COMS 3007: Machine Learning Assignment 2020

Group: Boolean Autocrats Group

members:

1. **Donald Mbara: 1839569**
2. **Tebogo Mojela: 1857333**
3. **Mongezi Dlamini: 1984652**
4. **Kaone Senoelo: 1819369**

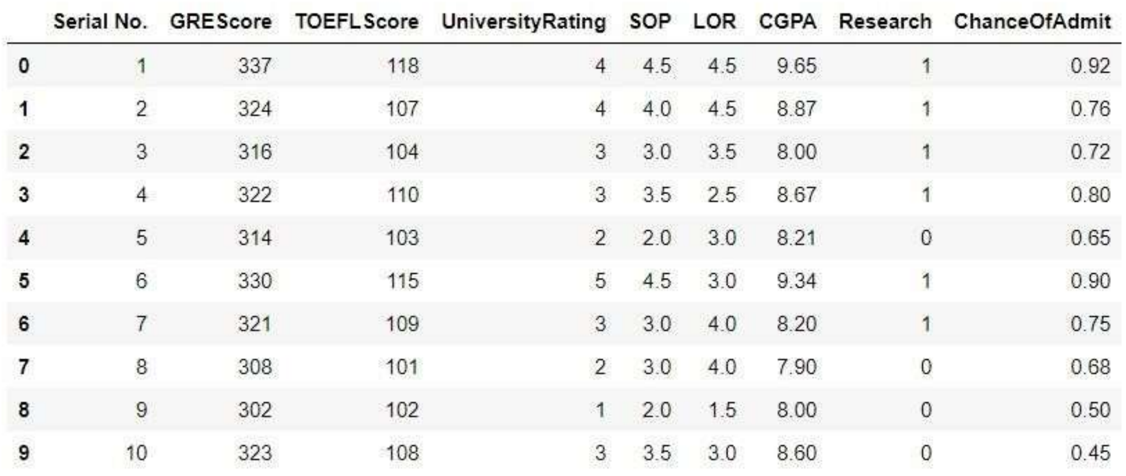
The dataset contains 400 data points and contains features which are considered important to look at during the application for Master's Program in India. The parameters (Independent variables)

included are:

- Graduate Record Examination (GRE) Scores (out of 340)
- Test of English as a Foreign Language (TOEFL) Scores (out of 120)
- University Rating (out of 5)
- Statement of Purpose (SOP) (out of 5)
- Letter of Recommendation Strength (LOR) (out of 5)
- Undergraduate GPA (CGPA) (out of 10)
- Research Experience (either 0 or 1)

The target values are in a class called **Chance of Admit** (dependent variable) which range from 0 to 1. The targets are normalized to either 0 – which represents not admitted/accepted or 1 – which represents admitted/accepted.

Here is the list of the first 10 rows of our dataset with dependent variable (**Chance of Admit**) not yet normalized



	Serial No.	GREScore	TOEFLScore	UniversityRating	SOP	LOR	CGPA	Research	ChanceOfAdmit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65
5	6	330	115	5	4.5	3.0	9.34	1	0.90
6	7	321	109	3	3.0	4.0	8.20	1	0.75
7	8	308	101	2	3.0	4.0	7.90	0	0.68
8	9	302	102	1	2.0	1.5	8.00	0	0.50
9	10	323	108	3	3.5	3.0	8.60	0	0.45

We are predicting whether a candidate get into the Master's Program or not.

The Algorithms that we are going to apply to this data to predict the chance of admission are Decision Trees, Linear Regression and Neural Networks.

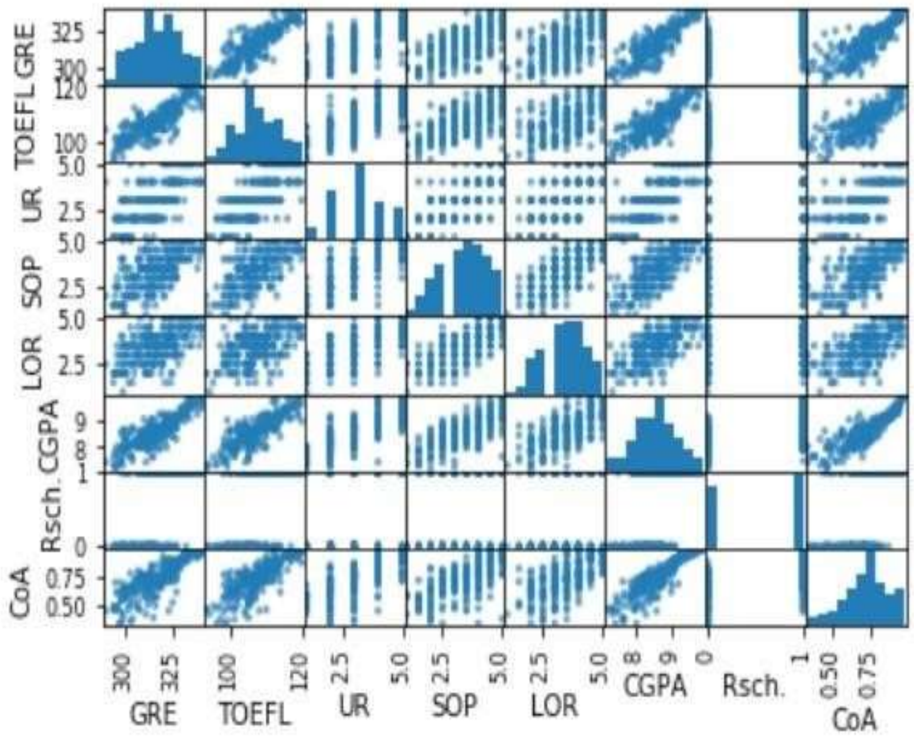
Linear Regression

Data processing

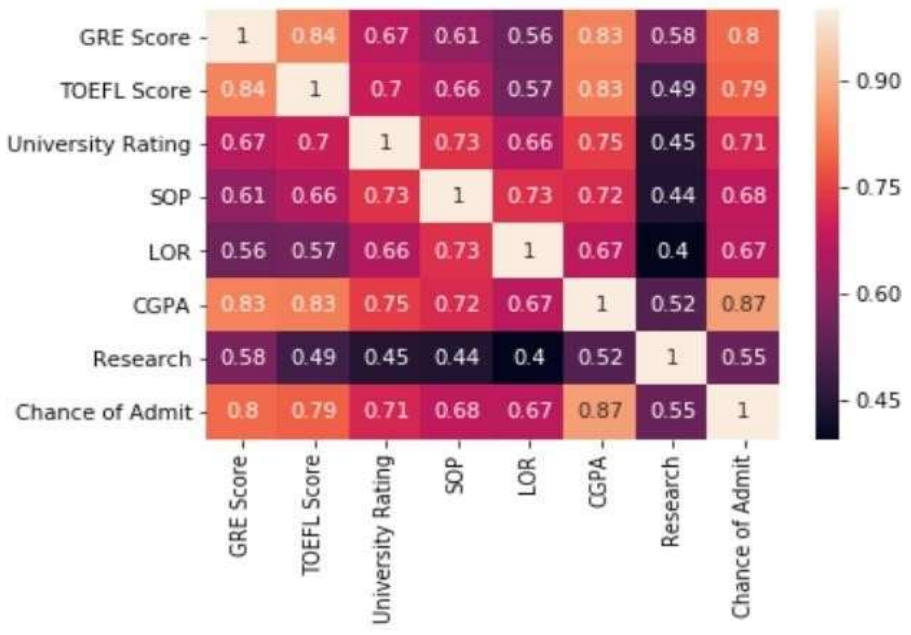
Data was loaded from a text file Admissions.txt and then converted into a numpy matrix for processing. We removed the first column from the matrix since it had no effect on the dependent variable. The data was then shuffled for randomness when we train the model. We separated our data into training, validation and testing data with 240, 80 and 80 data points respectively.

Analysis

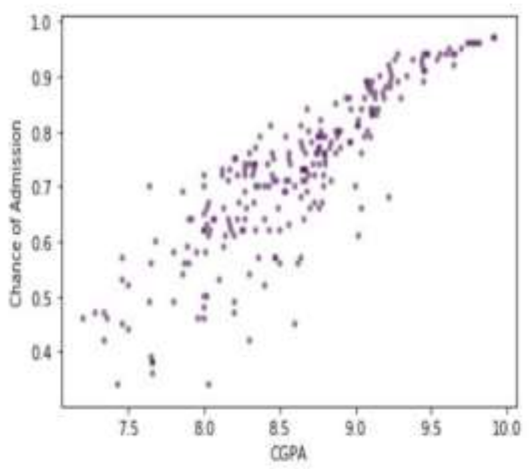
We first look at the scatter plot matrix of all the features. This plot will show us how the independent variables and the dependent variable are related.



There exists a positive correlation between the independent variables and the depended variables. At the same time, the independent variables are also correlated to each which will cause multi-collinearity. The correlation heat map of the whole dataset below shows how correlated the variables are to each other.



This Multi-collinearity creates redundant information (Stepahine, 2015). Multi-collinearity affects the only the independent variables that are correlated. If we had to eliminate some of the variables and fit the model using the only relevant variables, undergraduate GPA has the highest correlation with Chance of Admission with a value of 0.87, meaning simple linear regression would work using just the CGPA as the independent variable.



Multi-collinearity usually affects p-values and coefficients and has little to no influence in the predictions (Jim Frost, 2020). Since our goal is to make predictions, we don’t need to reduce multi-collinearity.

<u>Implementation</u>

We are going to use a closed form solution because our dataset is not that big and computation of matrix inverses will not be costly. We first trained the model using our training data of 240 data points. We will have a [240, 1] labels vector and a [240,7] feature matrix. To allow estimation of the y intercept we added a vector of ones to our feature matrix as the first column. Letting **X** be our design matrix and β as the vector of parameters the linear regression model can be written in the form

$$Y = \mathbf{X}\beta + £$$

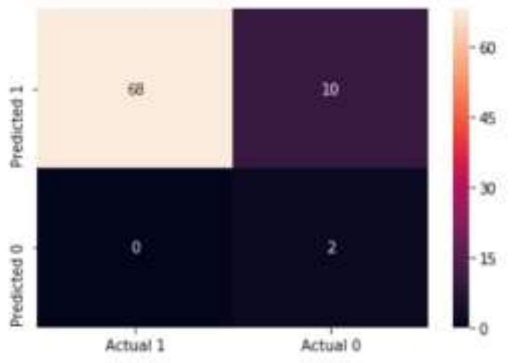
To estimate our parameters in vector form we are going to use

$$\beta = \mathbf{(X'X)^{-1} X'y}$$

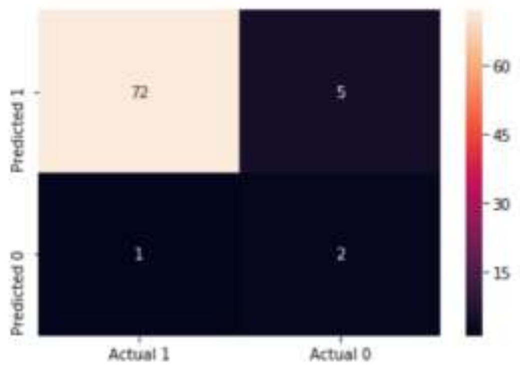
Since the inverse of our design matrix exists the closed form solution will work. After the parameters have been modelled, we now will check the accuracy of our model using validation and testing data.

<u>Results</u>

Our validation data produces a confusing matrix with accuracy of 87.5% having 10 wrong predictions from 80 data points.



Our testing data, which also has 80 data points had an accuracy of 92.5% and mean squared error of 0.075 with the following results table and confusion matrix



	Real y	Predicted y
0	1.0	1.0
1	1.0	1.0
2	1.0	1.0
3	1.0	1.0
4	1.0	1.0
5	1.0	1.0
6	1.0	1.0
7	0.0	1.0
8	1.0	1.0
9	1.0	1.0
10	1.0	1.0
11	1.0	1.0
12	1.0	1.0
13	1.0	1.0
14	1.0	1.0
15	1.0	1.0
16	1.0	1.0
17	1.0	1.0
18	1.0	1.0
19	1.0	1.0
20	1.0	1.0
21	1.0	1.0
22	1.0	1.0
23	1.0	1.0
24	1.0	1.0
25	1.0	1.0
26	1.0	1.0
27	0.0	0.0
28	1.0	1.0
29	1.0	1.0
30	1.0	1.0
31	1.0	1.0
32	1.0	1.0
33	0.0	1.0
34	1.0	1.0
35	1.0	1.0
36	0.0	1.0
37	1.0	1.0
38	0.0	1.0
39	1.0	1.0
40	1.0	1.0
41	1.0	1.0
42	1.0	1.0
43	1.0	1.0
44	1.0	1.0
45	1.0	1.0
46	1.0	1.0
47	1.0	1.0
48	1.0	1.0
49	1.0	1.0
50	1.0	1.0
51	1.0	1.0
52	1.0	1.0
53	1.0	0.0
54	1.0	1.0
55	1.0	1.0
56	1.0	1.0
57	1.0	1.0
58	1.0	1.0
59	1.0	1.0
60	1.0	1.0
61	1.0	1.0
62	1.0	1.0
63	1.0	1.0
64	1.0	1.0
65	1.0	1.0
66	1.0	1.0
67	1.0	1.0
68	1.0	1.0
69	1.0	1.0
70	0.0	0.0
71	1.0	1.0
72	1.0	1.0
73	0.0	1.0
74	1.0	1.0
75	1.0	1.0
76	1.0	1.0
77	1.0	1.0
78	1.0	1.0
79	1.0	1.0

Performance & Critics

The closed form solution doesn't have the best accuracy and removing some of the features which were slightly less relevant than others might have improved the performance. We also could've used gradient descent for an optimized set of parameters and better prediction.

Neural Network

Data Processing for Neural Network

The dataset is loaded using the Admission_Predict.csv and then converted into an array of dataset so that we can be able to analyze and process it better. We then removed the first column in our dataset which shows numbering of data that is, Serial No. and discard it since it is not important

We then split our input features(X) and target feature(Y) into two arrays, *X (all features with the exception of Chance of admit column)* and *Y_(Chance of admit column).* We normalized our target feature (Y_ array) into classification problem using the threshold 0.5 and round each values.

Processing our data was done by using preprocessing library sklearn to make sure that the scale of input features is similar by making it to lie between (0 – 1), this was done to avoid difficulties for the initialization of our neural network.

We divided our data into training - 60%, validation – 20% and testing – 20% using sklearn model selection library.

Implementation

We used an architecture of one hidden layer with 20 neurons rather than more hidden layers with few neurons for efficiency, in total it has 3 layers including the input layer with 7 input features, the output layer only has one node since this problem is a classification problem.

We have two weight matrices which are W1 and W2 and the dimension are (7 x 20) and (20 x 1) respectively. We also have two bias vectors b1 and b2 of size 20 and 1 respectively. T help our neural network to be able to learn correctly by updating the weights. We used dictionary to store weights and biases.

We used two activation functions for our neural network, that is the sigmoid function and Rectified Linear Unit (ReLU)

- Sigmoid function – we use sigmoid function for our ouput layer, it takes in input value and squashes it to a real valued output between 0 and 1

- ReLU activation function - does a threshold operation to each input values of our features and check whether the values are less than zero or not, if the values are less than zero they are set to zero.

We also have our loss function which measures how well our neural network learns over time. Loss function = minus sum of our true values times log of y-hat where our sum starts from one to two since it is a classification problem.

For training we used forward propagation and backpropagation with no gradient descent

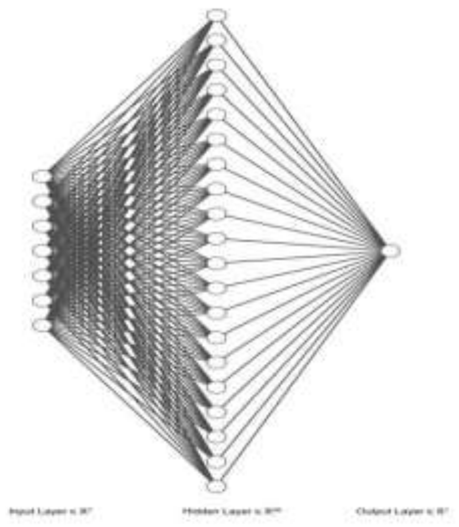
For forward propagation, we have weighted sum between first layer's weights and biases, then we apply our ReLU. We then have weighted sum between the output of our previous computation and second layer's weight, then we calculate the loss between predicted and true value.

Backpropagation was used to make prediction and update our weights and biases. The use of partial derivative make it possible for our neural network to update weights and basis as shown in the code.

The learning rate is 0.01 to avoid our neural network from learning too fast to avoid divergence and too slow to avoid. Our iterations are 100, the higher the iteration the more likely our neural network will over-fit.

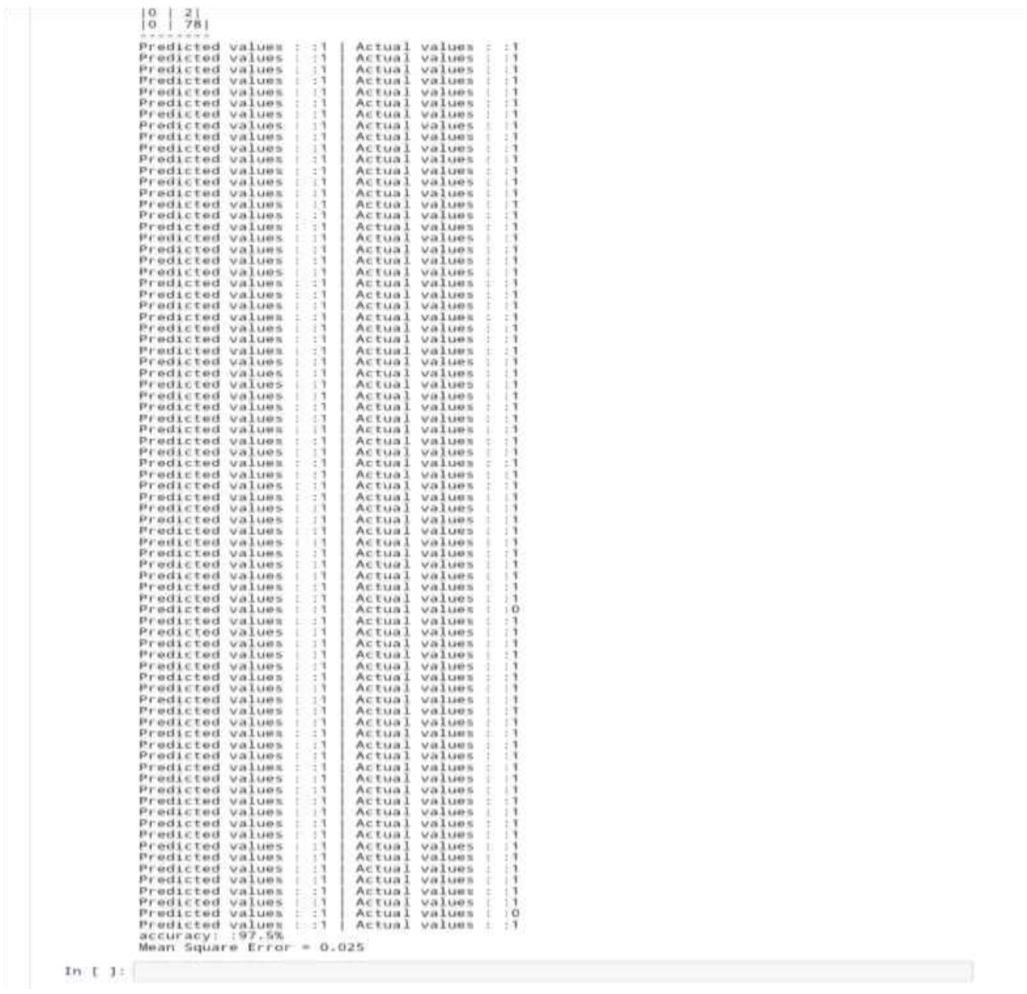
Neural Network Architecture

- The first set of 7 nodes is the input
- The second set of 20 nodes is the hidden layer
- The last node is the output



The results of Neural Network:

Testing results for predicted values and testing values.



```
|0  |  2|
|0  | 78|
--------
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :0
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :1
Predicted values : :1 | Actual values : :0
Predicted values : :1 | Actual values : :1
accuracy: :97.5%
Mean Square Error = 0.025
```

In [ ]:

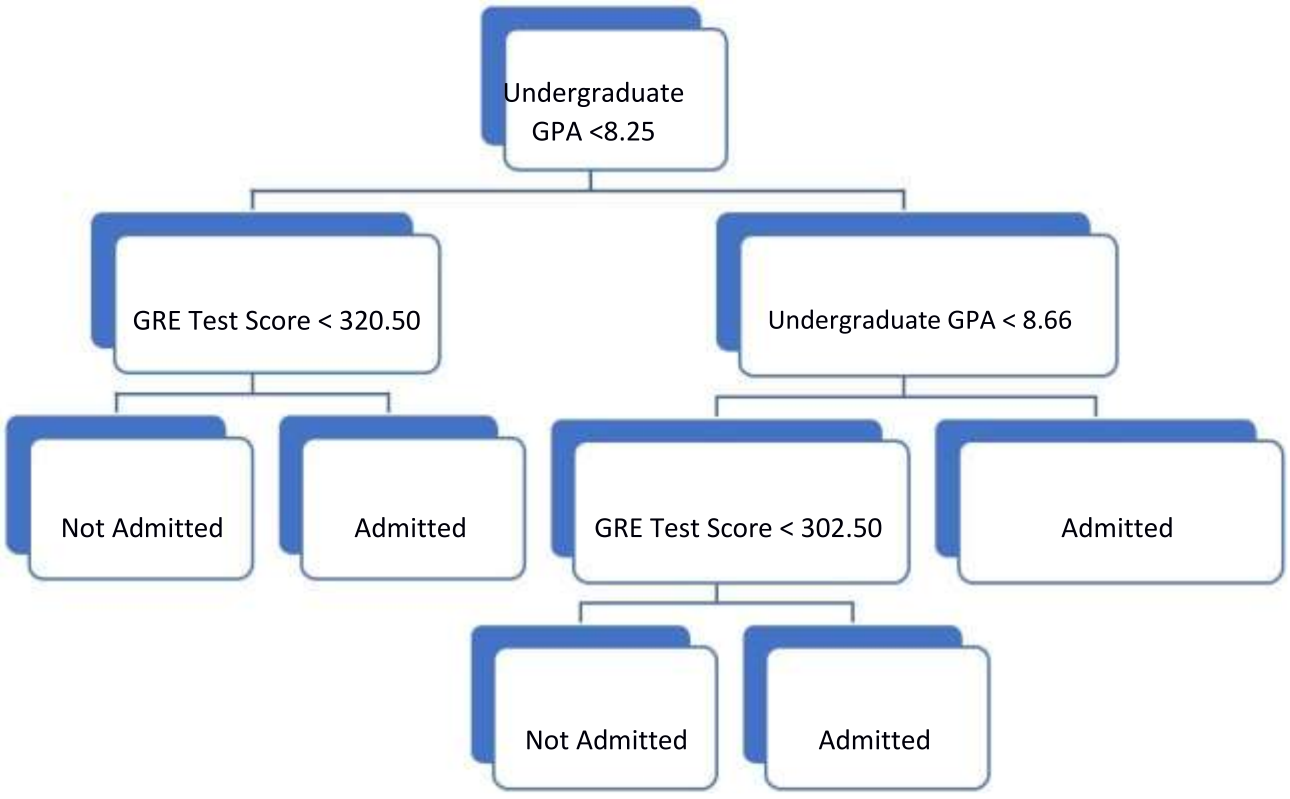
Performance:

Our neural network works fairly good, it can been seen in the logloss training graph that the loss decreases exponentially throughout the iterations. The accuracy of out testing is 97.5% (78) with Mean Squared Error of 0.025. It has managed to get everything right with the exception of the remaining 2.5% (2) which shows that it did pretty well although sometimes it fails to even worst.

Critics:

The best possible way to increase performance is to add more hidden layers with more nodes. We could also increase epochs (iterations) for learning. We could also change activation functions and try other ones to check which one works much better. If the dataset was quite large it would make it much efficient and reliable. We could have made our neural network learn with gradient descent in backpropagation

Classification and Regression Tree



Our data consists of 400 rows of raw data. The dataset can be found here https://www.kaggle.com/mohansacharya/graduate-admissions (in its raw form)

[about data]

The raw data has target class as a continuous value. To make our lives easier we converted the target class to discrete values. If the probability of being admitted is greater than 0.65 one would be admitted otherwise not admitted. The threshold '0.65' is based on the fact it is used by most institutions as initial discriminative value.

[about the algorithm]

This algorithm was chosen because we are dealing with simply classification problem with few number of features and data points. Furthermore, the above algorithm is better suited to make predictions given the nature of our features, which compromise of continuous values. Initially, we tried to convert the values of each feature to discrete ones e.g. we tried to make GRE test score to have low average and high score but because our data compromises of only high scores the decision tree with only discrete values would perform badly as it never sees low or average scores it won’t learn to classify data with those inputs, this is true for other features.

[implementation]

By considering the latter, this made it ideal to implement decision which discriminates based on intervals. The metric that used to split the data was GINI impurity. If all the data in a node belong to one class, then that node is pure (*G = 0*) and therefore does not need to be divided further otherwise it is impure and it should be divided. More formally the Gini impurity of *n* training samples split across *k* classes is defined as G = 1 – sum(Pk^2) for all k in [1,n] where P*k* is the fraction of samples belonging to class *k*. The key to the Classification And Regression Tree algorithm is finding the optimal feature and threshold such that the Gini impurity is minimized. Given that the features have continuous values we had sort the values for a given feature, and consider all midpoints between two adjacent values then look for threshold.

[about the performance]

The best possible performance that can be archived is by increasing the maximum depth of a tree plausibly to 6 or 7. In this particular case I chose the maximum depth to be 3. The above Decision Tree uses this convention, go to the left if the statement in the node is True and go right if it is false.

We used 280 amount of rows to train the decision tree model, the decision tree then presented an accuracy of 87.5% when being tested through 120 rows of the data.

Reference

[1] Stephanie Glen. "Multicollinearity: Definition, Causes, Examples" From StatisticsHowTo.com: Elementary Statistics for the rest of us! https://www.statisticshowto.com/multicollinearity/

[2] Towards Data Science Site (Article) https://towardsdatascience.com/how-to-build-yourownneural-network-from-scratch-in-python-68998a08e4f6

[3] NN-SVG (for drawing neural network) http://alexlenail.me/NN-SVG/LeNet.html

[4] 3Blue1Brown (Deep learning chapter 1) https://www.youtube.com/watch?v=aircAruvnKk

[5] Python Machine Learning Tutorial (NN) https://www.python-course.eu/neural_networks.php

[6] Statistics and Machine Learning in Python Release 0.3 beta Edouard Duchesnay, Tommy Löfstedt, Feki Younes

[7] Multicollinerity in Regression Analysis: Problems, Detection and Solutions https://statisticsbyjim.com/regression/multicollinearity-in-regression-analysis