## 1 Abstract

The objective of this project is to perform real world Data Analytic tasks from the beginning to the final stage. The chosen datasets are race data from Formula 1 seasons from the year 1950 to 2017. These datasets can be obtained from Kaggle.

Formula One (F1) is one of the most popular motorsports in the world, with a high number of viewership globally. Multiple Constructors compete against each other in order to win the Constructor Championship (CC). Multiple Drivers compete against each other in order to win the Driver Championship (DC). By winning the CC, the Constructor will be allocated a larger budget to be used in the next season of F1. As such, Constructors should come up with a race strategy that would provide them the best advantage to win the CC.

The first way to improve race strategy is to predict if the racer will be in Point Winning Position (PWP) in a race. If he is predicted to not be in PWP, the Constructor could limit the usage and performance of the power unit, in order to preserve the life span of power unit elements for future races. This could help with the team’s overall race strategy. After performing KDD, we are able to achieve a decent accuracy rate for predicting whether a driver will be in PWP.

The second way is to leverage on the additional point gained by achieving the fastest lap time in a race. By predicting what the fastest lap time is for a race, the driver could try to hit that timing early on in the race and focus on getting into PWP afterwards. This additional point gained could affect the overall driver standing in the DC. After performing KDD, even though the size of the dataset is small, we were still able to achieve a decent accuracy rate in our prediction.

The third way is to have a driver that is compatible with the Constructor. After each season, Constructors could bid for their driver of choice. By predicting which driver performs better with the Constructor, Constructors could bid for that driver which could improve the chance of winning the Championships. After performing KDD, we were able to achieve a respectable accuracy rate in our prediction.

During the actual race, there could be other features that could have stronger correlation to the target labels that was not included in the datasets. Thus, by including these features, we were able to further increase our first model’s prediction by …

Therefore, this project has taught us valuable insights into the world of KDD and how it could be used to solve real world problems.

## 

## 2 Introduction

Formula One (F1) is one of the most well-known international motorsports. A F1 season consists of a series of races, known as Grand Prixs (GPs), which take place worldwide on purpose-built circuits and public roads. The results of each race are evaluated using a points system to determine two annual World Championships: Driver Championship (DC) and Constructor Championship (CC). Every season, F1 teams, also known as Constructors compete using cars with chassis built by themselves. Some examples are major car companies like Ferrari. Every Constructor in F1 must run two cars in every session in a GP weekend, and every Constructor may use up to four drivers in a season.

#### 2.1 Grand Prix

##### 2.1.1 Practice Session

Practice sessions are conducted on Fridays where drivers could get accustomed to the circuit.

##### 2.1.2 Qualifying Session

Qualifying sessions are conducted on Saturday. This will determine the starting grid for the actual race. Qualifying sessions consist of three periods, known as Q1, Q2, and Q3. In each period, drivers run laps with the slowest drivers knocked out of that period. The rest of the drivers continue on to the next period. The drivers that were knocked out solidifies their grid position for the race based on their lap time and will not be participating in the next period. Q1 determines the bottom 5 placing, Q2 determines the next 15 while Q3 determines the final 5. Drivers are allowed to run as many laps as they wish within each period and only the fastest timing will be taken into account. In the end, the drivers starting grid positions are determined.

##### 2.1.3 Actual Race

The actual race will be conducted on Sunday. Drivers start at the starting line based on starting grid position determined by their results in Qualifying. The drivers then raced for a predetermined number of laps on the track with the final race standings determined by when the drivers crossed the finishing line after completing all laps. In the current format, points are awarded to the first 10 drivers that finished the races, therefore the top 10 positions in a race are also known as Point Winning Positions (PWPs) . These points contribute to the DC and CC where the constructors points for each race are just the points obtained by the Constructor’s two drivers for that particular race.

## 2 Problem Description

### 2.1 Motivation

The main goal of a Constructor is to win the CC in order to be allocated a bigger budget in the next F1 season. The main goal of a Driver is to be placed in the PWP for each race in order to gain points towards the Driver’s Championships. These points earned will also be contributed to the driver’s own Constructor, which is used for the Constructor’s Championship. In order to win, the Constructor will need to have the best race strategy for the season in order to increase the chances of winning.

To win the DC, drivers must be placed in PWP for each race. Points earned will decrease based on the position. The driver points are accumulated throughout each race and after the final race, the driver with the highest driver points in the Final Drivers’ Standings will win the DC.

To win the CC, drivers under the Constructor must be placed in PWP. The points the drivers earn in each of the races are added to the Constructor points. The Constructor points are accumulated throughout each race and after the final race, the Constructor with the highest Constructor points in the Final Constructor Standing will win the CC.

However,it is difficult to determine if a driver will be in PWP for a particular race in a particular season. By predicting if a driver will be in PWP, we could determine the type of strategy the Constructor will execute for that race.

In F1, there is one way to earn additional points and that is to be the driver that achieves the fastest lap time for the race and finishing in PWP. This will award the driver with one additional point which could affect the overall driver point standing. However, it is hard to determine what will be the fastest lap time for a race in a season. If this could be predicted, the Constructor could notify the driver of the predicted time before the race and he could try to hit the time early on instead of waiting until the end of the race.

At the end of each season, Constructors are able to bid for new drivers. In order to prepare for the next season of F1, one of the strategies that the Constructor could execute is to bid for better drivers for their constructor. By determining if a specific driver will perform better in their constructor, the Constructor will know which driver to bid for in order to perform better in the next season.

## **2.2** **Problem Definition**

Objective: Devise a race strategy that will allow the Constructor to win the Constructor Championship.

### **2.2.1** **Problem 1**

To earn driver points, drivers must finish in a PWP in a race. However, it is not easy to predict whether a driver will finish in PWP. In addition, the amount of used F1 power unit elements that drivers are allowed during the season is limited. During the 2019 F1 season, drivers are allowed to use only 3 ICE, 3 TC, 3 MGU-H, 2 MGU-K, 2 ES and 2 CE units. Drivers that use more power unit elements will receive grid penalties in the next F1 race. Therefore, if a Constructor is able to predict that they are not able to finish in a PWP for a particular race, they can limit the usage and performance of the power unit, in order to preserve the life span of power unit elements for future races, avoiding grid penalties which could be a better strategy in the long run.

### **2.2.2** **Problem 2**

By finishing the race in PWP and achieving the fastest lap time, the driver is awarded an additional point towards his driver points. However, drivers normally achieve their fastest lap time during the last few laps of the race. If we could predict the fastest lap time for a particular race in the season, the Constructor could inform the driver of what time to hit before the race. The driver could try to hit this timing early on and focus on finishing in PWP afterwards.

### **2.2.3** **Problem 3**

When bidding for new drivers, it is hard to determine whether a driver is able to achieve a better performance using a new constructor. If we could predict how a driver performs when they switch to a new constructor, the Constructor could select the driver that performs the best and bid on them after the season ends. This would be one step towards the team’s goal of winning the CC.

### **2.2.4** **Problem 4**

As an extension to Problem 1, instead of performing a binary classification to predict if a driver will finish in a Points Winning Position in a race, we will perform a multiclass classification to predict the results of a driver in a race into more specific categories. This will also allow us to better understand which category of race classification is more predictable and which categories are harder to predict. We will also be utilizing another dataset for Formula One data, retrieved from Kaggle.

# 

# 3 Approach

**3.2** **Methodology**

#### Software Requirements

For this assignment, we will primarily be using Jupyter for our EDA, Data Cleaning and Data Mining.

#### Project Components

There are 3 components in our project:

1. Report Documentation: Documents our journey of this assignment
2. Source Code: Contains the datasets which are in csv format and Jupyter files to execute the codes
3. README.md file: Provides instructions on how to execute the Jupyter files

#### Sources and Datasets

The chosen datasets contain data of F1 seasons from the year 1950 to 2017. The data consists of tables describing constructors, race drivers, lap times, pit stops and more. This original dataset is provided by Chris G and is made publicly available on [Kaggle](https://www.kaggle.com/cjgdev/formula-1-race-data-19502017).

# 

## 4 Implementation

### 4.1 Data

After analysing the data, we decided to drop csv files that do not provide useful information. These are **constructors**, **constructorResults**, **constructorStandings**, **lapTimes**, **pitStops** and **seasons**.

The reasons are as follows:

* **constructors**, **constructorResults** and **constructorStandings**: Since Constructor point is equivalent to the cumulative driver points and constructorId exists in other datasets such as **results**, these three datasets were classified as duplicate information.
* **lapTimes**: This was considered as live data, as in unpredictable data that would not be helpful in solving the problems.
* **pitStops**: Pit stops strategy depends on the compound of the tires and since we do not have tire data, it is difficult to use pit stop information.
* **seasons**: This dataset consists of redundant information.

We kept the following csv files and the reasons are as follows:

* **results:** Contains the results of each race for each Grand Prix for each season.
* **circuits:** Contains circuit information for circuits used during Grand Prix
* **drivers**: Contains drivers information such as name and driver id.
* **driverStandings**: Contains the cumulative points for each driver in each race in each season. It also includes whether the driver won the race and their final position.
* **qualifying:** Contains the qualifying times for all 3 periods during the qualifying round.
* **races:** Contains race information for each season
* **status**: Contains the status that a driver and Constructor could have at each race.

### 

### 4.2 Data Pre-Cleaning

Before we start, we conducted a pre-cleaning of the datasets. This would make it easier for us to conduct our experiments. To summarise, we did the following:

1. Removed redundant attributes from most dataset, e.g: driverRef, url.
2. Convert time string from minute-second format to milliseconds.
3. Merge driver’s name into 1 attribute instead of 3.

Once this is completed, we export the datasets into a new csv file with “\_cleaned” appended to the end of the name. These files are stored in a folder called Cleaned Data.

### 4.2 Data Insertion

#### 4.2.1 Adding of Additional Information in circuit.csv

While studying the datasets, we discovered that the circuits.csv does not provide important information about the circuits. The original file only has basic information of where the circuits are located and not important information such as lap length, race length, etc. These additional information could be easily obtained from trustworthy websites such as <https://www.racefans.net/>. As we believed that this information is needed for our Data Mining tasks, we manually inserted the following information into the circuit\_cleaned.csv file based on information from the website.

* Turns, Lap Length, Race Laps, Race Distance, Max. Speed, DRS Zone, Full Throttle Percentage, Longest Flatout Section, Downforce Level, Gear Changes per Lap

#### 4.2.2 Categorizing Values in status.csv

In the status.csv file, we have 136 different status that a driver could have for each race in each season. Most of the status means the same thing and thus, could be binned into one category. As such, we have decided to manually bin the values inside status.csv into these 4 categories:

* Finished, Team, Track, Constructor

### 

## 5 Experimental Results and Analysis

### Problem 1 - Predict whether a driver will finish in a PWP

### Data Integration

We integrated the following data frames in this order:

* A = Merge Left, results\_df and status\_df on ‘statusId’
* B = Merge Left, A and races\_df on ‘raceId’
* C = Merge Left, B and circuits\_df on ‘circuitId’
* D = Merge Left, C and drivers\_df on ‘driverId'

### Data Cleaning

We cleaned the data using the following steps:

* Filter all rows to year between 2004 to 2017
* Remove all rows that have NaN value for ‘error\_category’
* Drop the following attributes name, time, milliseconds, points, laps, fastestLap, rank, fastestLapTime, fastestLapSpeed, status

After cleaning, we exported this dataset into a new csv, ‘combined-initial.csv’.

### Data Transformation

* We normalized the data by changing the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. This will bring all the variables to the same range so that no input feature is more important as a predictor than the others to the classification model.

### Experimental Setup

#### Initial Dataset

For this problem, we used the combined-initial.csv which contains the following features:

* Driver’s ID, Constructor’s ID, Driver’s Starting Grid, Circuit ID, Number of Turns in the Circuit, Lap Length of the Circuit, Number of Race Laps, Race Distance of the Circuit, Maximum Race Speed of the Circuit, Number of DRS Zones, Percentage of the Circuit that Drivers are able to Full Throttle, Longest Flatout Section of the Circuit, Downforce Level of the Circuit, Gear Changes Per Lap

These features contain information that is available to the Teams before the start of the race such as Track Features and Starting Grid Position. Using this information available pre-race, we will like to develop a classification model that is able to predict whether a Driver will end up in a PWP based on the input features.

For this, we first have to create the labels, or targets for the classification model. The labels are binary classes where ‘0’ represents that the Driver will not be in a PWP and ‘1’ represents that the Driver will be in a PWP.

For this Classification problem, we fitted the dataset on 4 different classification models. The 4 classification models used are:

* Support Vector Machine(SVM)
* Naives Bayes Classifier
* Decision Tree Classifier
* Random Forest Classifier

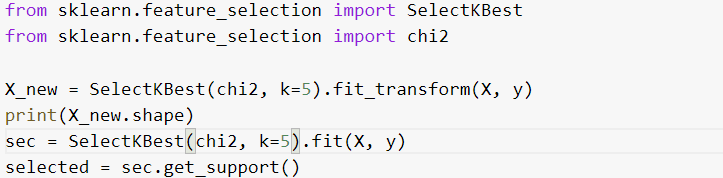
#### Dataset with Added Input Features

Adding new features could improve the model’s accuracy. One of the features we thought of is to calculate the error rate based on the ‘error\_category’ attribute. From this, we can derive 3 features, Driver error rate, Constructor error rate and Circuit error rate. These features are added to provide information on the percentage of races where Drivers, Constructors and Circuit incidents, prevented the Drivers from completing the race and dropped out of the PWP.

#### Dataset with Reduced Number of Features

A large number of input features may not necessarily result in a better fitted classification model and by having a smaller number of features would allow us to avoid the Curse of Dimensionality.

Therefore, for the last set of experiments, we used the *sklearn* function called ***SelectKBest()*** to select the k best input features based on **chi-squared stats,** where chi-squared stats between each non-negative feature and class are computed. This score can be used to select the n\_features with the highest values for the test chi-squared statistic from X, which must contain only non-negative features such as booleans or frequencies, relative to the classes. Chi-square test measures dependence between stochastic variables, so using this function “weeds out” the features that are the most likely to be independent of class and therefore irrelevant for classification.



Using the above method, these are the selected input features used for the classifiers:

Selected features:

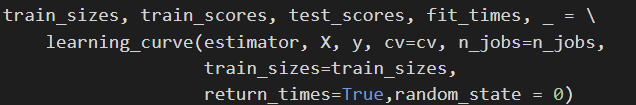
* driverId, constructorId, grid, driver\_error\_rate, constructor\_error\_rate

### Comparison Schemes

In order to determine the best classification model to use, we split the data into training and test sets using the dataset obtained from ‘combined-initial.csv’. This is done using the *sklearn* function called ***ShuffleSplit()***. The test size of each cross validation split is 20 percent of the dataset.



We fitted the 4 different classifiers using the cross validation dataset obtained from the ***ShuffleSplit()*** function. We fit the 4 different classifiers using the *sklearn* function called ***learning\_curve()***. The function determines cross-validated training and test scores for different training set sizes. A cross-validation generator splits the whole dataset, k times in training and test data. Subsets of the training set with varying sizes will be used to train the estimator and a score for each training subset size and the test set will be computed. Afterwards, the scores will be averaged over all k runs for each training subset size.



The best classification model is selected based on which model has the highest test accuracy, F1 score, as well as ROC AUC score obtained after fitting all the training examples to the 4 classification models.

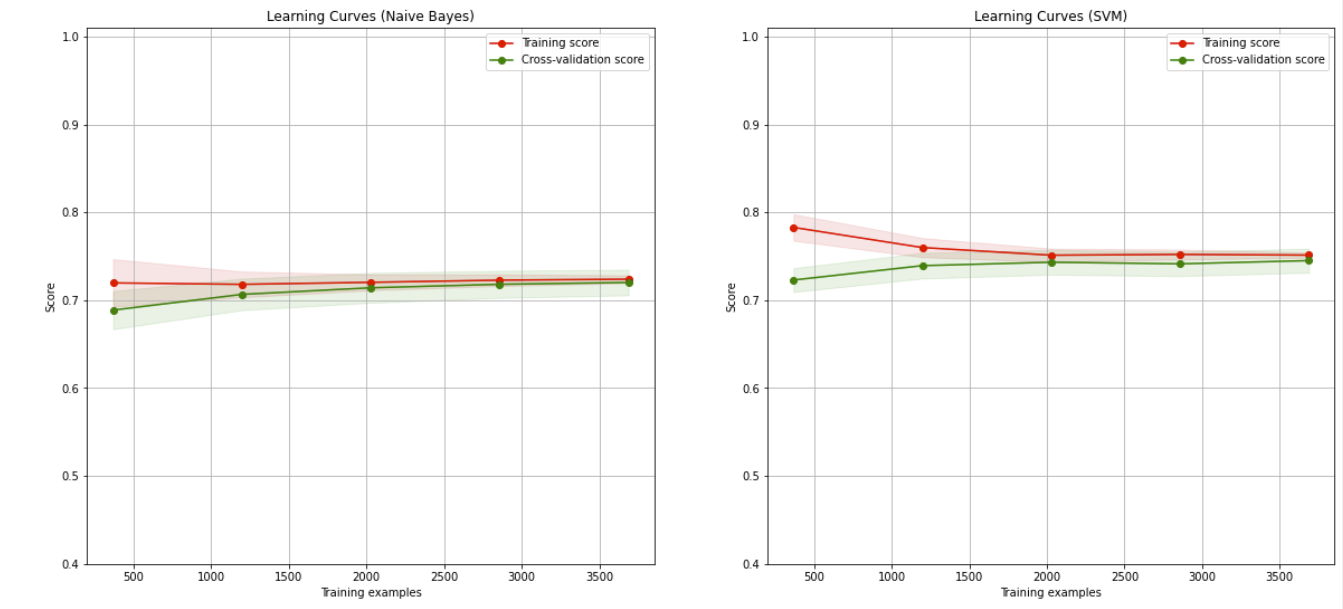
* The test accuracy tells us how accurately the classification model predicts the target label correctly.
* F1 score conveys the balance between the precision and the recall of the model, where precision is the number of True Positives divided by the number of True Positives and False Positives and recall is the number of True Positives divided by the number of True Positives and the number of False Negatives.
* ROC AUC score computes the Area Under the Receiver Operating Characteristic Curve to tell how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting the true positives and true negatives in the data.

### 

### Results and Analysis

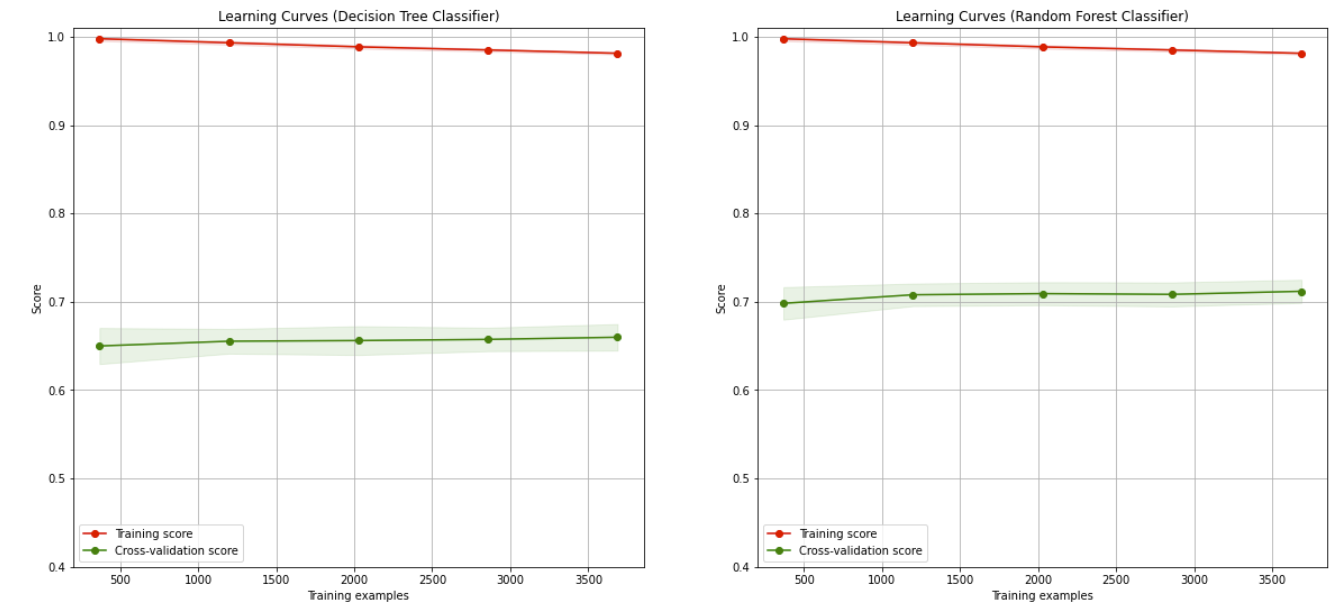
#### Initial Dataset

The training and test scores obtained after fitting the dataset to the 4 classifiers are plotted against the number of training examples. The results of the experiments for the initial dataset is shown below:



Naive Bayes Validation Accuracy: 71.70596205962062%

Support Vector Machine Validation Accuracy: 74.09214092140924%



Decision Tree Accuracy: 66.15040650406505%

Random Forest Accuracy: 71.54878048780492%

#### Test Accuracies Comparison of All Datasets

|  |  |  |  |
| --- | --- | --- | --- |
|  | Initial Dataset | Dataset with Added Features | Reduced Dataset |
| SVM | 75.623% | 76.056% | 76.273% |
| Naives Bayes | 74.000% | 74.431% | 74.864% |
| Decision Tree | 68.797% | 68.689% | 73.564% |
| Random Forest | 72.156% | 72.589% | 74.431% |

\* Tables for F1 Score, ROC AUC and Cross Validation can be found in Appendix: 8.1.1

From the results obtained, we can observe that the SVM classifier performed the best with the highest test accuracy, F1 Score as well as ROC AUC Score. From the experiments, we can observe that the added features helped improve the accuracy of 3 of the classifiers except for the Decision Tree Classifier. Even when we created the reduced features dataset based on chi-squared stats, 2 of the 3 added features were selected as input features for the classification models and those features are more strongly correlated to the target variables as compared to most of the input features in the initial dataset. This shows that the added features are relevant features to the classifiers in predicting whether a driver will finish in a PWP. From the results, the classifiers perform the best when using a reduced dataset, which implies that we should always eliminate redundant features in order to achieve better results for any prediction model.

#### Using Neural Network for Prediction

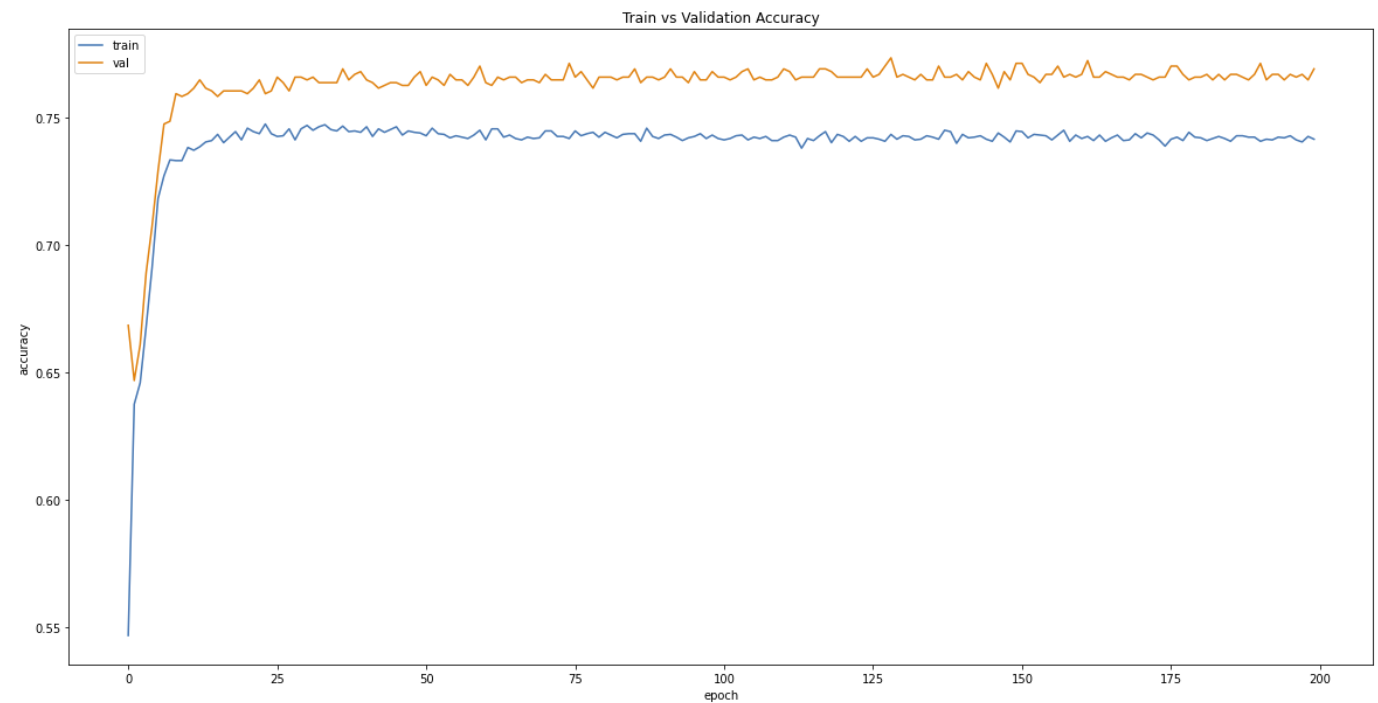
After obtaining the results with the various classifiers above, we would like to see if using a neural network would be able to achieve a higher accuracy.

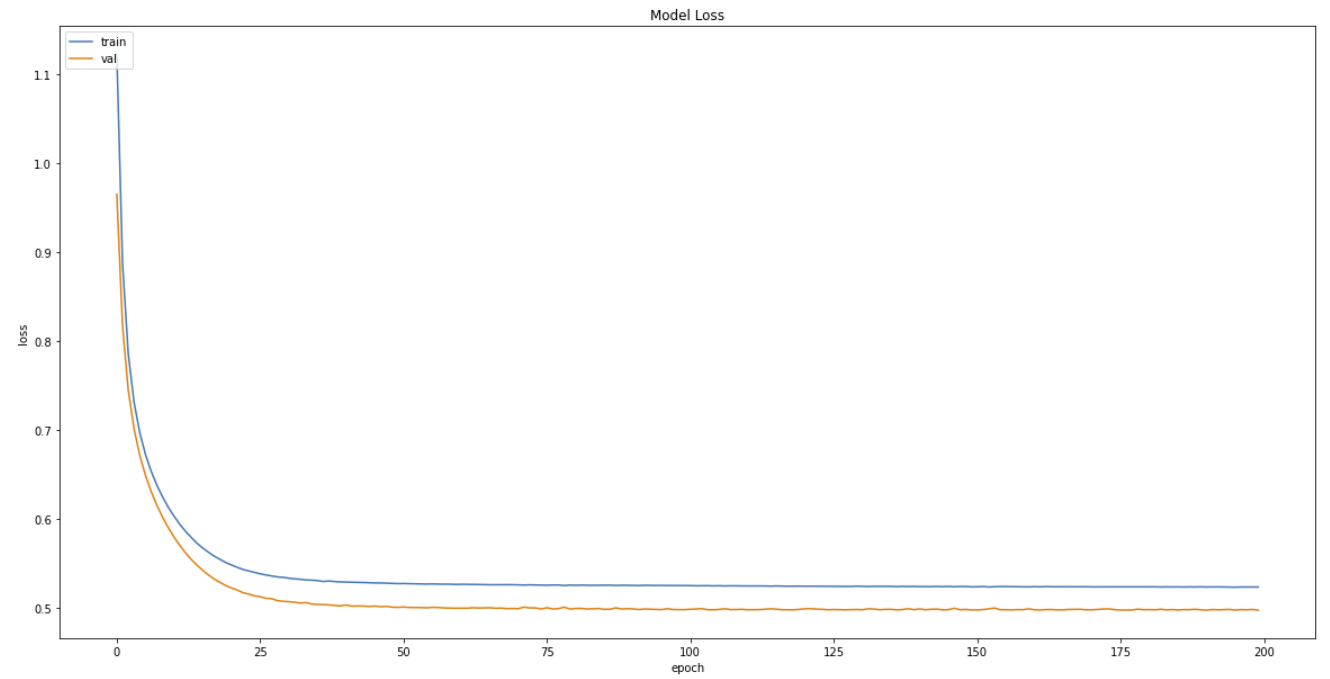
For this, we utilized a 4-layered Neural Network to perform the regression. The neural network consists of:

* Input layer
* Dense Layer of 20 neurons
* ‘Softmax’ Dense Layer
* Output Layer

We then used the SGD optimizer for the neural network with a learning rate of 0.001 and a learning decay rate of 1e-6. The model is trained for 200 epochs on 80 percent of the original dataset and then validated on 20 percent of the dataset for each epoch.

The input dataset that we used for this classification neural network is the reduced features dataset which consists of 5 input features.





From the first graph, the neural network manages to learn the classification rather quickly, with the validation accuracies converging even before 25 epochs. The validation accuracies converges around 76% to 77%.

We saved the model with the highest validation accuracy and made predictions on the test set. We compare the predictions against the actual class labels to give us the test accuracy of the model. Using the neural network classifier, we are able to achieve 77.36% test accuracy.

### Problem 2 (Predict the Fastest Lap Time for a Race)

To create the data needed for our experiment, we needed a dataset that consists of the circuit's information and fastest lap time from the previous years. Hence, we consolidated a series of data integration, cleaning and data transformation processes through the following steps.

### Data Used

* qualifying\_clean.csv, races\_df, circuits\_df, results\_clean.csv

### Step 1

We imported qualifying\_clean.csv and performed data cleaning and data transformation in this order:

* Remove all rows that have NaN value for “q1”
  + We realised that when “q1” is NaN, the drivers did not qualify for any other runs and gave us a NaN for getting the fastest qualifying runs timing.
* Generate “fastest\_q” column from the fastest timing within q1, q2 and q3.
  + This is done to get the fastest timing within all their qualifying runs

### Step 2

We imported Race\_df and Circuit\_df and performed data Integration to get a dataframe that consist of circuit informations and year of the races

* Merge Left races\_df and circuit\_df on “circuitId”.
* Drop columns of “name\_x”, “name\_y” and “country” because it was not required.

### Step 3

Next, we performed data integration for the dataset from step 1 and step 2 to obtain a dataset that consists of the years, circuit information and fastest qualifying run timing.

### Step 4

We import results\_clean.csv that consist of results information. We performed data transformation and data cleaning to obtain the fastest lap timing with the following steps:

* Performed group by “raceId” and obtained the best Lap time from “fastestLapTime” with aggregation max.
* We proceed to drop NaN values from fastestLapTime.

### Step 5

Next, we performed data integration for the dataset from step 3 and step 4 to obtain a dataset that consists of the years, circuit information, fastest qualifying run timing and fastest lap time.

* Merge Left step3 and step4 on “raceId”.

### Step 6

Next for the most important step, we performed data transformation on data obtained from Step 5 to acquire the fastest lap from the previous years of the same circuit.

* We group by “circuitId” and “year” to get the fastest Lap Time
  + This is done because every circuit Id and year has their own fastest lap time
* We then append 4 years worth of fastest lap data to the data from Step 4

This gave us a dataset that contains all the years, circuit information, and a total of 5 years worth of fastest lap time within the same circuit.

After all these steps were performed, we exported this dataset into a new csv, question2\_new.csv’

#### Initial Dataset

For this problem, we first used the question2\_new.csv which contains the following features:

* Driver ID, Constructor ID, Year, Number of Turns in the Circuit, Lap Length of the Circuit, Number of Race Laps, Race Distance of the Circuit, Maximum Race Speed of the Circuit, Number of DRS Zones, Percentage of the Circuit that Drivers are able to Full Throttle, Longest Flatout Section of the Circuit, Downforce Level of the Circuit, Gear Changes Per Lap, Fastest Qualifying TIme for Driver, Previous 1 Year Fastest Lap TIme, Previous 2 Year Fastest Lap TIme, Previous 3 Year Fastest Lap TIme, Previous 4 Year, Fastest Lap TIme, Fastest Lap Time (Target to predict)

By using the input features above where we have the circuit information, the history of the past 4 years’ fastest lap times of the circuit, we can design a regression model to try to predict the fastest lap time for a particular circuit for a particular year.

We normalized the data by changing the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values.

For this regression problem, we utilized a 4-layered Neural Network to perform the regression. The neural network consists of:

* Input layer
* Dense Layer of 10 neurons
* Dropout Layer of 0.2 Dropout Rate
* Dense Layer of 10 neurons
* Dropout Layer of 0.2 Dropout Rate
* Output Layer

We then used the Adam optimizer for the neural network with a learning rate of 0.001 and a learning decay rate of 1e-6. The model is trained for 500 epochs on 75 percent of the original dataset and then validated on 25 percent of the dataset for each epoch.

#### Reduced Dataset

A large number of input features may not necessarily result in a better fitted regression model and having smaller input features would allow us to avoid the Curse of Dimensionality.

Therefore, for the last set of experiments, we used the *sklearn* function called ***SelectKBest()*** to select the k best input features based on the f\_regression function in *sklearn*. The function uses a Linear model for testing the individual effect of each of many regressors. This is a scoring function to be used in a feature selection procedure, not a free standing feature selection procedure.

This is done in 2 steps:

1. The correlation between each regressor and the target is computed, that is, ((X[:, i] - mean(X[:, i])) \* (y - mean\_y)) / (std(X[:, i]) \* std(y)).
2. It is converted to an F score then to a p-value.

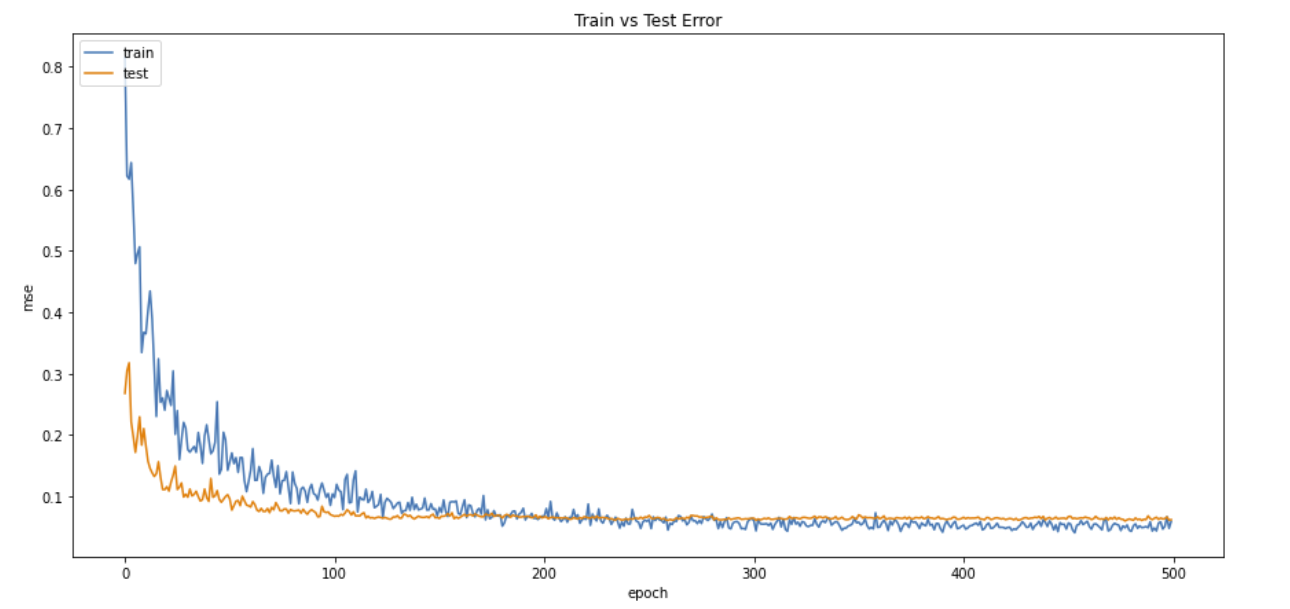
Using the above method, these are the selected input features used for the model:

* turns, lap\_length, race\_laps, fastest\_q, prev\_year\_1\_lap, prev\_year\_2\_lap, prev\_year\_3\_lap, prev\_year\_4\_lap

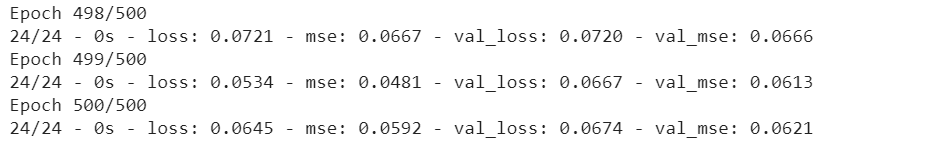
### Results and Analysis

#### Initial Dataset

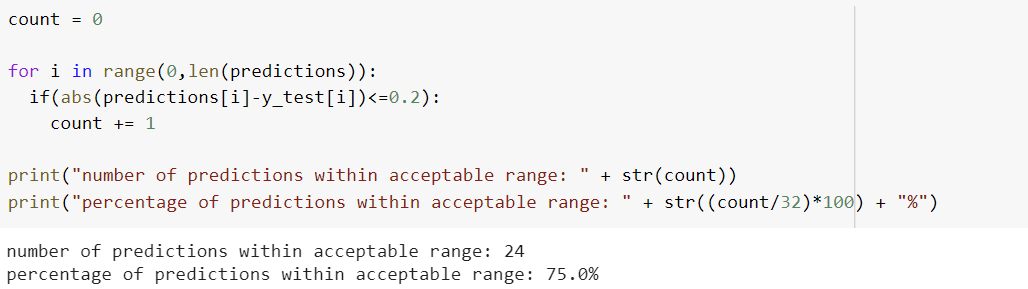
As this is a regression problem, we monitor the Mean Squared Error (MSE) of the regression model and the model is trained to give us the lowest M.S.E possible which reduces the error between the predicted fastest lap time value and the actual fastest lap time value. Below shows the plot of M.S.E against the number of training epochs for the model:



Towards the last 100 epochs, the M.S.E of the regression model converges to around 0.06 for each epoch.



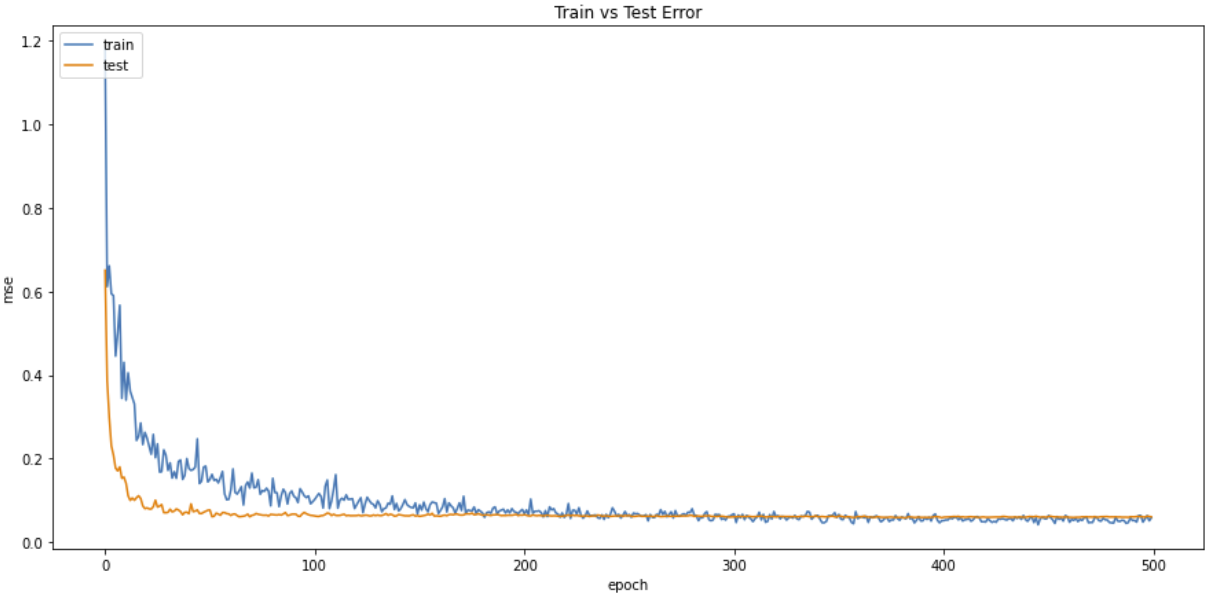
We also compared the predicted fastest lap times against the actual fastest lap times and observed the number of predicted fastest lap times that are within 0.2 minute of the actual fastest lap times.



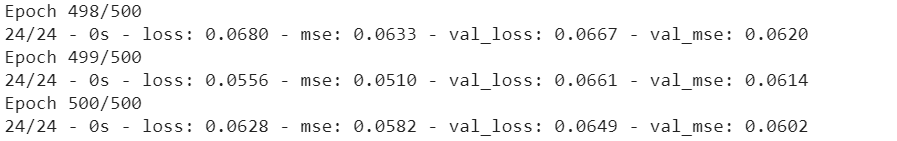
The regression model was able to give us 75 percent of predictions within acceptable range as can seen from the figure above.

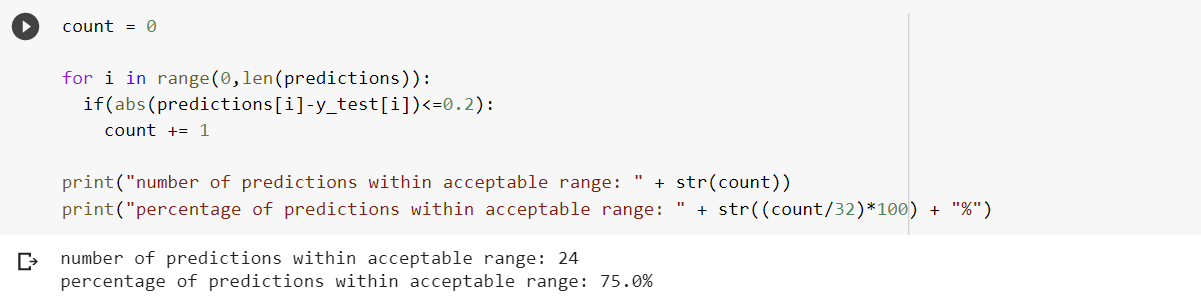
#### Reduced Dataset

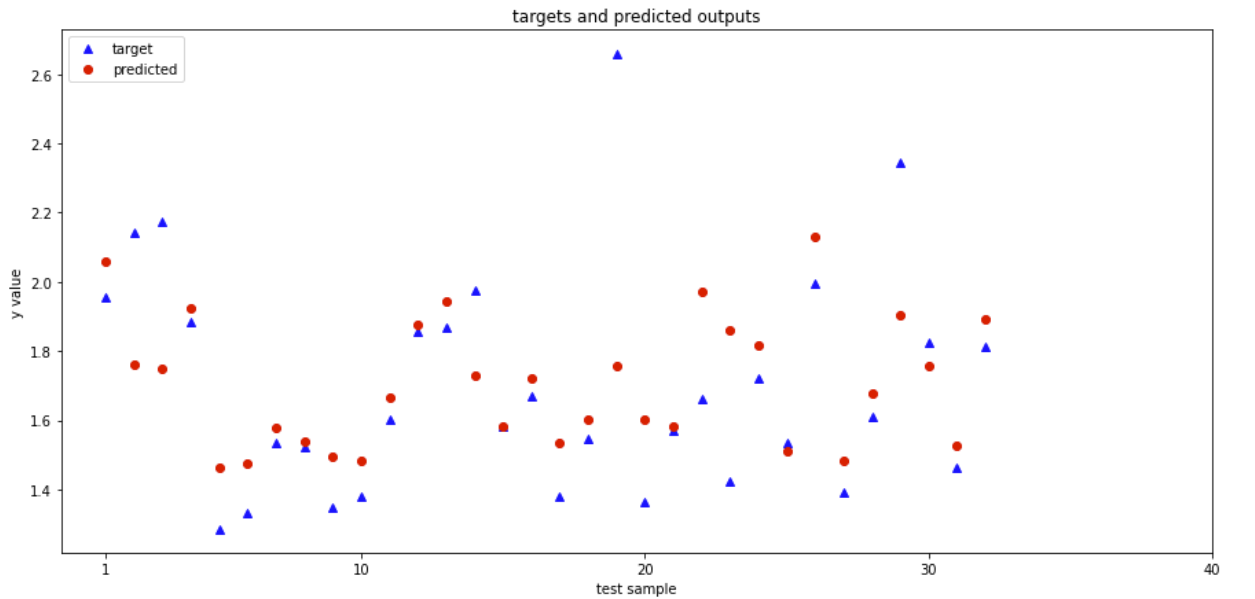
Below shows the plot of M.S.E against the number of training epochs for the model for the reduced dataset:



Similar to the model that used the full dataset, the M.S.E of the model converges to about 0.06 for the last 100 epochs. However, the M.S.E are in the lower 0.06 values which shows that the model does improve slightly with a reduced dataset.



The percentage of predictions within acceptable range did not increase even with the slightly better M.S.E values achieved by the model. Below shows the plot of the predicted fastest lap time values and the actual fastest lap time values for the test set obtained from the dataset that we used.



#### Conclusions

From the results, we can observe that the regression model is not able to obtain the best accuracy in predicting the fastest lap times for each of the circuits for each year. This can be partially attributed to the fact that we do not have enough data to train and validate the regression model on.

Also, the input features that we used for the prediction of the fastest lap times may not be the best features for the predictor to learn a pattern for fastest lap time. This is due to the fact that in reality, many other factors affect the fastest lap time set by the drivers during the race, these include the type of tyres used on the car, amount of fuel left in the car when attempting the fastest lap amongst other things and we do not have access to this information. Therefore, we are unable to improve the loss or the accuracy of the regression model for the prediction of fastest lap times.

### 

### Problem 3 - Predict the Final Drivers’ Standings for Drivers if they are Driving with a Particular Constructor

### Data Integration

* For these questions, we will be reusing the new data frame we created in Problem 1 with the added new features.

### Data Cleaning

We removed the following attributes as we deemed them to be insignificant to our model:

* error\_catagory, name\_y, country, fullName, resultId

### Data Splitting

We split the data frame by grouping the items by the ‘driverId’. With this GroupDataFrame, we created a train and test set with the following conditions:

* For each item in a group:

1. If a driver only drives 1 Construction Throughout 2004 to 2017, append the whole group into the train set.
2. If a driver drives multiple Constructor Throughout 2004 to 2017:
   1. If it is not the last item:
      1. Append item into train set
   2. If it is the last item:
      1. Append item into test set

After splitting, we exported the train and test set to train\_q3.csv and test\_q3.csv respectively.

### Data Transformation

* We group both Test and Train set by ‘driverId’, ‘constructorId’ and ‘year’ and we aggregate the rows by using mean.
* We normalized the data by changing the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values.

### Result and Analysis

For this problem, we first used the train\_q3.csv as the training dataset and test\_q3.csv as the test dataset which contains the following features:

* Driver ID, Constructor ID, Mean Starting Grid Position, Mean Position Order the Driver finishes each race, Driver Error Rate, Constructor Error Rate, Points obtained that Season, Position in the Final Drivers’ Standings , Number of Wins (1st place) in that season

For this task, the objective is to train a prediction model that can predict the Final Drivers’ Standing for a season for a given driver when they drive with a given Constructor. Thus, we decided to build a regression model where the output value will be a real number which will be then rounded off to a whole number to give us the final predicted Driver Standing for a given driver driving for a particular constructor.

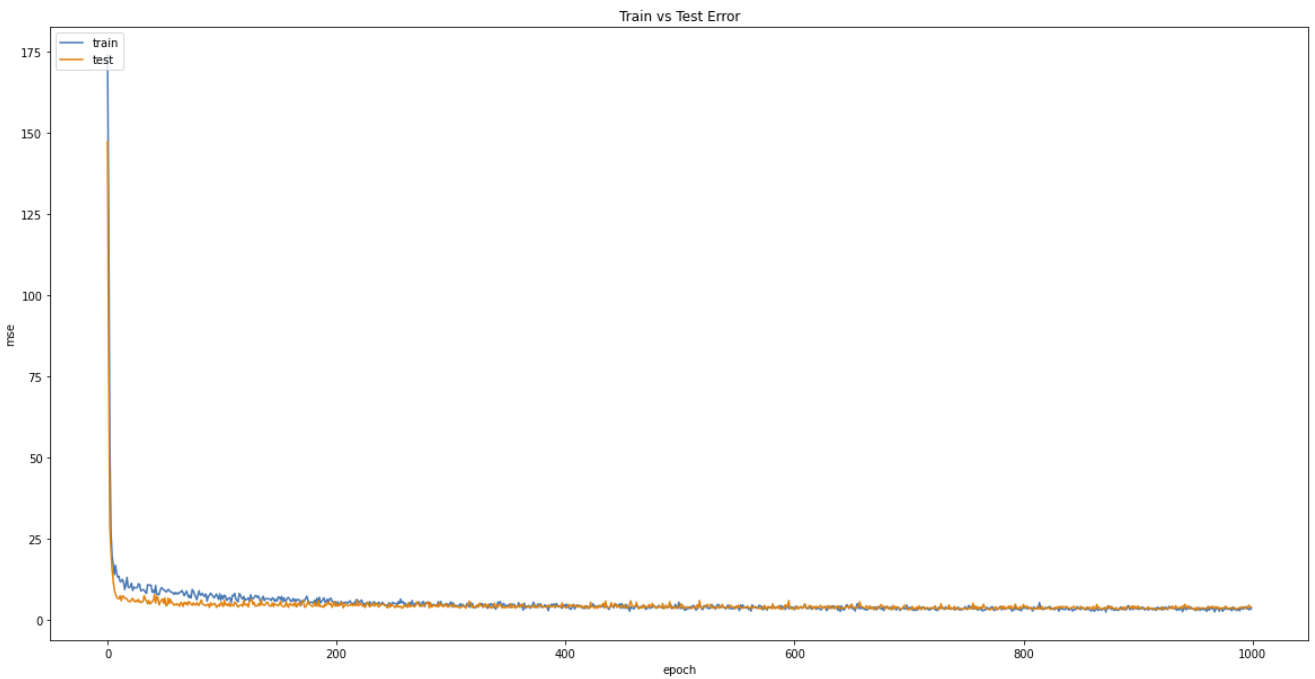
For this regression problem, we utilized a 4-layered Neural Network to perform the regression. The neural network consists of:

* Input layer
* Dense Layer of 32 neurons
* Dropout Layer of 0.2 Dropout Rate
* Dense Layer of 32 neurons
* Dropout Layer of 0.2 Dropout Rate
* Output Layer

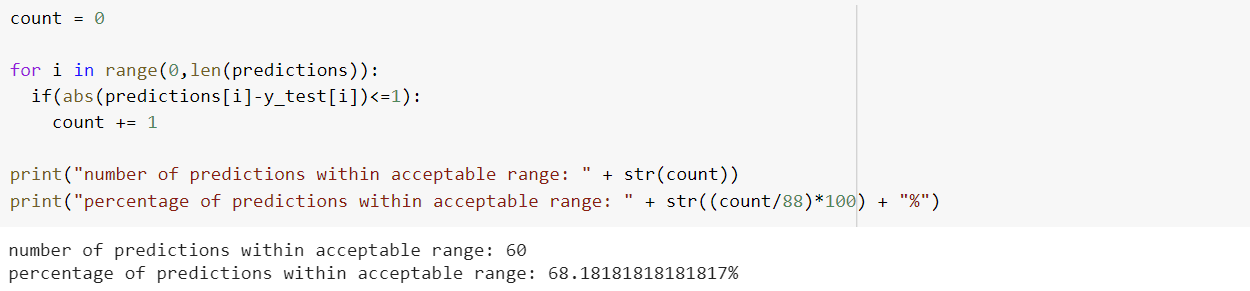
We then used the Adam optimizer for the neural network with a learning rate of 0.001 and a learning decay rate of 1e-6. The regression model is then trained for 1000 epochs.

### Results and Analysis

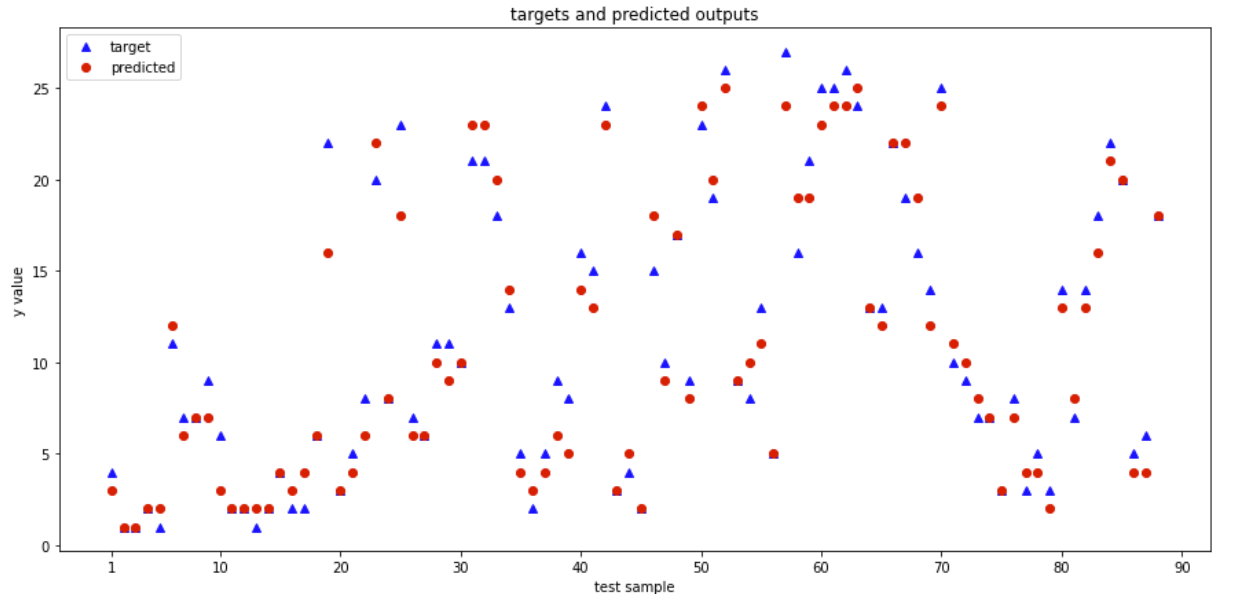
As this is a regression problem, we monitor the Mean Squared Error (MSE) of the regression model and the model is trained to give us the lowest M.S.E possible which reduces the error between the predicted Driver’s Standing and the actual Driver’s Standing. Below shows the plot of M.S.E against the number of training epochs for the model:



We then rounded the predicted Driver’s Standings to whole numbers and compared them against the actual Driver’s Standings. We then calculated the percentages of predictions that are within 1 position difference between the predicted Driver’s Standings and actual Driver’s Standings. The results are shown below:



From the results, we can see that 68 percent of the predictions are within 1 position difference between the predicted Driver’s Standings and actual Driver’s Standings. The graph below shows the plot for the predicted Driver’s Standings and the actual Driver’s Standing. We can see that there are quite a handful of predictions where the predictions are the same as the actual Driver’s Standings as shown by the red dots covering the blue triangles. Also, 88.63% of the predictions are within 2. position difference between the predicted Driver’s Standings and actual Driver’s Standings



#### Conclusions

From the results, we can observe that the regression model is not very accurate in predicting the final Driver Standing for a given driver driving for a particular constructor. This can be partially attributed to the fact that we do not have enough data to train and validate the regression model on.

In addition, the input features that we used for the prediction of the Driver Standing may not be the best features for the predictor to learn a pattern for Driver Standing for a given driver and given constructor. This is due to the fact that in reality, many other factors affect the final Driver’s standing as other teams may develop better cars in the season, the car developed by the given Constructoris not as good as the previous seasons, and many other contributing factors. Therefore, we are unable to improve the loss or the accuracy of the regression model for the prediction of Driver Standing for a given driver and given constructor.

## Problem 4 - Multiclass Classification of Formula One Races

For this problem, we wanted to see if other information regarding the races can help improve race classification accuracy. Therefore, we decided to include another dataset from Kaggle regarding F1. Also, after obtaining the results from problem 1, we wanted to understand the reason we were only able to achieve a maximum of 77.36% accuracy for the prediction of whether a driver will be in a PWP, with that in mind, we decided to extend Problem One into a multilabel classification. The new labels for classification are:

* 0 : DNF (Did Not Finish)
* 1 : Podium (1st to 3rd Positions)
* 2: Point Winning Positions (4th to 10th)
* 3: Finished (11th and above race finishes)

### Data Used

* dataset.csv, races\_cleaned.csv, drivers\_cleaned.csv, results\_clean.csv, combined - final.csv,
* For the new dataset from Kaggle, we have the following new race information:
* statusId : 1 is Finished, 0 is Did Not Finish
* Did not finish, Podium, Pos 4 to 10: probability of a driver in each of the race classification based on past 3 years’ race results for that circuit
* Sc: Probability of a Safety Car
* Wet: 1 is wet weather forecast, 0 is dry weather forecast
* pitStop timing (avg): average pitstop timing of driver
* pitStop timing prop(driver): proportion of pitstop time as compared to race time

### Step 1

We imported the new dataset from Kaggle dataset.csv as main\_df and performed data cleaning to remove all NaN values.

### Step 2

We imported races\_cleaned.csv, drivers\_cleaned.csv and results\_clean.csv as **races\_df** and **drivers** and **results** dataframes. We then performed data Integration to get a dataframe that consisting of data with the various race information, driverId raceId, circuitId as well as their finishing positions in a race.

* First, Merge Left **main\_df** and **races\_df** on “name”, “year” as **df\_combined**.
* Second, Merge Left **df\_combined** and **drivers** on “driverRef”.
* Third, Merge Left **df\_combined** and **results** on “raceId”, “driverId”.

### Step 3

Next, we loop through the rows in the **df\_combined** to search through the positionOrder column and statusId for the positions the drivers finished and whether they finished the race, to create the class labels: 0 : DNF (Did Not Finish), 1 : Podium (1st to 3rd Positions) ,2: Point Winning Positions (4th to 10th), 3: Finished (11th and above race finishes)

### Step 4

We import combined - final.csv as **combined** that consist of the driver error rates, constructor error rates and circuit error rates columns that we have created for Problem 1 as we know that they are important input features for race predictions from the sklearn **selectkbest** feature reduction.

* First, we drop all other column in combined - final.csv except ‘driver\_error\_rate’, ‘constructor\_error\_rate’ and ‘circuit\_error\_rate’, as well as 'raceId','driverId','circuitId' in order to merge with **df\_combined**
* Second, Merge Left **df\_combined** and **combined** on “raceId”, “driverId” and 'circuitId'.

### Step 5

Next, we drop the columns in the dataframe **df\_combined** that we do not need as inputs for the classification. They are: 'raceId', 'year','name','driverRef','positionOrder','statusId','Super Soft','Ultra soft','Hard', 'Medium','Soft'.

### Step 6

Finally, after all the previous steps were performed, we exported this dataset into a new csv, question4.csv’

For this task, the objective is to train a prediction model that can predict the Final Drivers’ Standing for a season for a given driver when they drive with a given Constructor. Thus, we decided to build a regression model where the output value will be a real number which will be then rounded off to a whole number to give us the final predicted Driver Standing for a given driver driving for a particular constructor.

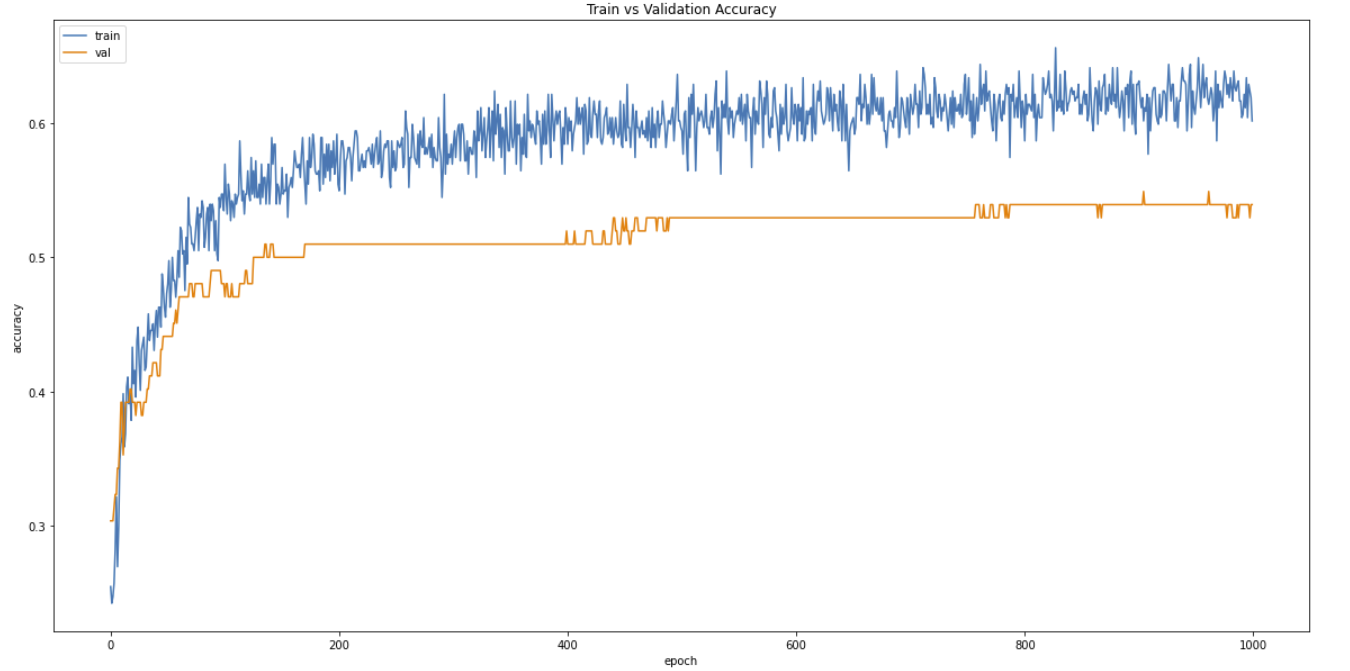
### Implementation of Classification Model

For this classification problem, we utilized a 4-layered Neural Network to perform the classification. The neural network consists of:

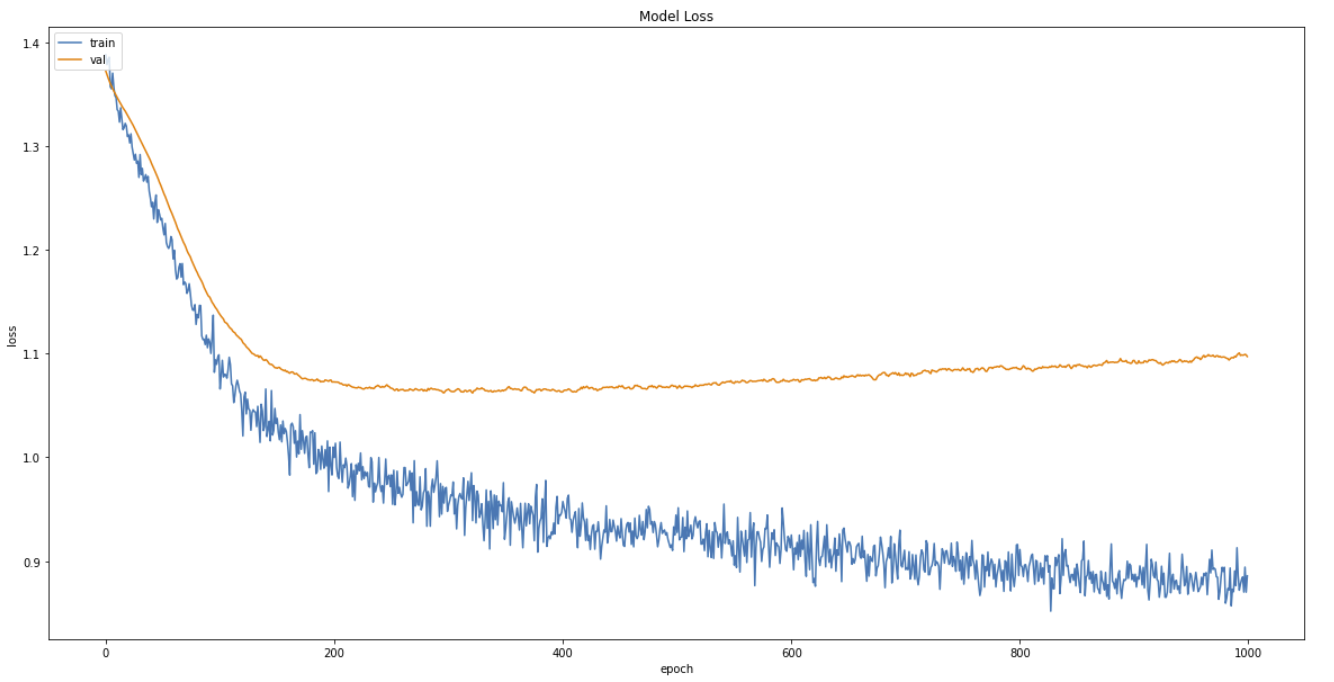
* Input layer
* Dense Layer of 32 neurons
* Dropout Layer of 0.2 Dropout Rate
* Dense Layer of 32 neurons
* Dropout Layer of 0.2 Dropout Rate
* Dense Output Layer of 4 neurons with “softmax” activation

We then used the SGD optimizer for the neural network with a learning rate of 0.001 and a learning decay rate of 1e-6. The classification model is then trained for 1000 epochs.

### Results and Analysis



**Train and validation Accuracy vs. Epochs**



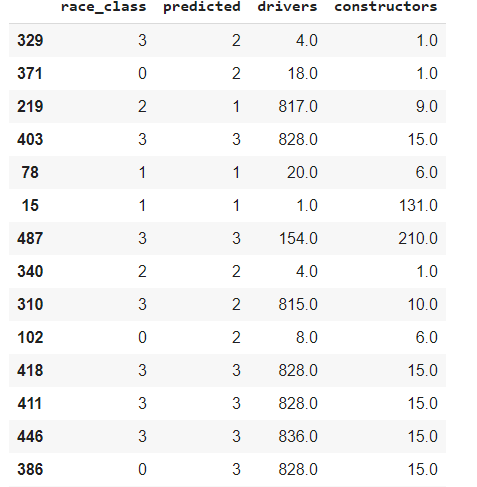
**Train and validation Loss vs. Epochs**

From the results above, we can see that the test accuracy obtained by the model is not very high, with the test accuracy converging at about 53.9% accuracy.

#### Further Analysis of Results obtained

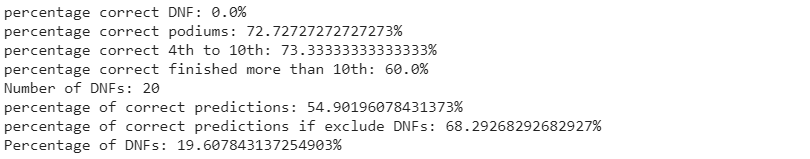
After observing that we are getting unsatisfactory results for our classification model, we went on to do further analysis on the results obtained.

First, we create a list of classes predicted by the model by using the **keras** model.predict() function using the input data from the test set. We then also get the target values, the driver and constructor we are predicting the classes for, and put them together into a dataframe.



**Head of Dataframe**

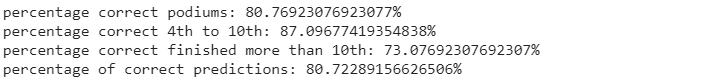
Next, for each of the classes, we loop through the dataframe to find out the percentage of correct predictions for each of the classes by comparing the predicted classes with the actual target classes. These are the results for each of the classes:



From the results, we can see that for the DNF class, we are getting 0% correct predictions, while for the other 3 classes, we are getting decent correct prediction accuracies with the accuracy of the 3 classes higher than that of the overall test accuracy of the model.

Not only that, we found out that DNFs took up 19% of the test set and if we do not consider the DNF rows in the test set, we are able to achieve an overall test accuracy of up to 68.29% for the other 3 classes.

This explains the poor performance of the model in predicting the race classes. If we were to take out predicting DNFs, and only predict 3 classes, and consider data that do not have DNF, these are the results obtained:



## 6 Conclusions

### 6.1 Summary of Project Achievements

### 6.2 Directions for Improvement

# 

# 

# 8 Appendix

## **8.1** **Additional Information**

#### Cross Validation Accuracies Comparison of All Datasets

|  |  |  |  |
| --- | --- | --- | --- |
|  | Initial Dataset | Dataset with Added Features | Reduced Dataset |
| SVM | 74.092% | 74.112% | 74.741% |
| Naives Bayes | 71.706% | 70.730% | 71.236% |
| Decision Tree | 66.150% | 66.644% | 71.244% |
| Random Forest | 71.549% | 71.806% | 72.533% |

#### F1 Score Comparison of All Datasets

|  |  |  |  |
| --- | --- | --- | --- |
|  | Initial Dataset | Dataset with Added Features | Reduced Dataset |
| SVM | 0.75936 | 0.76514 | 0.76776 |
| Naives Bayes | 0.74948 | 0.76447 | 0.76892 |
| Decision Tree | 0.68831 | 0.68207 | 0.73362 |
| Random Forest | 0.72747 | 0.73171 | 0.75210 |

#### ROC AUC Score Comparison of All Datasets

|  |  |  |  |
| --- | --- | --- | --- |
|  | Initial Dataset | Dataset with Added Features | Reduced Dataset |
| SVM | 0.75617 | 0.76045 | 0.76259 |
| Naives Bayes | 0.73968 | 0.74355 | 0.74786 |
| Decision Tree | 0.68803 | 0.68710 | 0.73579 |
| Random Forest | 0.72142 | 0.72575 | 0.75611 |

## **8.2** **Implementation Guidelines**