<u>CS461 Program 3 Report:</u>

Fall 2020

Prof. Brian Hare

Lauren Magee

Preprocessing

The goal of this program was to take a dataset, with varying information about consumers that currently had health insurance with a company, and create a neural network to predict whether or not they'd also be interested in purchasing car insurance. One of the most important parts of developing a reliable and accurate neural network is ensuring your recoded weights from the initial data support the correct learning outcomes from your program. For my input layer, I preprocessed the data in the manner designated by the assignment's instructions. For example, Gender was changed to 0/1, Region_Code to 1-hot, etc. While not needed in the final turn in, I provided my data recoding scripts along with the resulting 70/15/15 .csv files in the zipped submission. In evaluating the final results, I find it useful to see what was used to produce them.

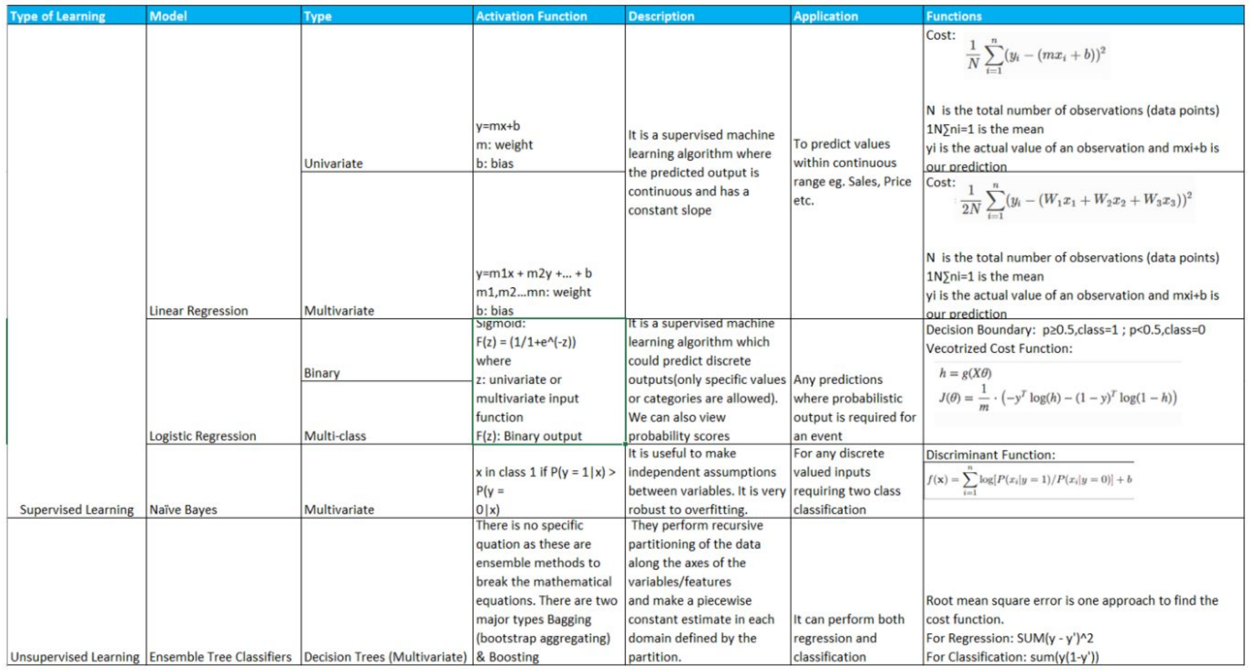
Network Configuration: Learning and Testing

For personal curiosity, I decided to run different independent variables of the training data in separate executions with the learning model. For example, in addition to running the entire 70% .csv file at once, I ran the Gender column against the Response Column, the Vehicle Age column against the Response Column, and so on for all the independent variables. The goal was to find the variables that had the highest correlation with the target response value of "1" so that I had another angle to look at for their codependent prediction later in the final results. It was interesting to identify which column synapses were weighted heavier within the hidden layer(s).

Furthermore, I utilized many different methods of machine learning techniques from a multitude of sources to finalize my network configuration. In the beginning, I combined basic Tensorflow templates found from our class resources and other references just to test I could execute a successfully run program. After this was proven, I started making slight alterations to various parameters and adding different open sourced Tensorflow functions to see if any learning improvements could be made.

One of the first comparisons I made was the accuracy difference between one or two layers. I wanted to see if the addition of another activation function would make a dramatic difference in the resulting output layer. I didn't find this to be the case, with the average accuracy for one layer being .8759 and the average accuracy for two layers being .8768. However, it is possible this similar accuracy is due from overtraining.

Both of my hidden layers use relu as their activation functions. After reading several different open source point of views, this seemed to be the best choice because of its simple linear computation methods. My response possible output is binary, this fits with relu because the rectified linear unit will return zero for all negative values, and linearly shape out for the positive ones. Hence if my machine resolves to zero it'll still be zero, as one will remain one. My output layer uses the sigmoid activation function because it mirrors the same logic as the relu function, by limiting the output values between 0 and 1 (or -1 and 1 but this doesn't apply with the relu hidden layers) it also supports the concept of an "either or" binary result. I found great guidance making these network configuration decisions from the graphic below.



Type of Learning	Model	Type	Activation Function	Description	Application	Functions
		Univariate	y=mx+b m: weight b: bias	It is a supervised machine learning algorithm where the predicted output is continuous and has a constant slope	To predict values within continuous range eg. Sales, Price etc.	Cost: $\frac{1}{N}\sum_{i=1}^{n}(y_i-(mx_i+b))^2$ N is the total number of observations (data points) 1N∑ni=1 is the mean yi is the actual value of an observation and mxi+b is our prediction
	Linear Regression	Multivariate	y=m1x + m2y +... + b m1,m2...mn: weight b: bias			Cost: $\frac{1}{2N}\sum_{i=1}^{n}(y_i-(W_1x_1+W_2x_2+W_3x_3))^2$ N is the total number of observations (data points) 1N∑ni=1 is the mean yi is the actual value of an observation and mxi+b is our prediction
		Binary	Sigmoid: F(z) = (1/1+e^(-z)) where z: univariate or multivariate input function F(z): Binary output	It is a supervised machine learning algorithm which could predict discrete outputs(only specific values or categories are allowed). We can also view probability scores	Any predictions where probabilistic output is required for an event	Decision Boundary: p≥0.5,class=1 ; p<0.5,class=0 Vecotrized Cost Function: $h=g(X\theta)$ $J(\theta)=\frac{1}{m}\cdot\left(-y^T\log(h)-(1-y)^T\log(1-h)\right)$
	Logistic Regression	Multi-class				
Supervised Learning	Naïve Bayes	Multivariate	x in class 1 if P(y = 1|x) > P(y = 0|x)	It is useful to make independent assumptions between variables. It is very robust to overfitting.	For any discrete valued inputs requiring two class classification	Discriminant Function: $f(\mathbf{x})=\sum_{i=1}^{n}\log[P(x_i\|y=1)/P(x_i\|y=0)]+b$
Unsupervised Learning	Ensemble Tree Classifiers	Decision Trees (Multivariate)	There is no specific quation as these are ensemble methods to break the mathematical equations. There are two major types Bagging (bootstrap aggregating) & Boosting	They perform recursive partitioning of the data along the axes of the variables/features and make a piecewise constant estimate in each domain defined by the partition.	It can perform both regression and classification	Root mean square error is one approach to find the cost function. For Regression: SUM(y - y')^2 For Classification: sum(y(1-y'))

Source: How to perform Quality Assurance for Machine Learning models? \| by Dhaval M \| Data Driven Investor \| Medium

Continuing to follow the binary logic, I chose to compile the model using the "adam" optimizer, which executes stochastic gradient descent, and assigning my loss function to "binary_crossentropy" because, again, the machine to trying to predict a result from two potential outcomes. I also added a repeating shuffler function in an effort to reduce machine memorization.

In terms of sequencing parameters, because the dataset was so large initially and stayed fairly large in size after the 70/15/15 split, I assigned my batch size to be 250. The other batch sizes I tested were 50, 100, 200, 300, 500, and 1000. I found there were no dramatic changes in the results, only a decrease in the time the program took to execute when the value was higher. Therefore, I went with the number that provided the highest accurate return.

To determine the optimal amount of epochs, I applied the 'Elbow Method' to my machine training. The goal was to avoid underfitting and overfitting the network because the more times the dataset is run through, the more times the weights are changed and slightly biased to that particular set. Using the loss return as my indicator, the Elbow Method attempts doing this process by test running different epoch values and looking for a value that 'dips' or causes a drastic change that makes the loss return closer to zero. From my various tests, I found the value of 5 seemed to be the "sweet spot" for this parameter. The values I tested and their loss return results are shown in the table below for reference.

Epochs	**Loss Percentage**
1	.5201
2	2.908
3	2.828
5 (value was the median of the 2-3 range)	2.739
10	.2724
15	.2725

The other sequencing parameter adjusted was the "steps" value within the train function. Following the open source references I found, the refined value was the product of the traditional equation for steps for epoch.

Steps = len(training_data) / batch_size
Steps = 266,000 (# of lines in 70% training) / 250
Steps = 1064

My supervised learning machine then uses Tensorflow's estimator function to achieve "deep learning classification". To set it up, I uploaded two .csv files, the training (70%) and the testing (15%), and transformed them into dataframes utilizing the Panda library. Once they were formatted properly I created four variables, two of them were assigned the independent variables' data from training and testing, and the other two were assigned the dependent data (Response Column) from training and testing. I first ran Tensorflow's train function against the training data frame and its expected results. After this was completed, the program then executed Tensorflow's evaluate function on the testing data frame and its expected results to gauge the accuracy from its prior learning experience. For the final test, I exchanged the testing .csv file for the validation .csv file, followed the same mapped out process, and obtained the outputted results.

Most all other functionality of the Tensorflow templates remained the same, the other minor differences can be viewed with my source code provided in the FinalTensor.py file.

Validation and Results

I chose to use the k-fold cross validation strategy to validate my final results. I selected this method because the split of 70/15/15 was already preassigned and I thought it would be interesting to train and test the datasets in different combinations to see which one produced the most accurate return. In this instance of k-fold, k equated to 3 which resulted in 6 tests that needed to be run and finalized. The different combinations of datasets, their accuracy, and loss values are listed in the table below for evaluation.

Trained Set	Tested Set	Accuracy	Loss
Training70	Testing15	.8762	.2746
Training70	Validate15	.8757	.28
Testing15	Validate15	.8774	.2941
Testing15	Training70	.8774	.3144
Validate15	Training70	.8791	.2836
Validate15	Testing15	.8786	.2792

Unfortunately with the results I received, not many trends were identifiable. I was hoping to see that with a smaller training dataset, a larger tested dataset would predict with lower accuracy and higher loss. This can partially be demonstrated at a low level with "Testing15" as the training set and "Training70" as the tested set. Another trend I was hoping to find is the opposite of the first case, that a larger training dataset paired with a smaller tested dataset, would result in higher accuracy and lower loss. Mostly because the machine would've had more cases to shift through meaning more time to adjust the weights of signals within its hidden layers. This was not a loud result either. It's possible the likeness of these results are due to overfitting. The calculated average of accuray my machine had correctly predicted whether a health insurance customer would also be interested in car insurance was .8774 with an average loss of .2877.

In conclusion, I learned an abundance about Tensorflow, the implementation of machine learning, and how to preprocess data from categorical to quantitative values. Utilizing the Tensorflow library made me feel like I was on the "frontend" of machine learning and in the future I would like to know more about the math and theory that goes into the backend. I find it hard to fully understand its functionality and capabilities without diving directly into the source. For this project, a lot of the code used was from templates and tutorials, with some of it then customized later on by me. Given this foundational experience, the next time I have a machine learning project I would like to explore more of the advanced features listed on Tensorflow's open source page. The numerous methods I researched through various channels revealed this is subject matter with great depth, a lot of which has not been fully explored yet. Overall, this was a very beneficial and interesting program to develop and I valued the experience.

I know references weren't required in the turn in but they're still listed below, the Quality Assurance article from Medium was a great read!

References

A Gentle Introduction to k-fold Cross-Validation (machinelearningmastery.com)
A Practical Guide to ReLU. Start using and understanding ReLU… | by Danqing Liu | Medium
Elbow Method in Supervised Machine Learning(Optimal K Value) | by Moussa Doumbia | Medium
How to perform Quality Assurance for Machine Learning models? | by Dhaval M | Data Driven Investor | Medium
Implementation of Artificial Neural Network in Python- Step by Step Guide (mltut.com)
neural network - Why should the data be shuffled for machine learning tasks - Data Science Stack Exchange
Sigmoid Function Definition | DeepAI