JHU AAP EN.605.649.83.FA21 Project 4 Report:

Decision Trees – Feedforward Neural Networks

Mauro Chavez Mauro.antoine.chavez@gmail.com | MCHAVEZ8@JH.EDU
AAP M.S. Bioinformatics Student *(858)354-7465*
San Diego, CA

Abstract

This document summarizes the fourth project deliverable for Introduction to Machine Learning taken through Johns Hopkins AAP in Fall of 2021. This project involved implementing simple linear and logistic regression models before moving on to neural networks with two hidden layers. Parameters like the number of neurons in each hidden layer, the learning rate, and the maximum allowed iterations in training were set based of testing a small collection of potential options. Once these parameters were set, the predictive power of the simple models, the neural network, and the neural network assisted by an autoencoding layer were quantified to study how well each model improved predictive power. The UCI Machine Learning Repository data sets for Breast Cancer, Car Evaluation, and Congressional Vote, were used for classification and the Abalone, Computer Hardware, and Forest fires data sets were used for regression. Classification tasks utilized SoftMax while regression tasks performed a linear combination of inputs. Neural networks outperformed the simple models by significant margins. The addition of the autoencoder only increased performance in a few key data sets. This is likely the result of a poorly optimized autoencoder training protocol.

Video demonstration of code can be found here: https://youtu.be/fJfEssXI47E

1 Introduction

This assignment involved implementing feedforward neural networks. This included basic linear and logistic regression algorithms as baseline models in addition to larger networks with hidden layers trained via back propagation. Autoencoders were used to pretrain input layers to the neural network in an effort to improve predictive power. Previous UCI Machine Learning Repository data sets were studied using a linear network, a feed forward network with two hidden layers, and a feed forward network where the first layer had been trained from an autoencoder. The classification and regression tasks differed in both output layer and loss functions. Unlike previous labs, data normalization required a different approach as attributes had to be tweaked to fit a 0-1 range to support passage through neural networks.

1.1 Problem Statement

The high-level goals of this assignment were to implement networks of increasing complexity on the UCI Machine Learning Repository data sets. Classification tasks utilized the Breast Cancer, Car Evaluation, and Congressional Vote data sets while regression tasks required the Abalone, Computer Hardware, and Forest Fires data sets. Each data set was adjusted such that discrete/categorical features were one-hot encoded and continuous attributes were min/max normalized. Logistic regression for classification and linear regression for regression were first

©2021 Mauro Chavez

considered as fundamental algorithms. Feedforward neural networks with two hidden layers were then implemented and deployed to the same learning problems.

All three models were applied to each data set; linear/logistic regression, neural network, and neural network + autoencoder. Testing involved 5-fold cross validation in profiling predictive capabilities to compare results statistically. Regression tasks quantified accuracy via mean squared error while classification tasks tracked proportion of the data where the predicted label matched the actual label.

1.2 Hypothesis

I anticipate the linear and logistic regression models to perform modestly well with scores comparable to ones seen from past assignments. Neural networks should be a 20-30% improvement from the modest scores of the simpler models as they are much more complex and have the ability to better decern more complicated decision boundaries. Auto encoders will likely provide an improvement inversely proportional to the gain of predictive power between the simple and neural network models. For example, if jumping to the neural network provided a large increase in predictive power, then adding an autoencoder will provide a smaller increase than if jumping to the neural network only provided a modest increase. In short, if a neural network was very successful in classifying a data set, adding an autoencoder won't change too much. That being said, if a neural network is still missing 10-20% of the data, an autoencoder might help bridge that gap.

2 Data Pre-Processing

Similar to past assignments, attributes that functioned as data point identifiers were dropped prior to learning. Missing feature values were imputed using the average feature value. As noted earlier, it was important to normalize feature values to be in the range 0-1 prior to learning. In general, discrete/categorial features were broken out using one-hot encoding into sets of features with Boolean values for each category. Continuous attributes were adjusted by min max normalization to put them on the same 0-1 scale as the Boolean one-hot encoded features.

To help partitioning of the data for 5-fold cross validation, continuous labels were discretized into 100 equal width bins to help define partitions with comparable label frequency to the complete data set.

2.1 Abalone Data (Regression)

Abalone data was used for regression where attributes were used to predict age which was quantified by the number of rings recorded on the sample shell. The "sex" attribute of this data set was broken out as nominal data resulting in sex_m and a sex_f columns. All other columns were min max normalized which included; length, diameter, whole_weight, shucked_weight, viscera_weight, and shell_weight.

2.2 Breast Cancer Data (Classification)

Breast cancer data was used for classification where attributes were used to predict cancer class. Cancer class for this data set was either 2 for benign or 4 for malignant which made this a binary classification task. There existed some "?" values in this data set. These were replaced with null values and imputed using the column mean. The sample column was dropped as data points were instead identified by indices in the data frame. All other columns were min max normalized which included: clump_thickness, uniformity_of_cell_size, uniformity_of_shell_shape,

maginal_adhesion, single_epithelial_cell_size, bare_nuclei, bland_chromatin, normal_nucleoli, mitosis

2.3 Car Data (Classification)

Car data was used for classification in order to predict acceptability. Similar to previous assignments, the buying, maint, safety, and acceptability attributes were normalized into ranks where 0 was the best/most desirable. Once normalized into ranked values, the buying and main features were min max normalized to squash the rating scale between 0-1. As these values were initially rank ordered, min-max normalization but equal distance between each value in the ranking. Acceptability did not need to be normalized as it functioned as the class label. As there were multiple acceptability classes, this classification task was not binary. Nominalization of ordinal columns without clear value rank was performed on features: doors, persons, and lug_boot attributes, as these were broken out into nominal data columns.

2.4 Forest Fire Data (Regression)

Forest fire data used for regression in order to predict burnt area. To start, the month and day attributes were broken down into nominal data columns. It was recommended in the data description file for this set that the area be log transformed as values are heavily skewed towards 0.00. However, when $\log_{10}$ transforming this column, all 0.00 values would be replaced with -inf in the data frame which was not useful for regression. To correct for this, a noise factor of 0.0000001 was added to each data point's area attribute. After this adjustment, log transformation gave non infinite values to each data point. Furthermore, the X and Y features were dropped as they functioned more as data point identifiers. All other columns were min max normalized, this included; FFMC, DMC< DC, ISI, temp, RH, wind, and rain.

2.5 House Votes Data (Classification)

The House votes data set was used for classification where voting history was used to predict party affiliation. As only democrats and republicans were represented in this data set, this data set was used for binary classification. Every attribute was broken down into nominal data columns such that for each vote, there was a "yes", "no", "?" feature. Normalization for this data set was different from the other two classification data sets as this was the only one not to have min max normalized features.

2.6 Computer Hardware (aka Machine) Data (Regression)

The computer hardware data set was used for regression where the estimated relative performance was used as the target of prediction. The vendor and model_name attributes were dropped as they did more to serve as data point naming than actual device performance quantification. All other features were min max normalized which included; MYCT, MMIN, MMAX, CACH, CHMIN, CHMAX, PRP.

3 Experimental Approach

3.1 Linear Networks for Classification and Regression

Two simple models were initially used for predictions to set a base line to compare future models too. The first was a linear regression model that was applied to regression tasks. Here a single linear separator was applied to the data in order to learn a linear combination of feature values that closely predicted an examples class value. The weight vector was initialized using a uniformly distributed

set of values between -0.01 and 0.01. A batch updating approach was taken where each example in the training set was iterated in random order. For each example, the input was scaled by the error between the predicted value and the actual value and added to an adjustment vector. Once all examples where checked, the adjustment factor and bias were normalized by the amount examples in the training and scaled by a learning rate (.001). This normalized adjustment was then added to the weight vector and bias used in the linear model. Each regression data set was tested using 5-fold cross validation and the mean squared error on the training set was averaged over each fold. Learning on each fold continued for a predefined minimum value for iterations through the training set or until the mean squared error increased by 1% from one iteration to the next. Logistic classification was similar only that each class had its own weight vector. For each example in the training set, the weight vector of each class would be applied to the example and then passed through soft max normalization. For each example, classes that did not match the actual label were adjusted by (0 – class_score)*example_vector and the class that did match the actual label were adjusted by (1 – class_score)*example_vector. Both these adjustments were scaled by the learning rate (0.10) prior to combination with a class's weight vector. During an entire pass though the training data, all these adjustments were summed and divided by the number of examples in the training set for a batch update approach. Learning involved iterating through the training set in random order, updating the weight vectors for each class after each example. Once an iteration through the training set increased the count of misclassified examples, learning stopped.

3.2 Application of Simple Feedforward Neural Network

For all examples, each neural network only utilized two hidden layers. The structure of the output layer changed when performing classification vs regression. Neural networks completing regression tasks used a mean squared error loss function and linear combination for predictions. Here the output layer was a single neuron combining the values from the last hidden layer. For classification, cross entropy loss was used as a loss function and SoftMax was used for prediction. In this case the output layer had N neurons where N was the number of classes in the data set. The expected value for the classification case was a vector where the dimension matching the actual class was 1 and all other values were 0. Error for the last hidden layer was calculated using the derivative of cross entropy multiplied by the derivative of the SoftMax function for backpropagation. In both classification and regression, the sigmoid activation function was utilized in hidden layer neurons. In applying the neural network to each problem set, different values of hidden layer neuron count were tested along with various learning rates and maximum allowed iterations. In learning training sets, input data was shuffled and iterated through applying stochastic gradient decent to alter the weights of the network after each example. Training was constrained by a maximum number of allowed passes through the training set. However, if an iteration through the training set resulted in performance worse than the average performance of the past 20 iterations, learning stopped. Once layer neuron count, learning rate, and maximum allowed iterations were determined, performance on each data set was recorded.

3.3 Addition of Autoencoder layer to Neural Network

Autoencoders were pretrained and applied as the first hidden layer of the neural network. The autoencoders themselves were designed with one encoding/hidden layer and one output layer. The encoding layer had less neurons than there were features in the input examples. For each data set, the encoding layer had 60% as man neurons as there were features rounded down. The decoding layer had as many neurons as there were features in the input examples. Mean squared error loss was used to profile the difference between the input and output. 30 passes through the training set were allowed to train the autoencoder. Once trained, the hidden layer was extracted and used as the first hidden layer of a neural network that was then trained as usual on the same data. This had the

effect of giving the network better initial predictive power and to fine tune the weights of the autoencoder layer.

3.4 Model comparison

Models were compared based on their performance on test sets averaged across 5-fold cross validation. For all the regression tasks, each model would report the average mean squared error. Similarly, each classification tasks would report the average number of examples classified correctly. These values for each model would be compared to quantify performance.

4 Experimental Results

4.1 Hidden Layer Node Count Testing

The Car Evaluation data set was randomly chosen to be used as the test subject for the application of different hidden layer node counts. Due to time constraints, the results of testing this data set were generalized to all other data sets. As 7 neurons provided the highest classification score (proportion of text examples correctly classified during 5-fold cross validation), all other experiments used two hidden layers each with 7 neurons.

	5 Neurons	7 Neurons	15 Neurons	25 Neurons
Classification Score	0.8401	0.8772	0.83444	0.60484
Average Training Iterations	150	150	150	150

Table 1: Classification score on the Car Evaluation data set using a learning rate of 0.80 and 150 max iterations with various hidden layer neuron count.

4.2 Neural Network Learning Rate and Max Iteration Testing

The following describes tests ran for each data set comparing different learning rate and minimum iteration parameters. Each data set was tested with three sets of parameters where the bold set in each row shows those selected for future analysis.

	Test 1		**Test 2**		**Test 3**	
	Params	Results	Pararms	Results	Params	Results
Abalone (~4180 dp)	**lr: 0.1 mi: 200**	**mse: 5.6816 ai : 200.0**	lr: 0.20 mi: 100	mse: 5.6009 ai : 100	lr: 0.50 mi: 150	mse: 71.6840 ai : 98.8
Computer Hardware (~210 dp)	**lr: 0.0001 mi: 1200**	**mse: 8173.05 ai : 483.6**	lr: 0.005 mi: 1200	mse: 14328.37 ai : 36.2	lr: 0.01 mi: 350	mse: 20238.98 ai : 30.6
Forest Fires (~520)	lr: 0.00001 mi: 200	mse: 24.2728 ai : 200.0	**lr: 0.001 mi: 200**	**mse: 24.2396 ai : 138**	lr: 0.1 mi: 200	mse: 29.5709 ai : 200.0
Breast Cancer (~700 dp)	lr: 0.25 mi: 200	cs: 0.96148 ai: 200	lr: 0.5 mi: 300	cs: 0.964 ai: 300	**lr: 0.8 mi: 100**	**cs: 0.96146 ai: 100**
Car Evaluation (~1700 dp)	lr: 0.1 mi: 200	cs: 0.865 ai: 200	lr: 0.5 mi: 450	cs: 0.842 ai: 450.0	**lr: 0.8 mi: 150**	**cs: 0.877 ai: 150.0**

Congressional (~450 dp)	**lr: 0.25 mi: 800**	**cs: 0.822 ai: 800**	lr: 0.5 mi: 500	cs: 0.684 ai: 500	lr: 0.8 mi: 1200	cs: 0.808 ai: 1200

Table 2: Testing of various learning rate (lr) and maximum allowed iterations through the training data (mi) and the resulting mean squared error (mse) or classification score (cs) on the test set. Classification score quantifies the proportion of example classified correctly. The average number of iterations before learning was exited is also reported as a result.

4.3 Model Performance Comparison

The next set of figures show the actual vs predicted labels for each regression data set. The points are colored based off which fold they came from in k-fold cross validation where fold contents remained consistent across runs.

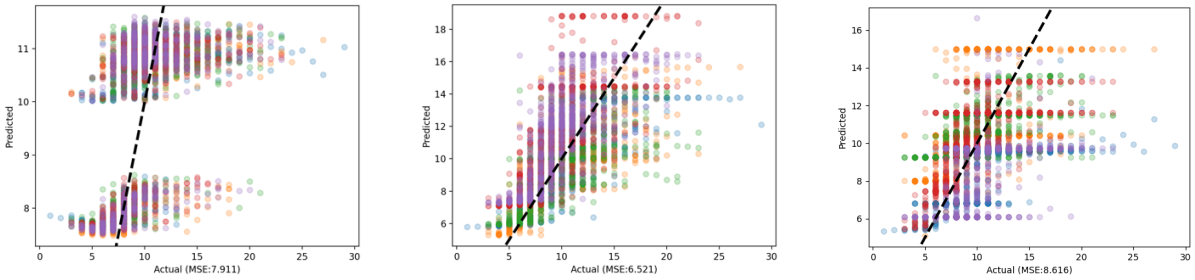

Figure 1: Predicted vs actual class value while learning of Abalone data set using linear regression (left), neural network (center), and autoencoder assisted neural network (right). Dashed lines on each graph signify x = y.

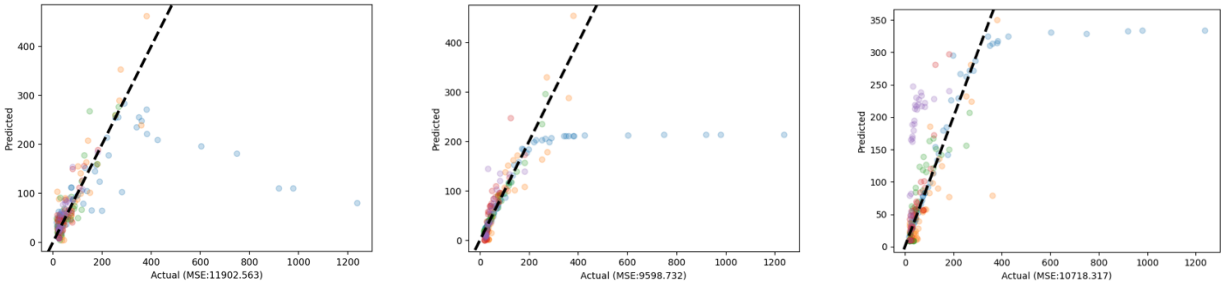

Figure 2: Predicted vs actual class value while learning of Computer Hardware data set using linear regression (left), neural network (center), and autoencoder assisted neural network (right). Dashed lines on each graph signify x = y.

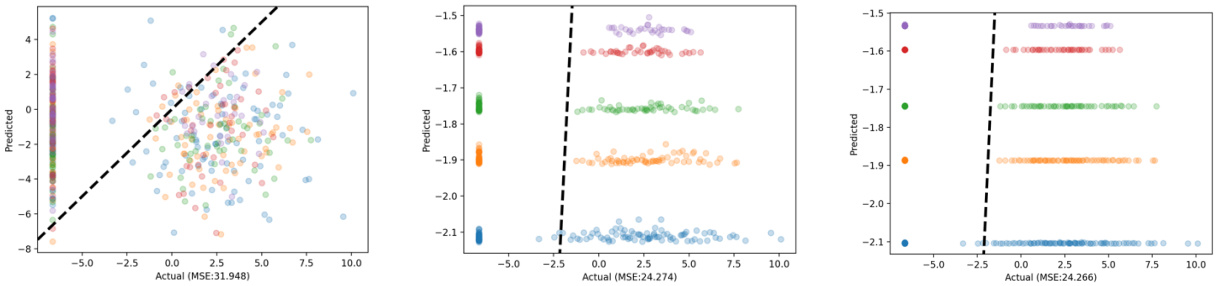

Figure 3: Predicted vs actual class value while learning of Forest Fire data set using linear regression (left), neural network (center), and autoencoder assisted neural network (right). Dashed lines on each graph signify x = y

	Logistic Regression	Neural Network	Autoencoder Assisted Neural Network
Breast Cancer (~700 dp)	0.3447	0.9629	0.96712
Car Evaluation (~1700 dp)	0.70022	0.877	0.8576
Congressional (~450 dp)	0.91482	0.822	0.963059

Table 3: Classification scores (proportion of data classified correctly) for each classification data set under the application of linear regression, neural network, and auto encoder assisted network models.

5 Discussion

Before getting into the comparison of models, there are a few important things to note. The first being that the min max normalization approach taken throughout this analysis may have hurt the predictive power of each model. Min max normalization is sensitive to outliers which means that extremely small or extremely large data points that may stray far from the norm will push more regular samples closer together artificially. This ultimately removes data from the analysis as distances between examples are minimized on account of being normalized linearly against outliers in the data.

Another interesting thing to note is that regression related data sets often required a much lower learning rates than classification data sets. Generally anything above 0.10 would likely result in iterations of the training data that never converged towards an optimal set of weights. This might be on account of differences in the derivatives used in backpropagating error between regression and classification tasks. Regression calculates the difference between the predicted and actual value as the derivative of MSE which initially will be very large. This means adjustments will be more sensitive to larger learning rates. However, classification derivatives go through first the derivative of cross entropy which scales exponentially based off how bad a prediction is and, and then the derivative of SoftMax which serves as a normalizing factor. I think these two steps make it so that the learning process is less sensitive to the learning rate in the classification case in general.

In general, neural networks outperformed the simpler models. In the case of the abalone and computer hardware data sets, there was clear decrease in mean squared error. This can be visually appreciated with the first two plots in figures 1 and 2 where the actual vs predicted label spread gets tighter. Interestingly, the linear model missed an entire range of predictions near the center of the label spectrum when working with the abalone data set. The forest fire data set continues to be a menace. The linear model was all over the place while the neural net seemed to predict the weighted average for each fold for the entire training set. For classification data sets, the neural network was a clear improvement in the breast cancer and car evaluation cases. However, the logistic model had success with the Congressional data set that was not recovered with the neural net. That being said, the application of the autoencoder to the neural net for this data set saw the biggest gains in resulting classification score.

The autoencoder learning protocol may have required more rounds of testing and optimizing as its application did result in lower initial mean squared error, but training on the larger network frequently violated the criteria that mean squared error did not exceed the average of past 20

iterations and terminated learning much earlier. On account of this much earlier termination, weights were never tuned as precisely as they were without the autoencoder. I think that the learning process of a network without an autoencoder should be parameterized very different from the learning process of a network with an auto encoder. Given more time I would have investigated that further.

Learning in general may have suffered from overlearning the training data. This is most clearly seen in the case of the Computer Hardware data set where the first fold was able to get mean squared error on the training set to ~541 which ended up giving 45826 mean squared error on the test set. It's important to note that this analysis did not have a bias constraint. If a check were added to stop learning once minimum competency was reached on the training set, this type of behavior might have been mitigated.

Conclusion

Neural networks in general have more predictive power than simpler models. However, they require much more overhead in terms of implementation and run time. This means that it is critical to at least consider simpler models to solve learning problems before reaching for a neural network. This is best exemplified with the congressional data set where the logistic regression model out preformed the neural net while requiring significantly less runtime. The forest fire data set continued to be a pain, but the abalone and computer hardware data sets saw noticeable improvements with the application of the neural net.

Performance of the autoencoder across the board with the exception of the congressional data set was disappointing. This may be a consequence of the one size fits all approach to training the auto encoder. Given more time, I would have liked to further investigate how to best use the autoencoder. This investigation could involve tweaking how much time an autoencoder gets for learning and to what degree it should cut down the size of the input. As shown with the congressional data set, there are significant improvements to be made from effectively adding an autoencoder to a model, so I do believe in their utility.

This assignment reinforced a trend that has popped up in past assignments where there are so many dimensions for optimizing these models that it's incredibly difficult to get everything dialed in. I discussed the number of hidden nodes per layer, learning rate, and maximum iteration threshold, but I didn't even touch optimizing the exit conditions to learning the test data. This would have provided an entirely new dimension to model tweaking that could easily have added another few days of work. This aspect of machine learning is incredibly interesting so long as I don't let myself get anxious about finding the perfect blend of runtime parameters to optimize model performance.