UFC Historical Data Analysis

Zafar Iqbal, Sean Staley, Yue Zhao, Vincent Hu

Our dataset and the type of data it contains:

The Ultimate Fighting Championship (UFC) is an American mixed martial arts promotion company based in Las Vegas. The UFC produces events worldwide that showcase twelve weight divisions and abides by the Unified Rules of Mixed Martial Arts. As of 2019, the UFC has held over 500 events. The UFC produces events worldwide that showcase twelve weight divisions. The first event was held in 1993.

Dataset:
UFC-Fight historical data from 1993 to 2019

This data set has been scraped from the UFC stats website. After some basic preprocessing with pandas, the dataset was made available publicly on the kaggle machine learning repository. The following link can be used to download the dataset directly (https://www.kaggle.com/rajeevw/ufcdata).

This dataset contains information about every fight in the history of the organization. Each row contains information about both fighters, fight details and the winner. The original data was not one-hot encoded or processed for missing values. It had a total of 5,144 instances, 145 attributes and 95,390 missing values.

Fighters are represented by 'red' and 'blue' (for red and blue corner). For instance, data on the red fighter has the complied average stats of all the fights before the current one. The stats include damage done by the red fighter to the opponent and the damage done by the opponent on the fighter in all the fights this particular red fighter had except the current one (in the data). The same information exists for blue fighter. The target variable is 'Winner', which is the only column that tells you what happened in the current fight. It has three unique values i.e. "Red", "Blue" and "Draw". Red and Blue represent the winning team and draw represents a match drawn between both Red and Blue teams. The number of instances when Red team won are 3,470 and for the Blue team are 1,591. The number of 'Draw' matches was a mere 83, which implies an imbalanced dataset.

The original dataset had 108 continuous and 4 nominal attributes, and the remaining 33 were categorical attributes.

Column definitions:
R_ and B_ prefix signifies red and blue corner fighter stats respectively
opp containing columns is the average of damage done by the opponent on the fighter
KD is number of knockdowns
SIG_STR is no. of significant strikes 'landed of attempted'
SIG_STR_pct is significant strikes percentage
TOTAL_STR is total strikes 'landed of attempted'
TD is no. of takedowns
TD_pct is takedown percentages
SUB_ATT is no. of submission attempts

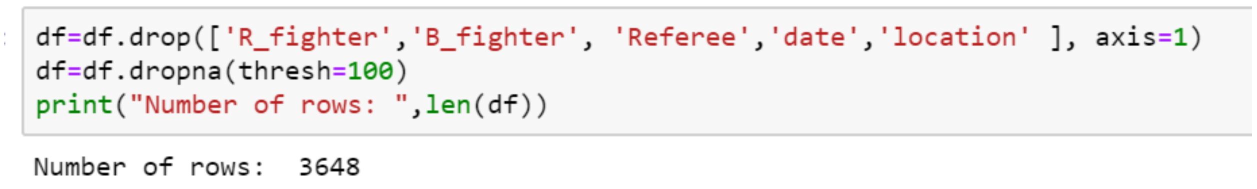
PASS is no. times the guard was passed.
REV is the no. of Reversals landed
HEAD is no. of significant strinks to the head 'landed of attempted'
BODY is no. of significant strikes to the body 'landed of attempted'
CLINCH is no. of significant strikes in the clinch 'landed of attempted'
GROUND is no. of significant strikes on the ground 'landed of attempted'
win_by is method of win
last_round is last round of the fight (ex. if it was a KO in 1st, then this will be 1)
last_round_time is when the fight ended in the last round
Format is the format of the fight (3 rounds, 5 rounds etc.)
Referee is the name of the Ref
date is the date of the fight
location is the location in which the event took place
Fight_type is which weight class and whether it's a title bout or not
Winner is the winner of the fight
Stance is the stance of the fighter (orthodox, southpaw, etc.)
Height_cms is the height in centimeter
Reach_cms is the reach of the fighter (arm span) in centimeter
Weight_lbs is the weight of the fighter in pounds
age is the age of the fighter
title_bout Boolean value of whether it is title fight or not
weight_class is which weight class the fight is in (Bantamweight, heavyweight, Women's flyweight, etc.)
no_of_rounds is the number of rounds the fight was scheduled for
current_lose_streak is the count of current concurrent losses of the fighter
current_win_streak is the count of current concurrent wins of the fighter
draw is the number of draws in the fighter's ufc career
wins is the number of wins in the fighter's ufc career
losses is the number of losses in the fighter's ufc career
total_rounds_fought is the average of total rounds fought by the fighter
total_time_fought(seconds) is the count of total time spent fighting in seconds
total_title_bouts is the total number of title bouts taken part in by the fighter
win_by_Decision_Majority is the number of wins by majority judges' decision in the fighter's ufc career
win_by_Decision_Split is the number of wins by split judges' decision in the fighter's ufc career
win_by_Decision_Unanimous is the number of wins by unanimous judges' decision in the fighter's ufc career
win_by_KO/TKO is the number of wins by knockout in the fighter's ufc career
win_by_Submission is the number of wins by submission in the fighter's ufc career
win_by_TKO_Doctor_Stoppage is the number of wins by doctor stoppage in the fighter's ufc career

Preprocessing steps and descriptive statistics:

As the number of attributes was very large, the preference was to drop irrelevant features from the dataset. We did this because leaving a very high dimensionality (number of features), would increase the computational cost and the model might over fit to training data with too many features. With overfitting, the model would not be able to perform well on test or unseen data.

Four nominal features ('R_fighter', 'B_fighter', 'Referee', 'location') are dropped in the first step because they should not have a significant impact on the prediction model.

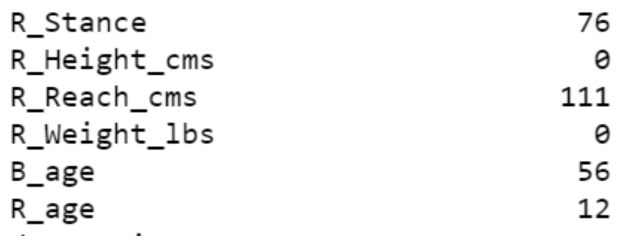
The next step was to deal with missing values. As mentioned earlier, there was a whopping 95,390 missing values in the dataset, which needed to be dealt with. Therefore rows containing more than 45 NaN values out of a total 144 values were dropped. The rational behind this was that since these rows contain so little information, attempting to fill them in with any technique could adversely impact the prediction capabilities of a model on test data. As shown in figure below, the number of records dropped to 3,648.



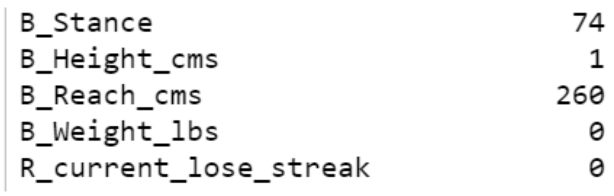
```
df=df.drop(['R_fighter','B_fighter', 'Referee','date','location' ], axis=1)
df=df.dropna(thresh=100)
print("Number of rows: ",len(df))
```

```
Number of rows:  3648
```

There are still missing values in the dataset. The figure below shows a list of attributes with missing values.



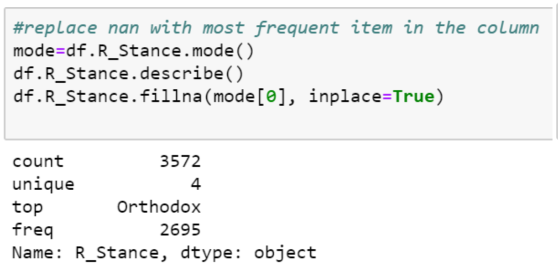
```
R_Stance                  76
R_Height_cms               0
R_Reach_cms              111
R_Weight_lbs               0
B_age                     56
R_age                     12
```



```
B_Stance                  74
B_Height_cms               1
B_Reach_cms              260
B_Weight_lbs               0
R_current_lose_streak      0
```

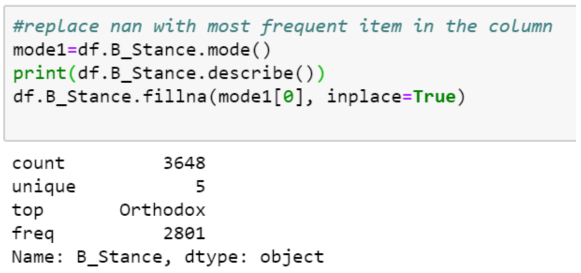
By carefully examining these four attributes i.e. (R_age, B_age, R_Reach_cms, B_Reach_cms) through histograms and other visual statistics, we choose to replace the missing values using linear regression, as we noticed a linear relationship among certain features (identified from correlation matrix). We also considered KNN but that is not suitable for high dimensional data. To fill in missing values, we identified the four most correlated features for each missing value and then applied linear regression to replace the NaN values with the predicted values.

For discrete value attributes (i.e.: R_Stance and B_stance), the mode is used to fill in the empty cells. The rationale being: because the most frequent item in the relevant columns had a very high frequency (2695 out of 3648), the mode seemed an appropriate choice. The statistical description of both the features are shown in figure below. After replacing NaN values with mode, the number of missing values in the dataset becomes zero.



```
#replace nan with most frequent item in the column
mode=df.R_Stance.mode()
df.R_Stance.describe()
df.R_Stance.fillna(mode[0], inplace=True)
```

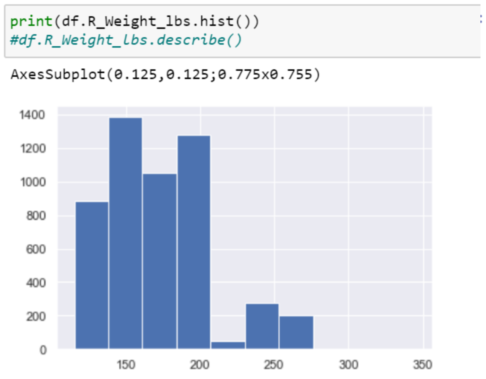
```
count          3572
unique            4
top        Orthodox
freq           2695
Name: R_Stance, dtype: object
```



```
#replace nan with most frequent item in the column
mode1=df.B_Stance.mode()
print(df.B_Stance.describe())
df.B_Stance.fillna(mode1[0], inplace=True)
```

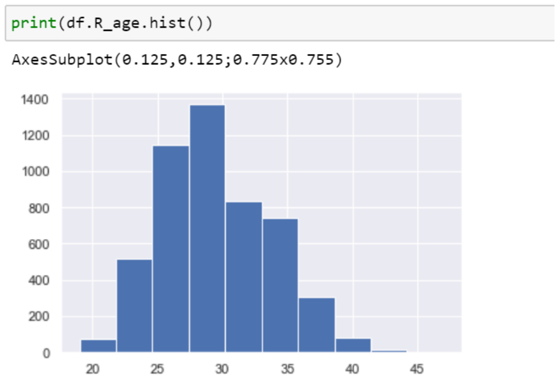
```
count          3648
unique            5
top        Orthodox
freq           2801
Name: B_Stance, dtype: object
```

We searched for anomalies in the dataset. Anomalies are outliers that could be caused by error or something else. For example, if there was a value of 200 in the age column, it could not be true, as the age of a human can't be greater then 120 and a players' age could not be more be 50. We did historgram analysis on all the features and found no anomalies in the data set. The figure below shows the value distribution of two features.



```
print(df.R_Weight_lbs.hist())
#df.R_Weight_lbs.describe()
```

```
AxesSubplot(0.125,0.125;0.775x0.755)
```

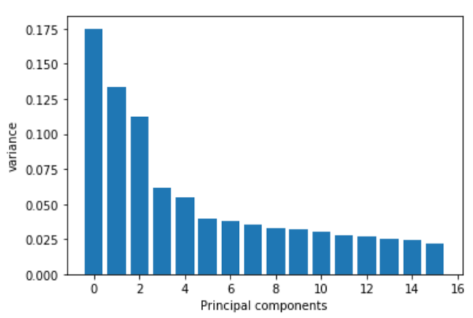


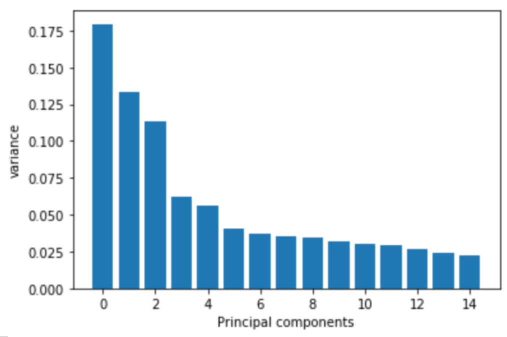
```
print(df.R_age.hist())
```

```
AxesSubplot(0.125,0.125;0.775x0.755)
```

Principle Component analysis:

After we finished dealing with missing values, the next step was to reduce the number of features while losing as little information as possible. There were 108 continuous values, 72 related to one team and the other half related to the opponent team. The first step was to standardize all the values. Then using a correlation matrix, we dropped features that had a correlation greater then 80%. The reason is that the correlation between features should be kept to a minimum to avoid colinearity. Afterwards, we did PC analysis separately on the 1st half of continuous value features related to the Red corner and retained PC's that contained a minimum of 85% of the information. We applied the same methods to the other half of the attributes. It can be observed in the following figures that the number of attributes decreased drastically from 108 to 31.

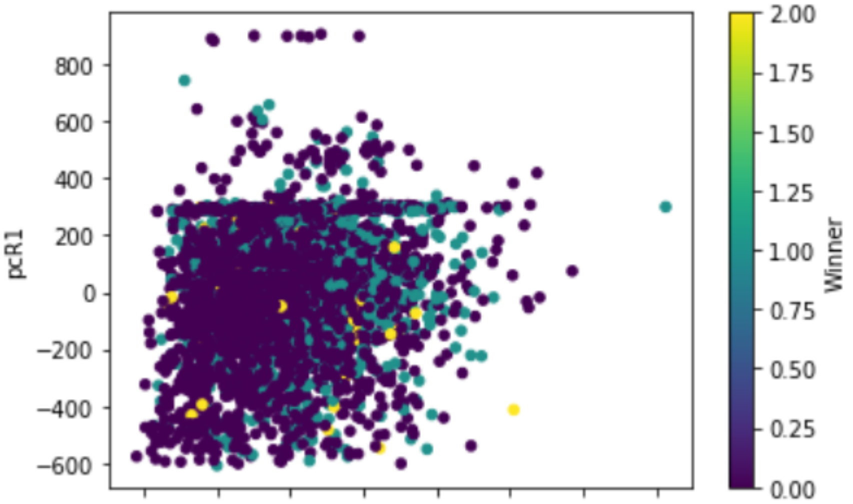




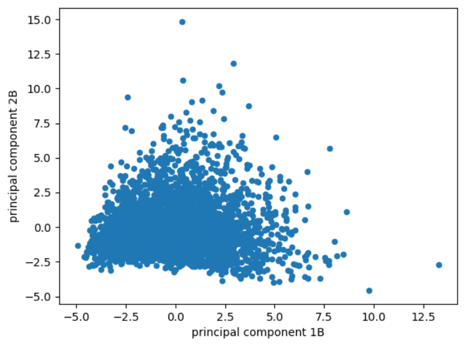
Characteristics of the processed data:

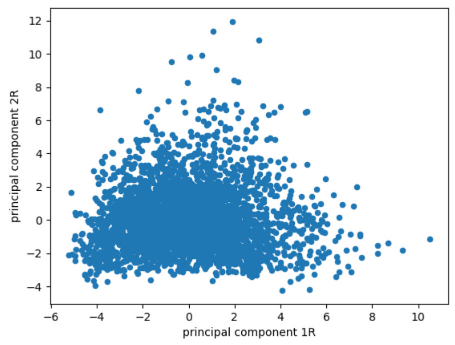
Columns: 65
Rows: 3647
Standardized values
No missing values
Low dimensional Data
No anomalies

The following figure depicts a scatter plot between PC1 for the Red corner and PC1 for the Blue corner team. The class information has also been added to it. It could be seen from the figure that the instances when red team won the match are in majority. There are only a few instances when there was a match draw (represented a yellow dots).

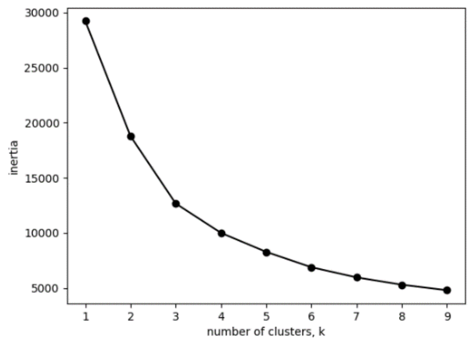


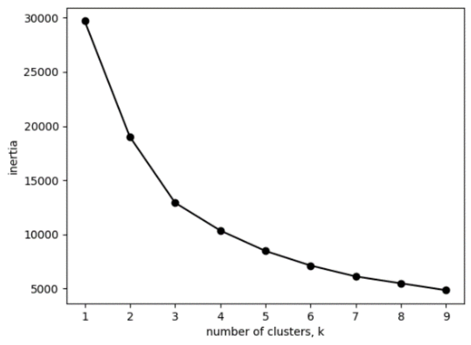
Before we start our clustering algorithms, we will view the first two principal components for the red and blue fighters plotted together in a scatter plot. This is the data we will be running our clustering algorithms on to see if there are any clearly defined groups in our fighter dataset. If we can group fighters into distinct clusters based on their attributes it will be easier to interpret and predict the fight results.



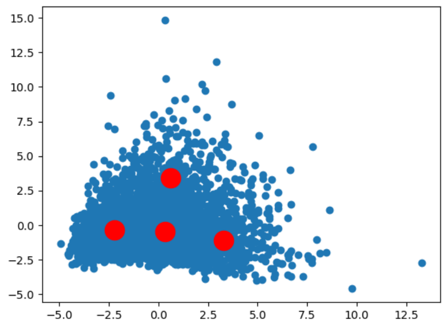
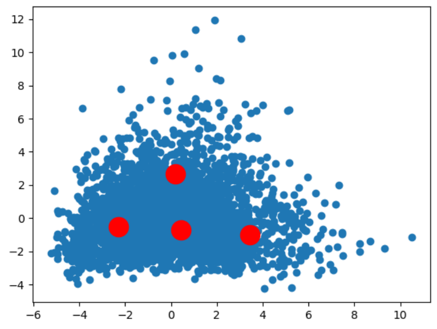


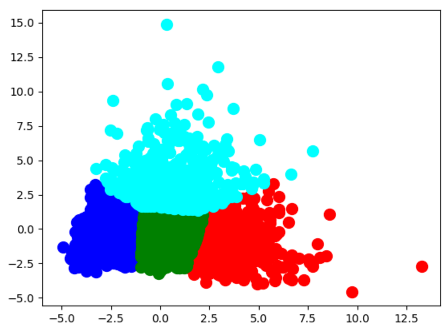
From these graphs it is not evident if there are any distinct clusters in the data. A cluster refers to data that is grouped together due to similarities. From just looking at these graphs, we are not likely to get much meaningful information from our clustering algorithms since there appear to be no aggregated clusters in our data, nevertheless we will run the algorithms and see what we get.
Since it is not clear how many clusters we should define for this data, and K-Means clustering requires us to set the cluster values before hand we will use the "Elbow Method" to find a good value for k. The elbow method runs the algorithm multiple times in a loop with increasing numbers of clusters and then plots the K-Means score as a function of the number of clusters. It does this by finding the inertia or the measured sum of the squared distances to the nearest cluster center. The results for both red and blue fighters are shown below.

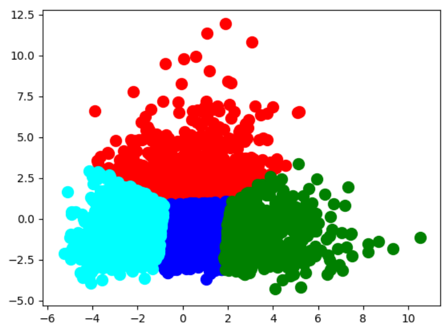




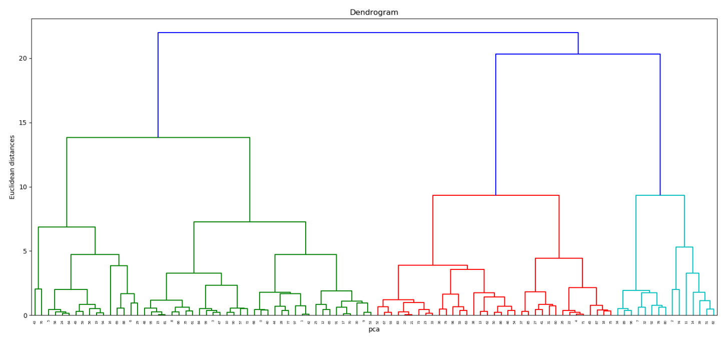
From the inertia graphs it looks like around 4 our results start to drop off, so this is a good value for k. Next is to run our algorithm and plot the results. K-Means works by finding a k number of centroids and allocates points around it to the nearest cluster. Below are the resulting centroids and clusters. The K-Means results for the blue fighter is shown below on the left, and the red fighter on the right. The algorithm was able to find 4 clusters in the data that we would not have noticed right away by visualizing.

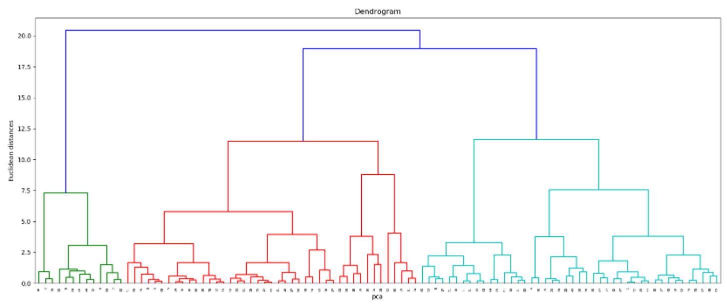





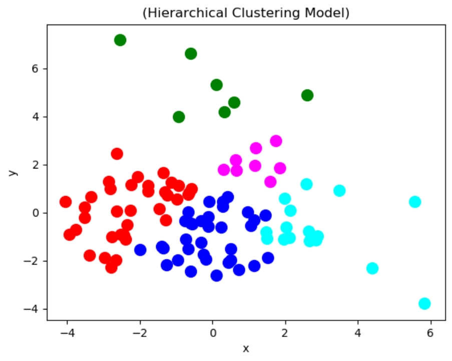


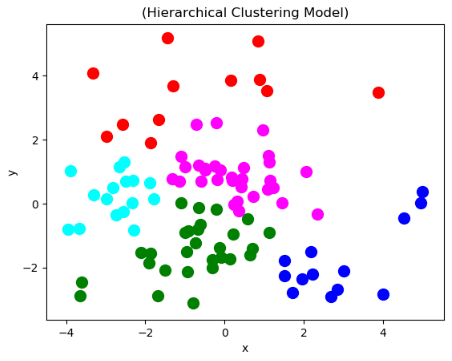
The next clustering algorithm we ran on the data is called Hierarchical Clustering. There are two types of Hierarchical clustering, agglomerative and divisive. Agglomerative is basically a bottom up approach where we start with small clusters and work our way up to a larger cluster, and divisive works in reverse, starting with one cluster and breaking it up into smaller ones. One positive of this method is that we do not need to know the number of clusters in advance before running it such as in our K-Means example. A downside however is that it will run very slow on larger sets of data. Because of this we have randomly sampled 100 from our 3000+ columns to use. Hierarchical Clustering works by making each datapoint a cluster and then making the points around it a cluster until it is left with only one cluster of data. We can visualize this happening in a dendrogram. I have plotted dendrograms for the red and blue fighter data below.





These look complicated but they are in fact very simple to understand. Each connection between two points on the graph resembles a correlation. The lower the connection the higher the correlation and the higher up the connection the lower the correlation.
The dendrogram is used to visualize the history of the groupings and determine how many clusters are needed. For this dataset, Hierarchical Clustering determined that 5 clusters were enough. With less data points that the K-Means example, it is easier to see how these clusters make sense.





So, how to interpret the results? from both clustering algorithms it seems like our dataset is just not that suitable to extract much meaningful information from clustering algorithms, and this makes sense. If we were able to clearly identify a group of fighters and whether they would win based on a few attributes it would make things much less interesting, and people would most likely not watch the fight because the winner would be clearly defined beforehand based on what specific cluster he/ she and his/ hers opponent were in.

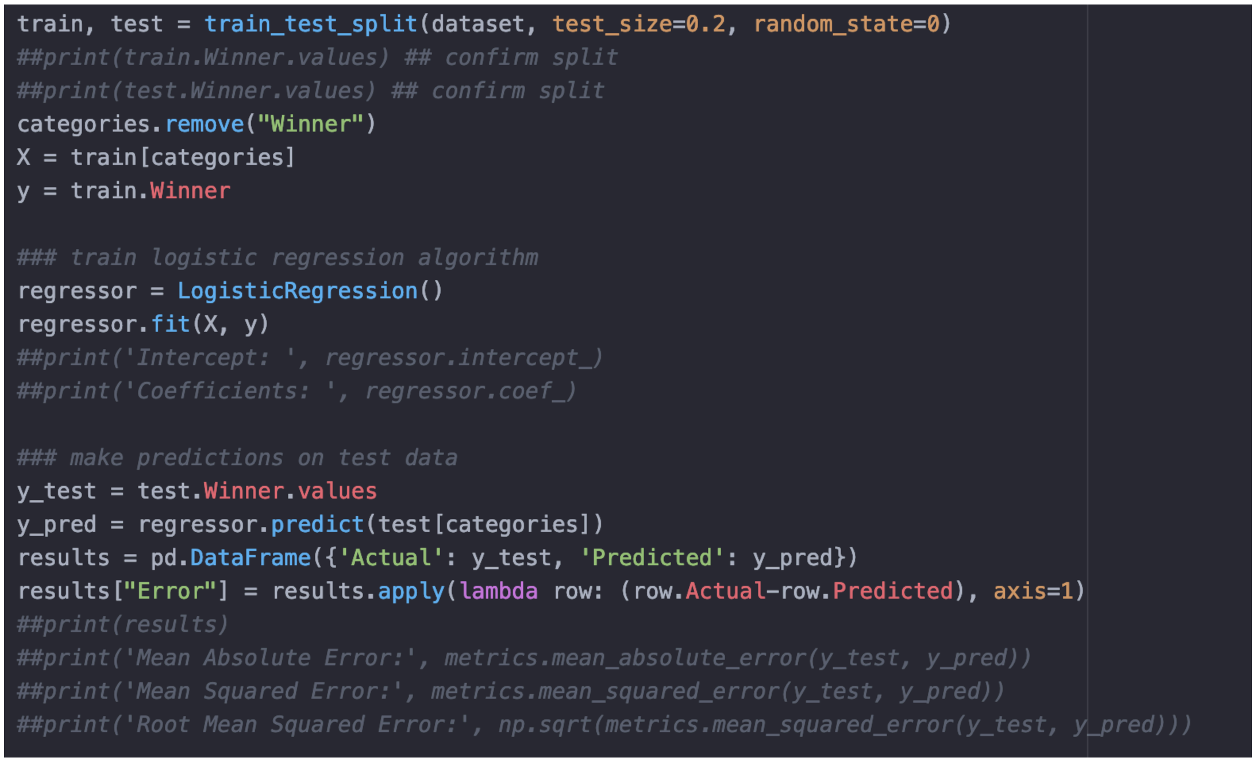
Categorical Prediction:

For categorical prediction, we used logistic regression. We took the preprocessed data and looked for correlation to the winner amongst the categories. Because almost all of these categories had a very weak correlations, we did not expect our model to be a great predictor of outcome, nonetheless, predicting the outcome of the fight still remains to be the most interesting use of the data, so that remained our goal for this algorithm.

categories	**correlation to winner**
R_Weight_lbs	0.080
B_total_title_bouts,	0.076
B_Weight_lbs,	0.072
R_current_win_streak,	0.061
R_Reach_cms,	0.057
R_Height_cms,	0.055
B_win_by_TKO_Doctor_Stoppage,	0.046
R_total_title_bouts,	0.037
B_Height_cms,	0.032
B_win_by_Decision_Majority,	0.031
B_win_by_Submission,	0.015
B_win_by_KO/TKO,	0.014
B_current_lose_streak,	0.007
R_win_by_TKO_Doctor_Stoppage,	0.004
R_win_by_Submission,	-0.001
R_win_by_Decision_Majority,	-0.005
R_longest_win_streak,	-0.005
B_losses,	-0.006
B_Reach_cms,	-0.010

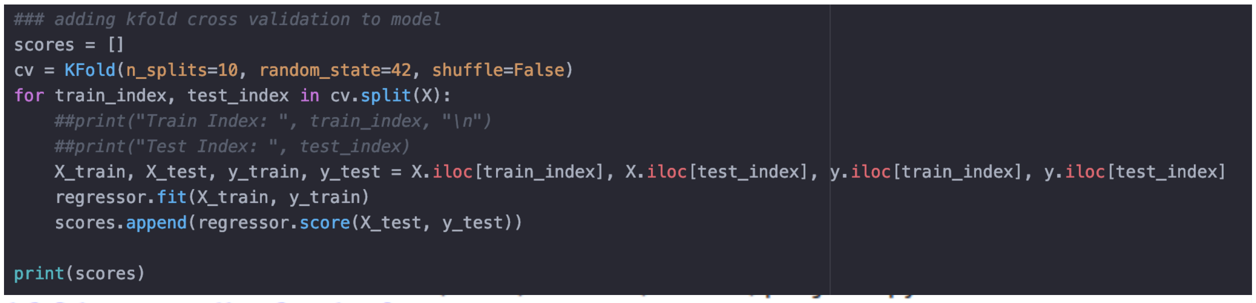
B_win_by_Decision_Split,	-0.013
B_wins,	-0.014
B_longest_win_streak,	-0.015
R_win_by_KO/TKO,	-0.033
B_current_win_streak,	-0.038
R_win_by_Decision_Unanimous,	-0.059
R_wins,	-0.064
R_current_lose_streak,	-0.068
B_win_by_Decision_Unanimous,	-0.072
R_win_by_Decision_Split,	-0.120
R_losses,	-0.154

Logistic regression models the probability of a binary outcome, so we removed draw results from the data since they were immaterial and less interesting. Then we applied the logistic regression and viewed our results.

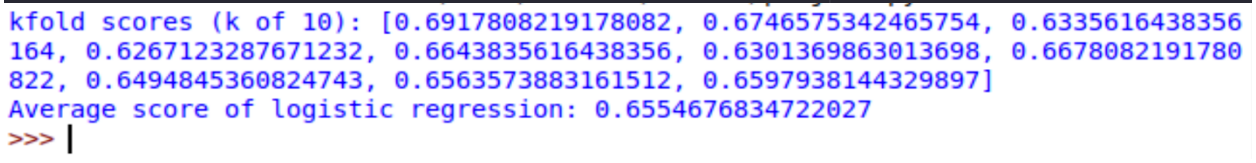


```
train, test = train_test_split(dataset, test_size=0.2, random_state=0)
##print(train.Winner.values) ## confirm split
##print(test.Winner.values) ## confirm split
categories.remove("Winner")
X = train[categories]
y = train.Winner

### train logistic regression algorithm
regressor = LogisticRegression()
regressor.fit(X, y)
##print('Intercept: ', regressor.intercept_)
##print('Coefficients: ', regressor.coef_)

### make predictions on test data
y_test = test.Winner.values
y_pred = regressor.predict(test[categories])
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
results["Error"] = results.apply(lambda row: (row.Actual-row.Predicted), axis=1)
##print(results)
##print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
##print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
##print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

We tried running the algorithm with a number of different inputs but none of the results were very successful. This is to be somewhat expected: if predicting the winner of a fight was easily done based on this kind of statistical data, it would probably ruin the sports gambling industry. From a logical standpoint, this also makes sense. Sports in general are a combination of physical ability, mentality and luck. The statistics in the original data set cannot account for two of these factors. Assuming that luck is the noise that affects any model, the psychological aspect of the fights (and sports in general) is hard to measure in this kind of data.
After running the logistic regression, we applied k-fold cross validation to get the results across a few different cases.



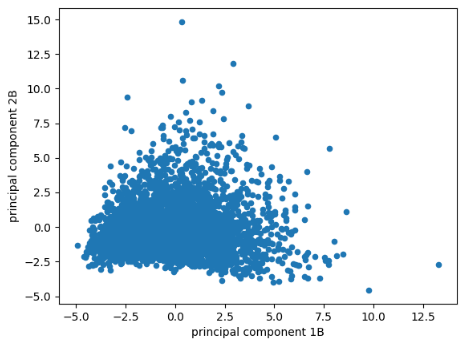
```
### adding kfold cross validation to model
scores = []
cv = KFold(n_splits=10, random_state=42, shuffle=False)
for train_index, test_index in cv.split(X):
    ##print("Train Index: ", train_index, "\n")
    ##print("Test Index: ", test_index)
    X_train, X_test, y_train, y_test = X.iloc[train_index], X.iloc[test_index], y.iloc[train_index], y.iloc[test_index]
    regressor.fit(X_train, y_train)
    scores.append(regressor.score(X_test, y_test))

print(scores)
```

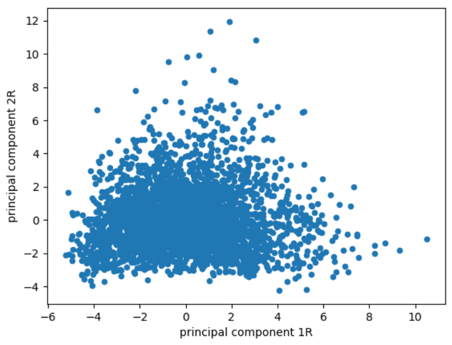


```
kfold scores (k of 10): [0.6917808219178082, 0.6746575342465754, 0.6335616438356
164, 0.6267123287671232, 0.6643835616438356, 0.6301369863013698, 0.6678082191780
822, 0.6494845360824743, 0.6563573883161512, 0.6597938144329897]
Average score of logistic regression: 0.6554676834722027
>>> 
```

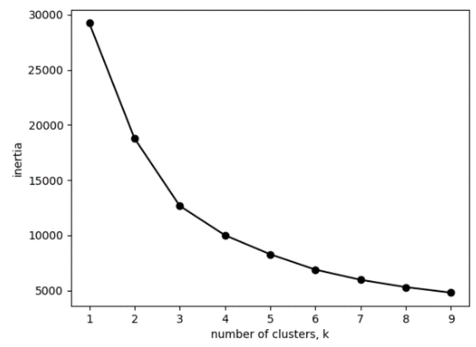
Clustering Algorithm:

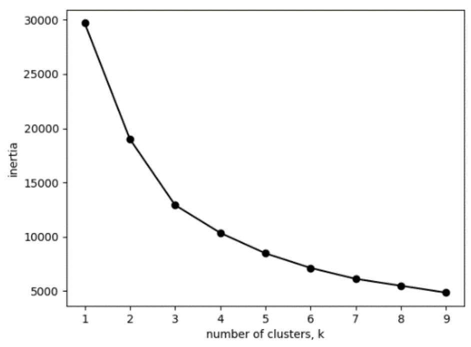
Before we start our clustering algorithms, we will view the first two principal components for the red and blue fighters plotted together in a scatter plot. This is the data we will be running our clustering algorithms on to see if there are any clearly defined groups in our fighter dataset. If we can group fighters into distinct clusters based on their attributes it will be easier to interpret and predict the fight results.



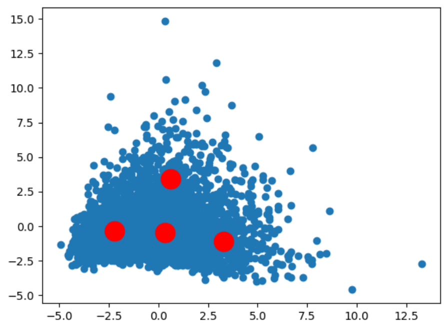
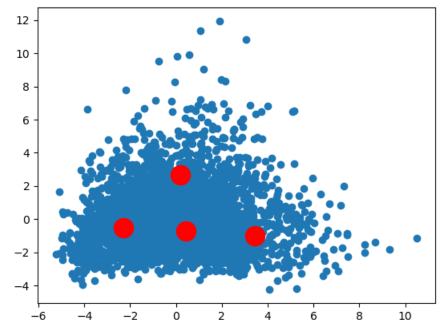
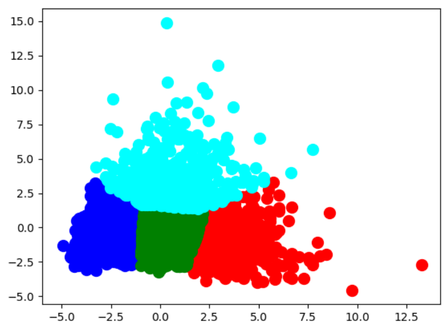
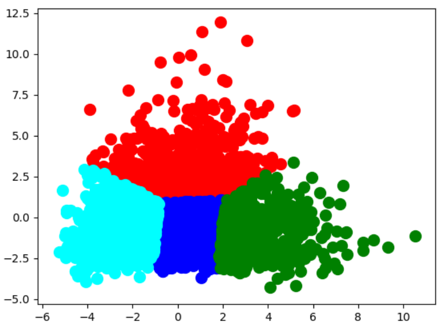


From these graphs it is not evident if there are any distinct clusters in the data. A cluster refers to data that is grouped together due to similarities. From just looking at these graphs, we are not likely to get much meaningful information from our clustering algorithms since there appear to be no aggregated clusters in our data, nevertheless we will run the algorithms and see what we get.
Since it is not clear how many clusters we should define for this data, and K-Means clustering requires us to set the cluster values before hand we will use the “Elbow Method” to find a good value for k. The elbow method runs the algorithm multiple times in a loop with increasing numbers of clusters and then plots the K-Means score as a function of the number of clusters. It does this by finding the inertia or the measured sum of the squared distances to the nearest cluster center. The results for both red and blue fighters are shown below.

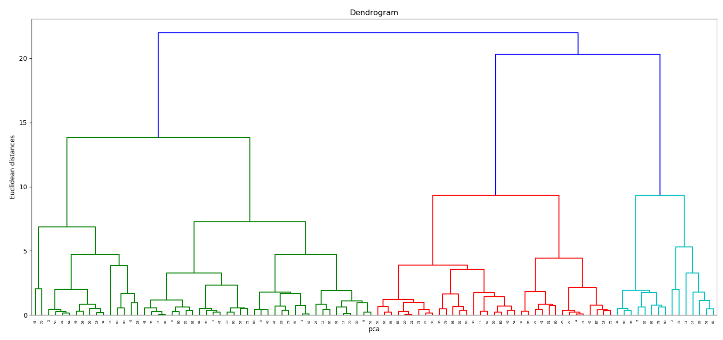


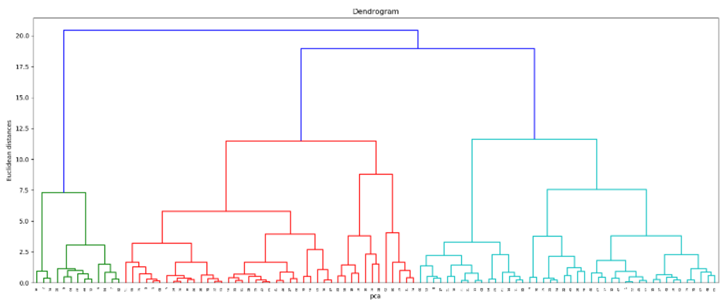


From the inertia graphs it looks like around 4, our results start to drop off, so this is a good value for k. Next is to run our algorithm and plot the results. K-Means works by finding a k number of centroids and allocates points around it to the nearest cluster. Below are the resulting centroids and clusters. The K-Means results for the blue fighter is shown below on the left, and the red fighter on the right. The algorithm was able to find 4 clusters in the data that we would not have noticed right away by visualizing.

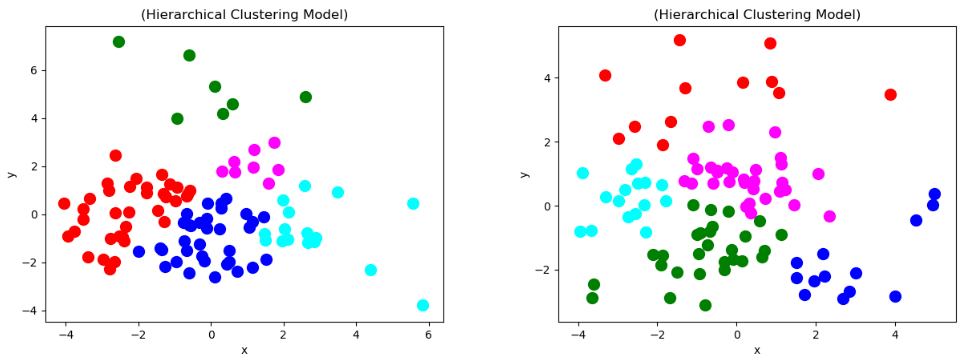





The next clustering algorithm we ran on the data is called Hierarchical Clustering. There are two types of Hierarchical clustering, agglomerative and divisive. Agglomerative is basically a bottom up approach where we start with small clusters and work our way up to a larger cluster, and divisive works in reverse, starting with one cluster and breaking it up into smaller ones. One positive of this method is that we do not need to know the number of clusters in advance before running it such as in our K-Means example. A downside however is that it will run very slow on larger sets of data. Because of this we have randomly sampled 100 from our 3,000+ columns to use. Hierarchical Clustering works by making each datapoint a cluster and then making the points around it a cluster until it is left with only one cluster of data. We can visualize this happening in a dendrogram. I have plotted dendrograms for the red and blue fighter data below.





These look complicated but they are in fact very simple to understand. Each connection between two points on the graph resembles a correlation. The lower the connection the higher the correlation and the higher up the connection the lower the correlation.
The dendrogram is used to visualize the history of the groupings and determine how many clusters are needed. For this dataset, Hierarchical Clustering determined that 5 clusters were enough. With less data points that the K-Means example, it is easier to see how these clusters make sense.



So, how to interpret the results? from both clustering algorithms it seems like our dataset is just not that suitable to extract much meaningful information from clustering algorithms, and this makes sense. If we were able to clearly identify a group of fighters and whether they would win based on a few attributes it would make things much less interesting, and people would most likely not watch the fight because the winner would be clearly defined beforehand based on what specific cluster he/ she and his/ hers opponent were in.

Continuous Value Prediction:

METHODOLOGY

The primary approach of our model, is to take fighters' attributes as input to tag them with Winner or Non-Winner and show the probability. After this tagging process, the accuracy score for each algorithm is calculated. Algorithms are filtered to exclude those with lower accuracy score. In this process, we got accuracy scores as Table.1. Among the algorithms with higher accuracy score including SVM Rbf, SVM Poly, Random Forest and Gradient Boosting Decision Tree, we performed grid search parameter optimization for each.

For SVM Rbf, I optimized parameter c to control the tradeoff between maximum marginal boundaries

Table 1

Algorithms	Accuracy score
SVM linear	0.683333333333330
SVM RBF	0.708333333333330
SVM Sigmoid	0.661111111111110
SVM Poly	0.702777777777780
Random Forest	0.694444444444440
Gradient Boosting Decision Tree	0.694444444444440
KNN	0.652777777777780

Table 2

C	Accuracy score
0.1	0.719444444444440
1	0.708333333333330
10	0.677777777777780
100	0.666666666666670
1000	0.666666666666670

and misclassification rate. I searched parameter c among (0.1, 1, 10, 100, 1000), Table.2 shows that c=0.1 gives the best accuracy score.

Table 3

Degree	Accuracy score
0	0.7194444444444440
1	0.7194444444444440
2	0.7083333333333330
3	0.7027777777777780
4	0.7
5	0.7027777777777780
6	0.6916666666666670
7	0.7
8	0.6944444444444440

For SVM Poly, I optimized parameter degree to decide the best polynomial degree for this dataset. I searched parameter degree from 0 to 8, Table.3 shows that degree=0 or 1 yields the best accuracy score.

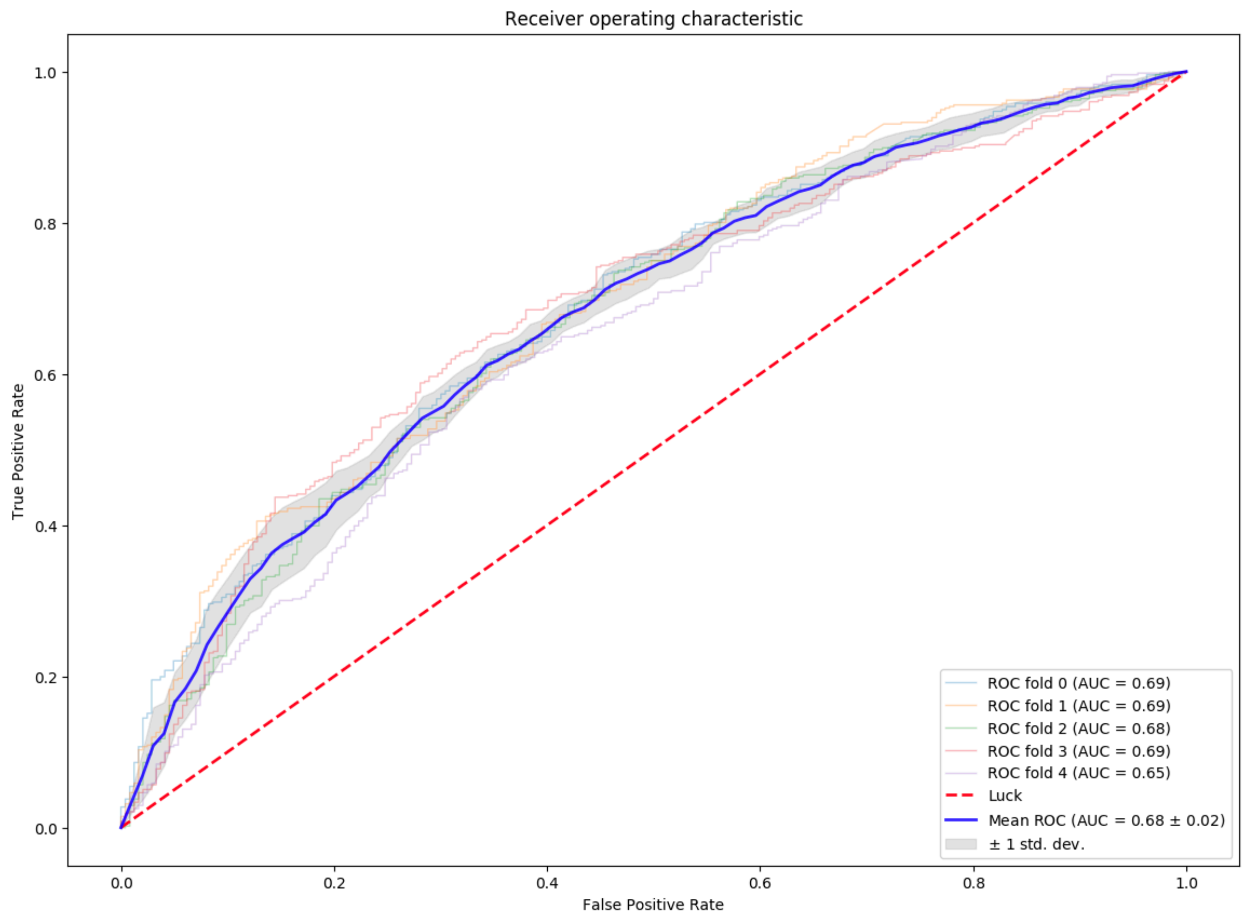
I optimized parameter learning rate for Gradient Boosting Decision Tree, it controls how much the tree

Table 4

learning rate	Accuracy score
0.05	0.7194444444444440
0.075	0.7194444444444440
0.1	0.7194444444444440
0.25	0.7138888888888890
0.5	0.7055555555555560
0.75	0.7055555555555560
1	0.6722222222222220

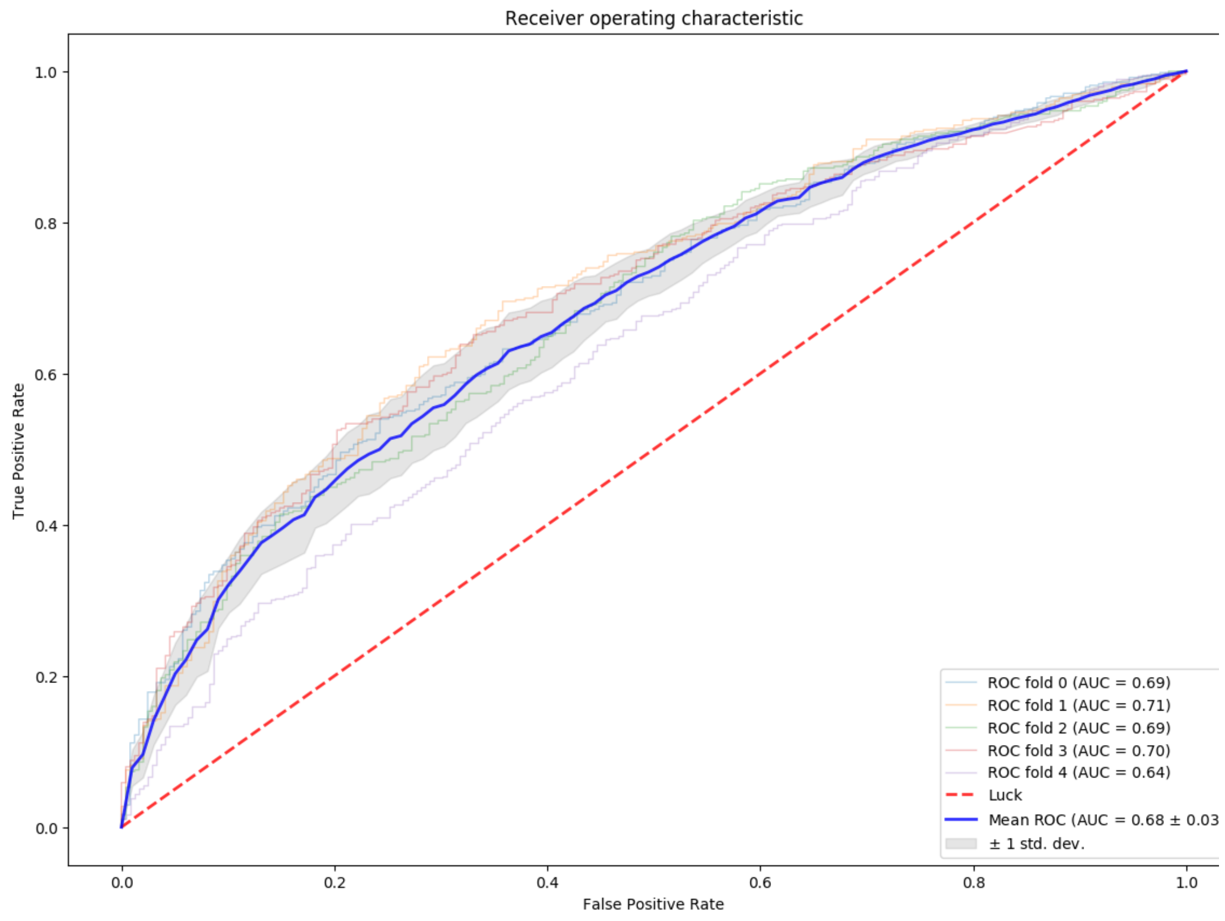
correcting the residual errors based on the previous tree. I searched parameter learning rate among (0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1), Table. 4 shows that learning rate (0.05, 0.075, 0.1) yields the best accuracy score.

In each algorithm, 5-fold stratified cross validation is performed based on the parameters that yields the best accuracy score in previous process. And five different classifiers are built by partitioning the initial dataset into five disjoint sets. During each instance, four sets are used for training and the remaining set for testing. At the end, performance metrics are averaged to get an overall result. By having it stratified, distribution of the class over the test and train sets is kept approximately the same as



Receiver operating characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC fold 0 (AUC = 0.69)
ROC fold 1 (AUC = 0.69)
ROC fold 2 (AUC = 0.68)
ROC fold 3 (AUC = 0.69)
ROC fold 4 (AUC = 0.65)
Luck
Mean ROC (AUC = 0.68 ± 0.02)
± 1 std. dev.

Fig.1



Receiver operating characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC fold 0 (AUC = 0.69)
ROC fold 1 (AUC = 0.71)
ROC fold 2 (AUC = 0.69)
ROC fold 3 (AUC = 0.70)
ROC fold 4 (AUC = 0.64)
Luck
Mean ROC (AUC = 0.68 ± 0.03
± 1 std. dev.

Fig.2

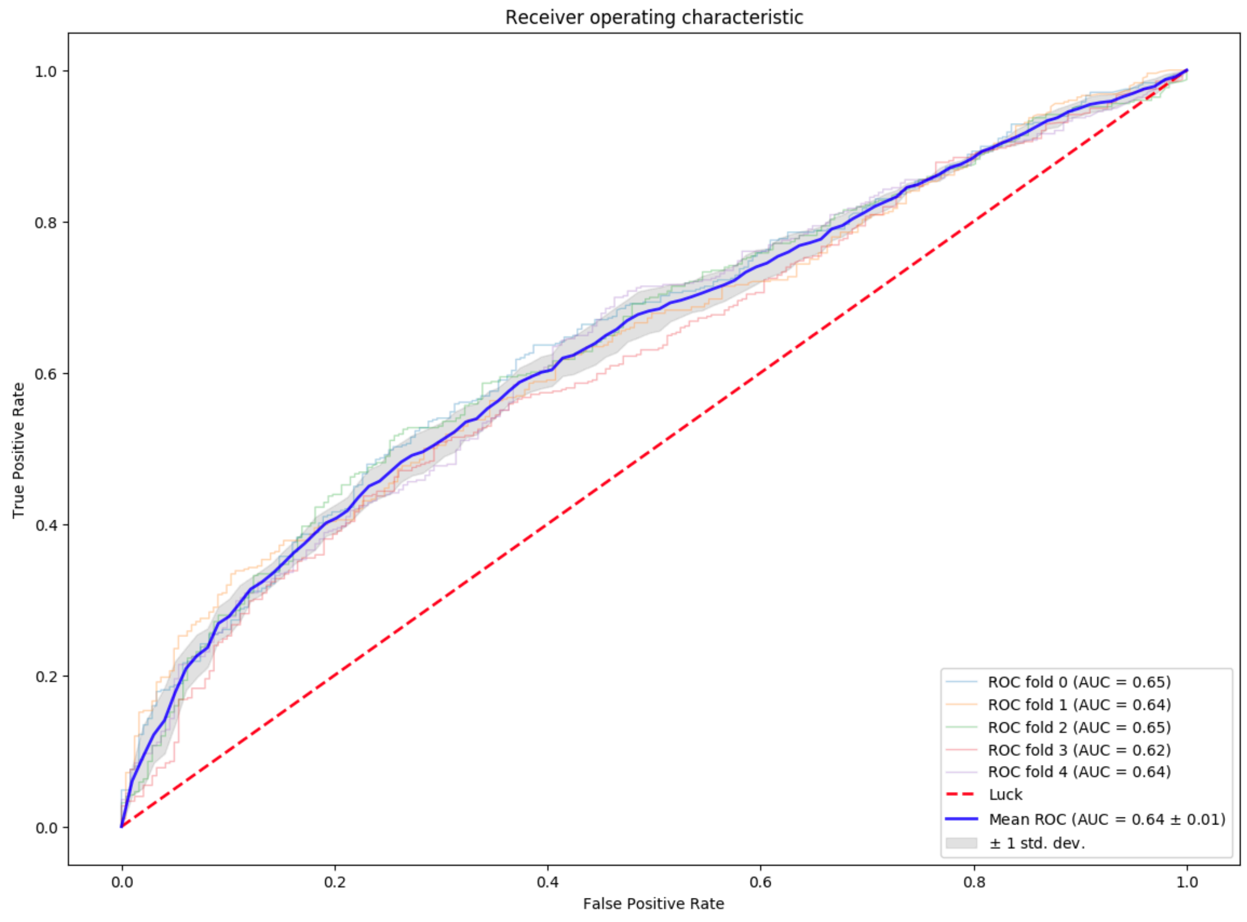


Fig. 3

the distribution of the class over the entire set. This ensures the generality of the classification model.

During this process, each fighter is assigned to the Winner or Non-winner binary class with higher probability and no further classification task is needed. ROC AUC curve is used to evaluate the classification performance. Averaged ROC score and ROC score for each fold were presented in Fig. 1 (SVM Rbf), Fig. 2 (SVM Poly), Fig. 3 (Gradient Boosting Decision Tree), Fig. 4 (Random Forest). AUC score for training and testing sets are presented, see Fig. 5 (SVM Rbf), Fig. 6 (SVM Poly), Fig. 7 (Gradient Boosting Decision Tree), Fig. 8 (Random Forest).

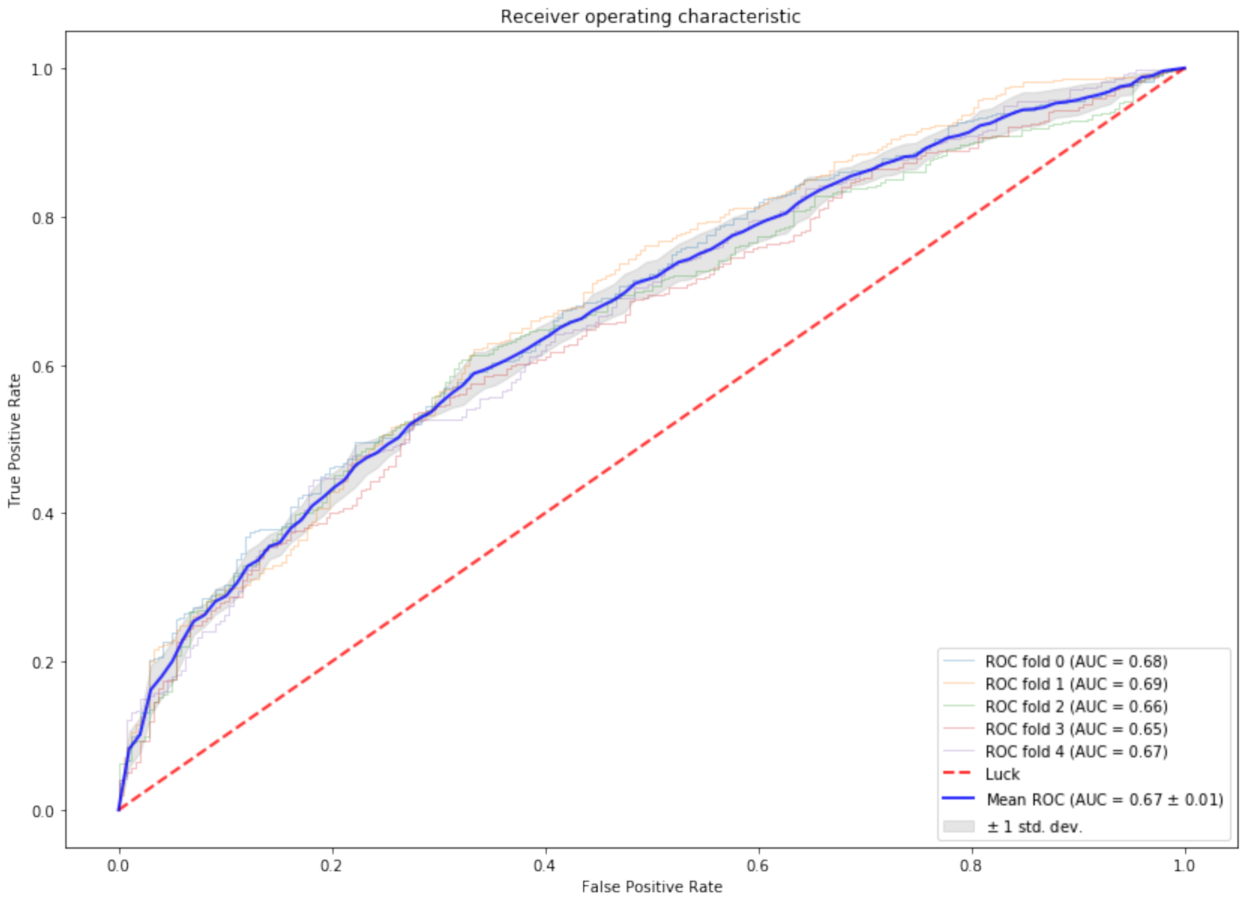
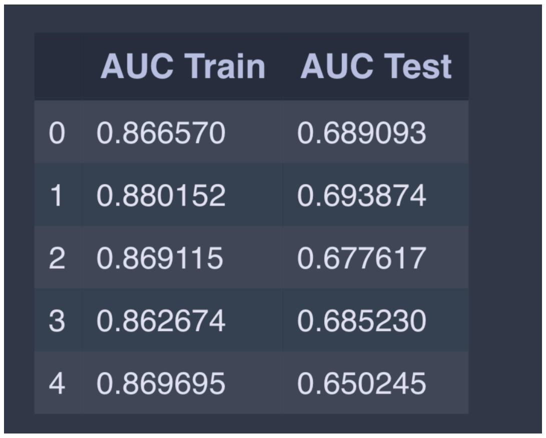
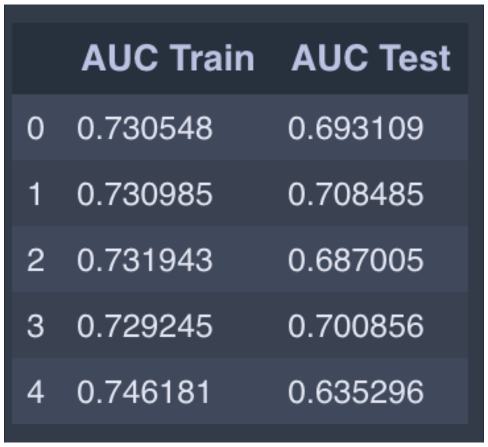


Fig. 4



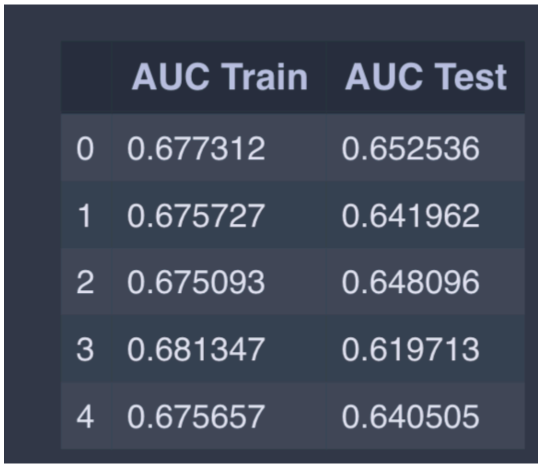
	AUC Train	AUC Test
0	0.866570	0.689093
1	0.880152	0.693874
2	0.869115	0.677617
3	0.862674	0.685230
4	0.869695	0.650245



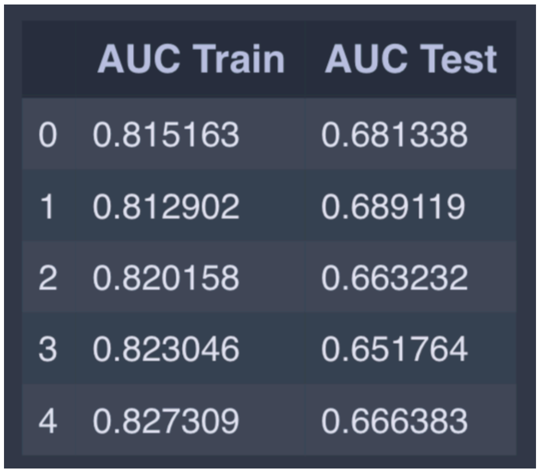
	AUC Train	AUC Test
0	0.730548	0.693109
1	0.730985	0.708485
2	0.731943	0.687005
3	0.729245	0.700856
4	0.746181	0.635296

Fig. 5

Fig. 6



	AUC Train	AUC Test
0	0.677312	0.652536
1	0.675727	0.641962
2	0.675093	0.648096
3	0.681347	0.619713
4	0.675657	0.640505



	AUC Train	AUC Test
0	0.815163	0.681338
1	0.812902	0.689119
2	0.820158	0.663232
3	0.823046	0.651764
4	0.827309	0.666383

Fig. 7 Fig. 8

I also optimized parameters, max depth, min samples split and min samples leaf for random forest using hyper parameter optimization. The method I used was Downhill Simplex.

RESULTS

SVM Rbf classification outperforms Random Forest and Gradient Boosting Decision Tree classification and it gets comparable results with classification using SVM Poly. Because SVM Rbf with parameter c=0.1 yields better ROC score than Random Forest and Gradient Boosting Decision Tree and same ROC score with SVM Poly, while having a smaller deviation which means that it’s even more stable than SVM Poly.