorithm

For each Feature (j) in the Domain(d) of the Data Set:
    Set W_j to a random value between -0.01 and 0.01
For Each Step in the Number of Iterations for Convergence:
    For each Feature (j) in the Domain(d) of the Data Set:
        $\Delta W_j = 0$
    For each observation (t) in the Data Set:
        Set the Temp Prediction (o) to 0.
            For each Feature (j) in the Domain(d) of the Data Set:
                $(o) = (o) + W_j * X_j$
            If the problem is Regression:
                Set the Model Prediction (y) to the Temp Prediction Value (o)
            If the problem is Classification:
                Set the Model Prediction (y) to the Sigmoid(o)
            For each Feature (j) in the Domain(d) of the Data Set:
                $\Delta W_j = \Delta W_j + (r - y)X_j$
    For each Feature (j) in the Domain(d) of the Data Set:
        $W_j = W_j + \eta \Delta W_j$

Where η is the learning rate.
Note that the algorithm shown above is can be used for both Regression and Classification. However, for Classification only the two class problem is shown. This algorithm can be expanded to include a multiple class Classification problem.

Linear Neural Networks

With a basic understanding of the Linear and Logisitc regression algorithms, these processing techniques can be expanded to build a Neural Network. For the purposes of this paper the Linear Neural Network will be built with a Input Layer, Two Hidden Layers and the Final Output Layer. The structure of a Neural Network is something that can be tuned and tweaked to provide different results, that fell outside the scope of work for this paper, however provides an area of future work. The structure used for the Neural Network in this paper, has the number of nodes in each layer being equal to the number of features in the data set. In other words, the number of nodes in the hidden layers is equal to the domain of the data set. The neurons in each of the layers are connected by edges and those edges are tied to weight values. The algorithm explored in this paper is Back Propagation. The general idea behind Back Propagation is to model a pattern or model in the data set by modifying the weight of the internal network structure. This network is trained using supervised learning, where the error between the predicted value and the actual output value of an observation, is used to back propagate and update the weights of the edges inside the Network.

An observation is fed forward through the network, using an initial set of weights. As each feature in the input data set passes through a Neuron the Neuron is activated, as in the input feature times the weight connecting the feature to that neuron is calculated and then passed through the Sigmoid function. The Sigmoid function is used for the activation function in the neurons for both Classification and Regression problems.

The difference between the Classification and Regression problems comes about in the output layer of the network. For a two class Classification problem, there is only one output node and if the output node is between 0 and 0.5, then the data will be classified as class A. If the output node is above 0.5 to 1, then the data will be classified as class B. If it is a multiple class classification (greater than 2 classes), then there are as many output nodes as there are classes and the soft max function is used to chose the node with the largest value, the data will then be classified according to the class output node that contains the highest value. In regression, there is also only one output node, but the value stored in that output node is the continuous value predicted by the data.

In order to train the weights inside the network, a technique is used called back propagation. An observation is fed forward, and then the error between the predicted and actual output is propagated back through the network, so that each weight connecting each node is adjusted for the error it contributed to the final output. This is done by using Gradient Descent and feeding back the derivative of the Sigmoid function to each Neuron. The weights are trained using this process and presented in this paper as a number of iterations until the weights have converged.

Linear Neural Networks Algorithm

Build Network with the Desired Number of Hidden Layers and Nodes in Each Layer.
For each Edge connecting a node to another node in the network, set the Weight to a random value between (-0.01 and 0.01)
For Each Step in the Number of Iterations for Convergence:
    For each observation (t) in the Train Data Set:
        Feed Forward
            For Each Layer in the Network:
                If the input layer use the observation as the data into the Neuron.
                If any other layer use last layers output as the data the Neuron.
                For Neuron in the Layer:
                    Calculate the Activation At the Neuron..
                    Store that Neuron Output in the Neuron.
        Back Propagate
            Reverse the Order of the Network (so the the output layer is first).
            For Each Layer in the Reversed Network:
                If output layer:
                    Calculate the error between the neuron output and the actual data set predictor. $(r - y)$
                Calculate the derivative of the Sigmoid function, using the Neurons current output.
                Update the Neurons Updated Weight = $error - sigmoidderivative$
                If any other Layer:
                    For Node in Layer:
                        Update the Neurons Weight
                        Updated Weight = Updated Weight * Current Weight
        Re-Reverse the Network to return the original network with new updated weights.

Once Weights have Converged:
For each observation (t) in the Test Data Set:
Run the Steps listed under **Feed Forward**
Return a list of predictions for every observation in the Test Data Set.

Auto Encoded Linear Neural Networks

An Auto Encoded Linear NN is a variation of the Linear NN algorithm described previously. The intent of the Auto Encoded is to act as a de-nosing technique for the training of weights for the Linear Neural Network. The Auto Encoded Network is built with an input layer that has as many nodes as the input data set has features. There is one hidden layer and one output layer, both with the same number as nodes as the input layer. This Network has its weights trained using the same Back Propagation method discussed in the previously. Once the training has completed the Auto Encoded Hidden Layer weights are used to seed the first hidden layer weights of the Linear NN (instead of seeding them to random values between -0.1 and 0.1). In essence the Auto Encoded acts as a fine tuner of the weights using the data sets prior to running the full Linear NN.

AutoEncoded Linear Neural Networks Algorithm
Start Process for the Auto Encoded Neural Network
Build Network with the One Hidden Layer, with a number of nodes equal to the domain of the input.
For each Edge connecting a node to another node in the network, set the Weight to a random value between (-0.01 and 0.01)
For Each Step in the Number of Iterations for Convergence:
For each observation (t) in the Train Data Set:
Feed Forward
For Each Layer in the Network:
If the input layer use the observation as the data into the Neuron.
If any other layer use last layers output as the data the Neuron.
For Neuron in the Layer:
Calculate the Activation At the Neuron..
Store that Neuron Output in the Neuron.
Back Propagate
Reverse the Order of the Network (so the the output layer is first).
For Each Layer in the Reversed Network:
If output layer:
Calculate the error between the neuron output and the actual data set predictor. $(r - y)$
Calculate the derivative of the Sigmoid function, using the Neurons current output.
Update the Neurons Updated Weight = $error - sigmoidderivative$
If any other Layer:
For Node in Layer:
Updated the Neurons Weight
Updated Weight = Updated Weight * Current Weight

Re-Reverse the Network to return the original network with new updated weights.

Once Weights have Converged:

For each observation (t) in the Test Data Set:

Run the Steps listed under **Feed Forward**

Return a list of predictions for every observation in the Test Data Set.

Start Process for the Linear Neural Network

Build Network with the Desired Number of Hidden Layers and Nodes in Each Layer.

Seed the weights of the first Hidden Layer to be equal to the Weights obtained from the Auto Encoded Hidden Layer. For the Second Hidden Layer seed the Weight to a random value between (-0.01 and 0.01)

For Each Step in the Number of Iterations for Convergence:

For each observation (t) in the Train Data Set:

Feed Forward

For Each Layer in the Network:

If the input layer use the observation as the data into the Neuron.

If any other layer use last layers output as the data the Neuron.

For Neuron in the Layer:

Calculate the Activation At the Neuron..

Store that Neuron Output in the Neuron.

Back Propagate

Reverse the Order of the Network (so the the output layer is first).

For Each Layer in the Reversed Network:

If output layer:

Calculate the error between the neuron output and the actual data set predictor. $(r - y)$

Calculate the derivative of the Sigmoid function, using the Neurons current output.

Update the Neurons Updated Weight = $error - sigmoidderivative$

If any other Layer:

For Node in Layer:

Update the Neurons Weight

Updated Weight = Updated Weight * Current Weight

Re-Reverse the Network to return the original network with new updated weights.

Once Weights have Converged:

For each observation (t) in the Test Data Set:

Run the Steps listed under **Feed Forward**

Return a list of predictions for every observation in the Test Data Set.

Data Sets

The following data sets were used during the classification and regression tasks for this project.

Breast Cancer

Description:

Task: Classification
Predictor: Diagnosis (Malignant or Benign)
Link:
https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29

Car Evaluation
Description:
Task: Classification
Predictor: Car Evaluation (Unacceptable, Acceptable, Good, Very Good)
Link:
https://archive.ics.uci.edu/ml/datasets/Car+Evaluation

Congressional Vote
Description: 1984 United Stated Congressional Voting Records
Task: Classification
Predictor: Party (Republican / Democrat)
Link:
https://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records

Albalone
Description: Physical measurements of Albalone
Task: Regression
Predictor: Rings (int)
Link:
https://archive.ics.uci.edu/ml/datasets/Abalone

Computer Hardware
Description: Relative CPU performance data.
Task: Regression
Predictor: PRP
Link:
https://archive.ics.uci.edu/ml/datasets/Computer+Hardware

Forest Fires
Description: Forest Fire burn area data
Task: Regression
Predictor: Area (float)
Link:
https://archive.ics.uci.edu/ml/datasets/Forest+Fires

3. Results

Tables 1-3 display the results from Classification Tasks: the Breast Cancer, Car Evaluation and Congressional Vote data sets while running all three algorithms. Tables 4-6 display the results from the Regression Tasks: the Albalone, Computer Hardware and Forest Fire data sets while running all three algorithms. Each algorithm was tuned prior to being run on the full experiment set and optimal values were chosen out of that tuning process for the Learning Rate and the Convergence Factor. These tables show the results obtained during

each fold of the k-fold validation process. The results from each iteration of the tuning process was omitted from this report for brevity. The error used is Classification error and can be described as the Number of Times the Prediction was Wrong / Total Number of Comparisons. This error was averaged across the 5 folds to provide the Average Classification Error against each of the Classification data sets. The error used for Regression Tasks was the Mean Root Squared Error. This error was averaged across the 5 folds to provide the Average Mean Root Squared Error against each of the Regression data sets.

Table 1: Car Evaluation:Linear Networks - Experimental Results

Car Eval: Logistic Regression	
	Learning = 0.001 Convergance = 10
k-fold[1]	0.1336
k-fold[2]	0.2058
k-fold[3]	0.3695
k-fold[4]	0.2681
k-fold[5]	0.1993
Average Classification Error Across Folds	0.2352

Car Evaluation: Linear NN	
	Learning = 0.1 Convergance = 10
k-fold[1]	0.0497
k-fold[2]	0.2400
k-fold[3]	0.7016
k-fold[4]	0.0280
k-fold[5]	0.2441
Average Classification Error Across Folds	0.2532

Car Evaluation: Autoencoded NN	
	Learning = 0.01 Convergance = 10
k-fold[1]	0.2606
k-fold[2]	0.0498
k-fold[3]	0.0470
k-fold[4]	0.0361
k-fold[5]	0.7233
Average Classification Error Across Folds	0.2233

Table 2: Breast Cancer: Linear Networks - Experimental Results

Breast Cancer: Logistic Regression	
	Learning = 0.001 Convergance = 10
k-fold[1]	0.0440
k-fold[2]	0.0600
k-fold[3]	0.0500
k-fold[4]	0.2900
k-fold[5]	0.1300
Average Classification Error Across Folds	0.1100

Breast Cancer: Linear NN	
	Learning = 1 Convergance = 10
k-fold[1]	0.3240
k-fold[2]	0.6868
k-fold[3]	0.3400
k-fold[4]	0.6219
k-fold[5]	0.3700
Average Classification Error Across Folds	0.4687

Breast Cancer: Autoencoded NN	
	Learning = 0.1 Convergance = 100
k-fold[1]	0.6750
k-fold[2]	0.6860
k-fold[3]	0.3400
k-fold[4]	0.3780
k-fold[5]	0.6290
Average Classification Error Across Folds	0.5420

Table 3: Congressional Vote: Linear Networks - Experimental Results

Congressional Vote: Logistic Regression	
	Learning = 0.001 Convergance = 100
k-fold[1]	0.4000
k-fold[2]	0.4140
k-fold[3]	0.3285
k-fold[4]	0.3768
k-fold[5]	0.4492
Average Classification Error Across Folds	0.3930

Congressional Vote: Linear NN	
	Learning = 0.001 Convergance = 100
k-fold[1]	0.6079
k-fold[2]	0.3880
k-fold[3]	0.4100
k-fold[4]	0.6021
k-fold[5]	0.3799
Average Classification Error Across Folds	0.4777

Congressional Vote: Autoencoded NN	
	Learning = 1 Convergance = 10
k-fold[1]	0.6079
k-fold[2]	0.6115
k-fold[3]	0.4101
k-fold[4]	0.3978
k-fold[5]	0.3799
Average Classification Error Across Folds	0.4810

Table 4: Albalone: Linear Networks - Experimental Results

Alablone: Linear Regression	
	Learning = 0.001 Convergance = 10
k-fold[1]	773.0956
k-fold[2]	696.3123
k-fold[3]	641.3063
k-fold[4]	651.9946
k-fold[5]	692.11
Average RMSE Across Folds	690.96

Alablone: Linear NN	
	Learning = 0.001 Convergance = 100
k-fold[1]	9.44
k-fold[2]	10.35
k-fold[3]	9.80
k-fold[4]	10.06
k-fold[5]	10.14
Average RMSE Across Folds	9.97

Alablone: Autoencoded NN	
	Learning = 0.01 Convergance = 100
k-fold[1]	9.50
k-fold[2]	10.35
k-fold[3]	9.81
k-fold[4]	10.05
k-fold[5]	10.14
Average RMSE Across Folds	9.97

Table 5: Computer Hardware: Linear Networks - Experimental Results

Computer Hardware: Linear Regression	
	Learning = 0.001 Convergance = 10
k-fold[1]	320.27
k-fold[2]	100.04
k-fold[3]	162.87
k-fold[4]	201.24
k-fold[5]	331.59
Average RMSE Across Folds	223.20

Computer Hardware: Linear NN	
	Learning = 0.001 Convergance = 10
k-fold[1]	177.00
k-fold[2]	221.35
k-fold[3]	214.73
k-fold[4]	209.84
k-fold[5]	173.18
Average RMSE Across Folds	199.22

Computer Hardware: Autoencoded NN	
	Learning = 0.001 Convergance = 10
k-fold[1]	177.00
k-fold[2]	221.35
k-fold[3]	214.73
k-fold[4]	209.84
k-fold[5]	173.18
Average RMSE Across Folds	199.22

Table 6: Forest Fires: Linear Networks - Experimental Results

Forest Fires: Linear Regression	
	Learning = 0.001 Convergance = 10
k-fold[1]	10445.55
k-fold[2]	10070.43
k-fold[3]	3836.19
k-fold[4]	7845.00
k-fold[5]	8248.55
Average RMSE Across Folds	8089.15

Forest Fires: Linear NN	
	Learning = 1 Convergance = 100
k-fold[1]	68.12
k-fold[2]	68.09
k-fold[3]	20.97
k-fold[4]	67.01
k-fold[5]	65.89
Average RMSE Across Folds	58.02

Forest Fires: Autoencoded NN	
	Learning = 1 Convergance = 100
k-fold[1]	68.12
k-fold[2]	68.09
k-fold[3]	20.97
k-fold[4]	67.00
k-fold[5]	65.89
Average RMSE Across Folds	58.02

4. Discussion

The hypothesis presented in this report was that the Feed-Forward Network with an Auto Encoder will perform better than the Feed-Forward Back Propagation and Linear regression approaches on both the Classification and Regression problems.

Looking at the Classification data sets, it seemed that the Logisitc Regression performed best on average when compared with the Linear and Auto Encoded NN. Looking at the Regression data sets for the Linear NN and Auto Encoded NN, it seems that the

Alabolne models were able to obtain a line of best fit as it had the smallest average RMSE across the folds.While the Computer Hardware had the largest average RMSE across the folds. A factor in this trend might be how the data has handled prior to being run through the algorithm. Some of the data in these data sets was categorical, so a One Hot encoding was applied. The Computer Hardware data set had the highest number of features that needed to be one hot encoded, when compared to Forest Fires (middle amount of one hot encoding) and Albalone (smallest amount of one hot encoding). This one hot encoding may have skewed the data in some way causing the RMSE to be highest for the Computer Hardware data set. For Regression data sets it was also noted that Auto Encoded NNs versus a regular Linear NN, seemed to have no impact on the RMSE that was obtained fold to fold.

One possible explanation for these results is that it was observed during the tuning process that very frequently the same fold error would be obtained fold to fold, this may indicate that in many cases the predictions being made by the algorithms were often tending to choose only one class label. This would mean that instead of learning a pattern the algorithm is just calculating the performance of the fold observations that were of that one class that it was predicting. This was noticed on the most on the computer hardware data set and as an area of future work should be examined to see if that truly is the root cause of the error. If that is the case, then it may be hard to make concise conclusions about the results of these modes.

5. Conclusion

Overall there seemed to be some trends that supported the general hypothesis. For regression data sets it seemed that the Linear NN would perform better than the Linear Regression. For classification the logistic regression performed best.

A suggested area of future work, after fixing basic implementation details, would be to explore as part of the tuning process is the shape of the network. In the experiment the Neural Network that was built for the algorithm that included and input layer, 2 hidden layers and an output layer. This network could have been built with a different number of hidden layers and a variable number of nodes with in those hidden layers, that could have lead to different results. That structure of the network is an area that could be looked into for future work.

6. References

1. Alpaydin, E. (2004). Introduction to machine learning (Oip). Mit Press.