Prediction of Tennis Matches

David Scanlan, Johan Gerfaux, Joshua Edwards, Agustin Toll Villagra

Motivation

Tennis, a sport that has been played competitively since 1873, has changed very little since its creation. In this sport, two players challenge each other by hitting a ball with racquets over a net in aims of scoring points against their opponent. For the U.S. Open tournament which is played each year in Flushing Meadows, NY a tennis match consists of five sets, with the winner taking three of the five sets. Within a set, a player must win six games in order to win a set or seven games if the game difference between the two players is only one.

In just a few sentences, one can surely understand the vast amount of data, statistics, and information that can be attributed to a seasoned professional; the world ranking of a player, the total number of sets a player won, the number of forced errors a player makes, and so forth. With data in mind, the goal of this project is to investigate the applicability of utilizing machine learning algorithms to predict the winner of the 2020 U.S. Open tennis tournament.

The reasoning for taking the opportunity to further analyze the U.S. Open tennis tournaments derive from the group's interest in tennis. Using data that revealed information about a player's current statistics from the previous 52 weeks of playing, the group wanted to see how accurate a model could be made to correctly predict winners of this tournament. Could models predict upsets? Could the model predict semi-finalists and the true winner? These were all questions that we would soon learn.

Further purpose of creating a model that could predict tennis tournament winners with a high level of accuracy would be to increase profitability if one was a gambler and wanted to cash in on a player with high odds of winning. On the contrary, an odds making site or a casino could use this type of model to help adjust their odds so they could minimize their payout to gamblers. Evenmore so, companies that sponsor players such as Head, Penn, Nike, etc. could use this type of modeling on less significant tournaments to find the next "star" player to sponsor. Doing this would help companies sign a player for a lesser rate as they are less well known and still working through the ranks of the tennis world.

Data

Feature selection of utmost importance when designing and executing any prediction model. The team sourced it's raw data from the http://www.tennis-data.co.uk/alldata.php which contains match data from 2005 to 2020. For sake of this project, only data from 2010 to 2020

was considered to more accurately account for the tour players that are currently on the ATP men's circuit. For each year (2010 through 2020), our online source provided an Excel file with 55 features detailed in Appendix A.1.

By looking at the features and intuition from past years both playing and following tennis, we can conclude that the following features have a big impact on a match outcome:

1. Ranking
2. Match Winning percentage
3. Head to Head on Hard Court
4. Match Winning Percentage on Hard Court in the last 52 weeks (1 year)
5. Percentage of Best of 5 Set Matches Won (Majors are best of five sets)

Using the five features above as a baseline, the following twenty one features were created from the raw data for use in our predictive models. As shown below, there's a heavy emphasis on head-to-head performance as we believe this is the best way to compare two players in a match. One quick note, the "diff" part in every variable is a way to compare statistics between player 0 and player 1 of every match, thus it's a simple subtraction of player 1's individual statistic from player 0's statistic. To keep things simple, player 0 is designated as the higher ranked player for every match. For a more detailed explanation of how the following features were created please see the Appendix A.3.

1. Diff_rank
2. Diff_match_win_percent
3. Diff_games_win_percent
4. Diff_5_set_match_win_percent
5. Diff_close_sets_percent
6. Diff_match_win_percent_hard
7. Diff_games_win_percent_hard
8. Diff_5_set_match_win_percent_hard
9. Diff_close_sets_percent_hard
10. Diff_match_win_percent_52
11. Diff_games_win_percent_52'
12. Diff_5_set_match_win_percent_52
13. Diff_close_sets_percent_52
14. Diff_match_win_percent_hard_60
15. Diff_games_win_percent_hard_60
16. Diff_5_set_match_win_percent_hard_60

17. Diff_close_sets_percent_hard_60
18. Diff_match_win_percent_hh
19. Diff_games_win_percent_hh
20. Diff_match_win_percent_hard_hh
21. Diff_games_win_percent_hard_hh

A visualization of the distribution of these 21 features can be found in Appendix A.2.

Analytic Models

Since it would be a classification problem (win or lose), the machine learning algorithms that we plan to use are:

- Logistic Regression
- CART
- Random Forest
- Neural Networks
- PCA

Our main approach to evaluate and compare performance amongst the different models is accuracy. Indeed, this metric fits our case here. We want to limit the false positives and the false negatives and the accuracy takes into account these two measures. Using the dataset we created, we fitted the models and selected the one which yielded the best performance for predicting match outcomes. The first step was to create a Naive Baseline that consists of a simple classification model by which the player with higher rank at the moment of the game is always the winner. This baseline has an accuracy of 72% that is a good starting point. All other models evaluated will be compared with this value.

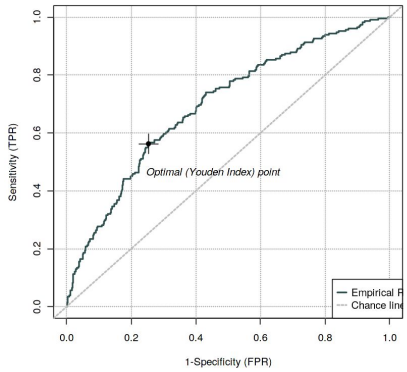
Logistic Regression

After viewing the Baseline model the next step in our analytical modeling was to develop a Logistic Regression model to view the accuracy of depicting if a given player won or lost a match based on their past 52 week playing statistics. This analysis began by first creating a Logistic Regression model with the “GLM” function in R. After modeling the outcome of a match based on all 21 variables, using the training set that consisted of 70% of the data set, the following three

variables showed the most significance:

- Diff_match_win_percent_hard
- Diff_close_sets_percent_hard
- Diff_match_win_percent_52

Upon the creation of the most optimal Logistic Regression model, an ROC curve was then created to identify at which value the optimal break-even threshold value occurred. Around a value of p = 0.25, the plotted ROC curve may be seen in the diagram below Furthermore, it was calculated that the AUC value of this RUC curve was 0.768. An AUC value of 0.768 demonstrates that 76.8% which helps demonstrate the performance of our model. This final determined AUC value shows that our model could be improved.



With our optimal break-even threshold discovered, the Logistic Regression model was used to predict the outcome of winners on the test set which consisted of 30% of the entire data set. The confusion matrix generated helps explain the outcome of the final model.

	FALSE	TRUE
0	179	76
1	34	656

With this confusion matrix, the following we calculated to help define the model

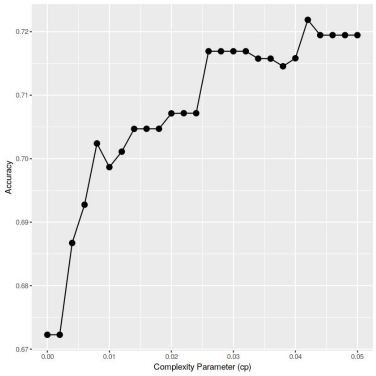
- $FPR = \frac{FP}{TN+FP} = 0.31$
- $TPR = \frac{TP}{TP+FN} = 0.65$
- $ACC = \frac{TP+TN}{TP+TN+FN+FP} = 0.69$

The FPR which is the percentage of incorrect positive predictions divided by the total number of negatives was calculated to be 0.31. This is the portion of tennis outcomes incorrectly

chosen as victories. The TPR which is the number of correct positive predictions divided by the total number of positive predictions was 0.65. This is the portion of correctly classified victories. Overall, this Logistic Regression model has an accuracy of 69% of correctly choosing the winning outcome of a tennis match. This is lower than the baseline so a higher accuracy is desired.

CART

Decision tree modeling was then introduced to this project in order to attempt to better create a predictive model. By using a CART model, the outcome of a tennis match was predicted with the assistance of the 21 different variables we had in our data set. For this model, we did not consider any loss matrix and it was decided upon to cross validate the model 10 fold. The CART model was tested using complexity parameters ("cp") ranging from 0 to 0.05. Upon execution in R using the training functions for CART modeling, a cp value of 0.042 was found to be most optimal. cp=0.042 was used to control the size of the decision tree in order to make up the most optimal tree possible; if the cost of adding a split did not exceed cp=0.042, it was not done. Using the "ggplot" library a plot was created to help determine the best cp value.



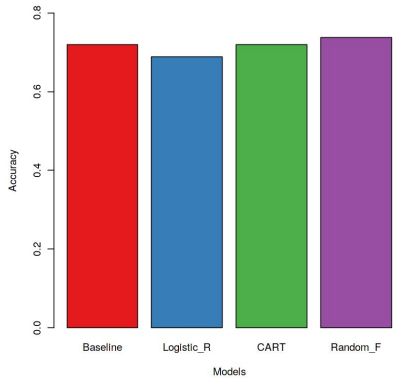
After using the training model with the testing data, the final best CART model predicted 72% of the outcomes correctly. An improvement from the Logistic Regression, we would like to even further improve this model.

Random Forest

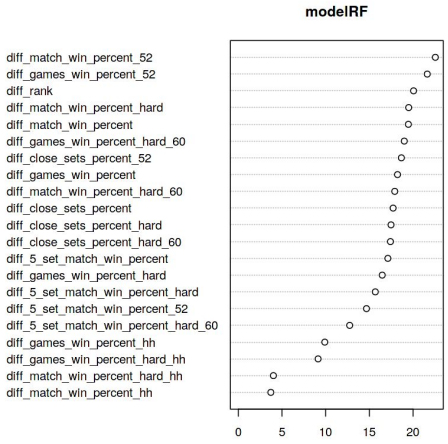
After analyzing the low accuracy that simple models like Logistic Regression anc CART have on predicting the outcome of tennis matches, we decided to try more complex models looking to improve the performance. The first model we tried was Random Forest, an algorithm invented by a former professor at UC Berkeley, Leo Breiman, and collaborators. Random Forest is a nonparametric supervised learning method that operates by constructing a "forest" of very

decorrelated decision trees.

To train this model, we used the same training and test set as in the previous models. To determine the technical parameters, we performed a 5-fold cross validation using the caret package in R, that helped us to optimize for the predicting error. This process divides the training data into 5 different groups, removes one group at a time, trains and computes the error on the rest, and then averages the results. It repeats that process for all the candidates, that in our case were models with 1 to 21 number of variables examined at each split of the fitted CART trees (mtry). The selected mtry by the tuning function was 3 and the resulting accuracy on the test set was 73.7%. It shows an improvement from Logistic Regression and CART, and for the first time, from the Baseline:



Because there are hundreds of CART trees, interpreting this model is much more difficult than the models we have seen so far in this report. What we can see is the relative importance of variables:



As we can see, there are many features that are similarly relevant when predicting tennis matches. That could be the reason why complex models perform better than simple models.

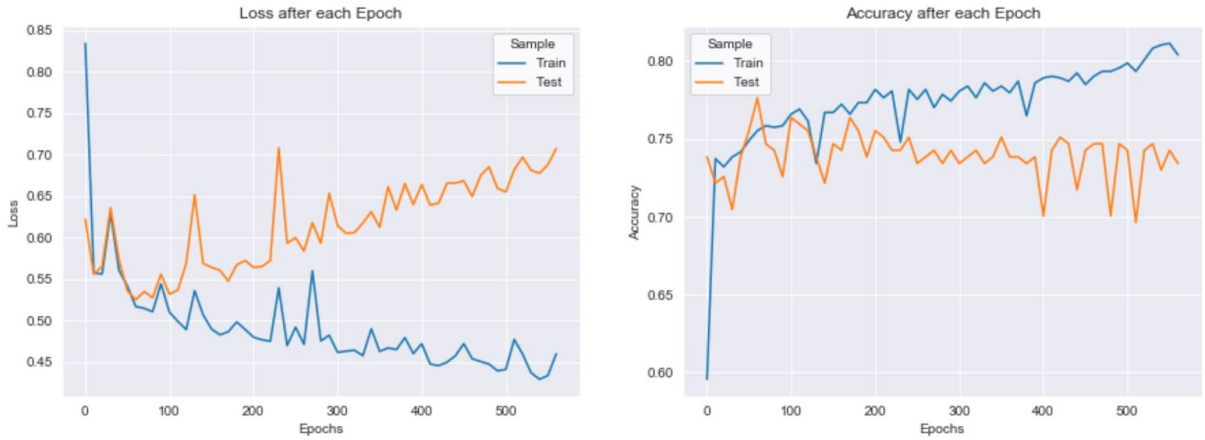
Neural Networks

A three layer neural network was created to predict the outcome of US Matches. However, before building the model using the 21 features mentioned above in the Data section, the data was split 80-20 for a training and test set in order to optimize our Neural Network on the training data and then validate it via the test data.

The following features were chosen for the model:

- Number of Layers: 3. (2 Hidden Layers)
- Number of Neurons in each layer: 64 -> 32 ->1
- Activation relu->relu->sigmoid
- Early Stopping if the validation loss does not improve for 500 epochs
- Model Checkpoint to save the model which provides the minimum validation set loss

The Model Checkpoint feature is used because as seen below, the neural network starts to overfit on the training data. What this means is that the model gains accuracy on the training set while at the same time losing accuracy on the set. Thus in order to overcome this issue, the model with the lowest Validation Loss on the test set was chosen.



The accuracy on the training and set is shown below:

Train Accuracy: 0.758, Test Accuracy: 0.776

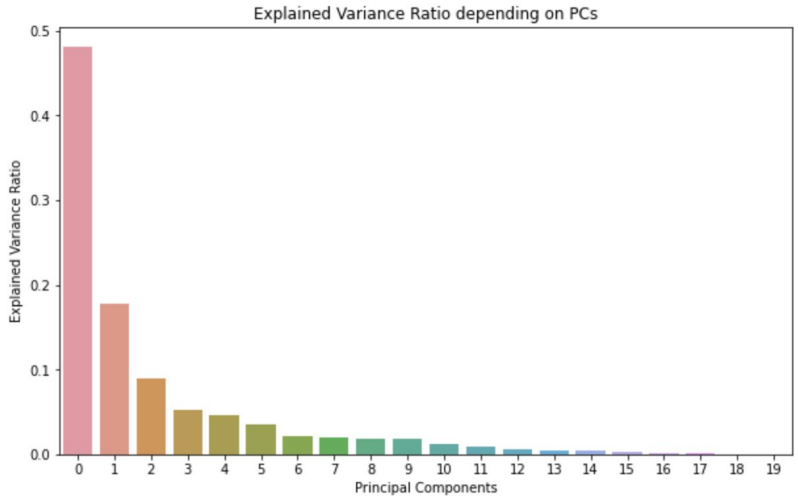
Principal Component Regression

Our database consists of approximately n = 1000 rows and p = 21 features. So we are far from being in the case where p = n. However, using a Principal Component Analysis in our case still makes a lot of sense. First, some of the features we defined can be dependent on one another. Thus the assumption that the features should be independent is not verified, and therefore this could cause a bias in our model. Thanks to the Principal Component Analysis, we generate genuine independent variables.

Also, we want to reduce the number of variables but we aren't able to identify variables to completely remove from consideration. Using principal component analysis allows use to reduce the dimensionality of the problem, and at the same time keep the important information.

Usually, little variability indicates noise whereas lots of variability indicates signal. Thus, one more advantage of using PCA is that the model does not consider the low variability features and thus is unlikely to overfit.

Here we plot the explained variance ratio for each principal component (see appendix A6 for the whole code). The sum of these ratios adds up to one.

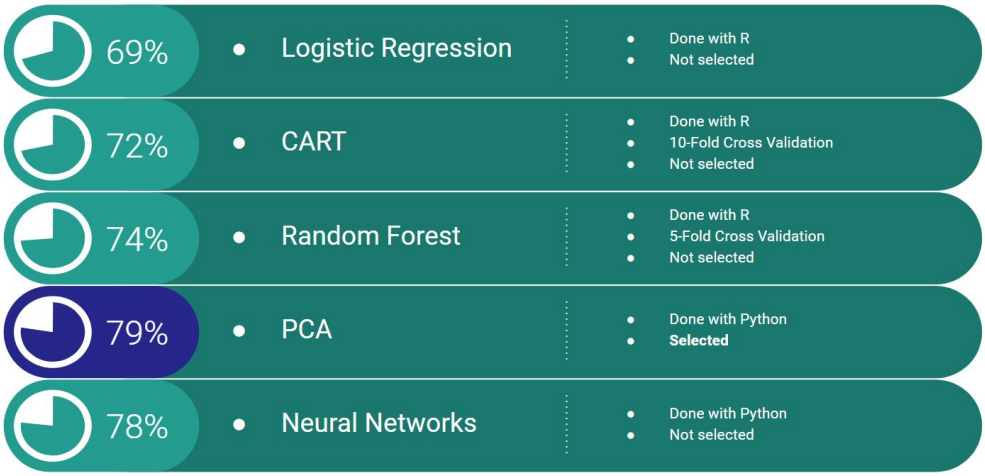


A simple method to determine how many PCs we want to keep is to find the PC from which the variation of the explained variance ratio is not significant anymore. On the graph, we can see that after PC#2, the variation of explained variance is very light. We are approximately at the "elbow" of the curve. We can conclude from this graph that selecting only the first 3 PCs is relevant. With these 3 PCs we retain 75% of the variability.

We can now apply a principal component regression model to predict the outcomes of the games and we obtain an accuracy of 79%.

Comparison and Model Selection

You can see on the following chart a comparison in accuracy of the different models that we implemented.



The principal component regression is the model that leads to the highest accuracy. Its high score is explained by the fact that it created genuinely independent variables, thus avoiding multicollinearity in the features. And also, it is less likely to overfit because it does not consider the low variability features. In the following section, we will use this model to predict the results of the US Open 2020 tournament.

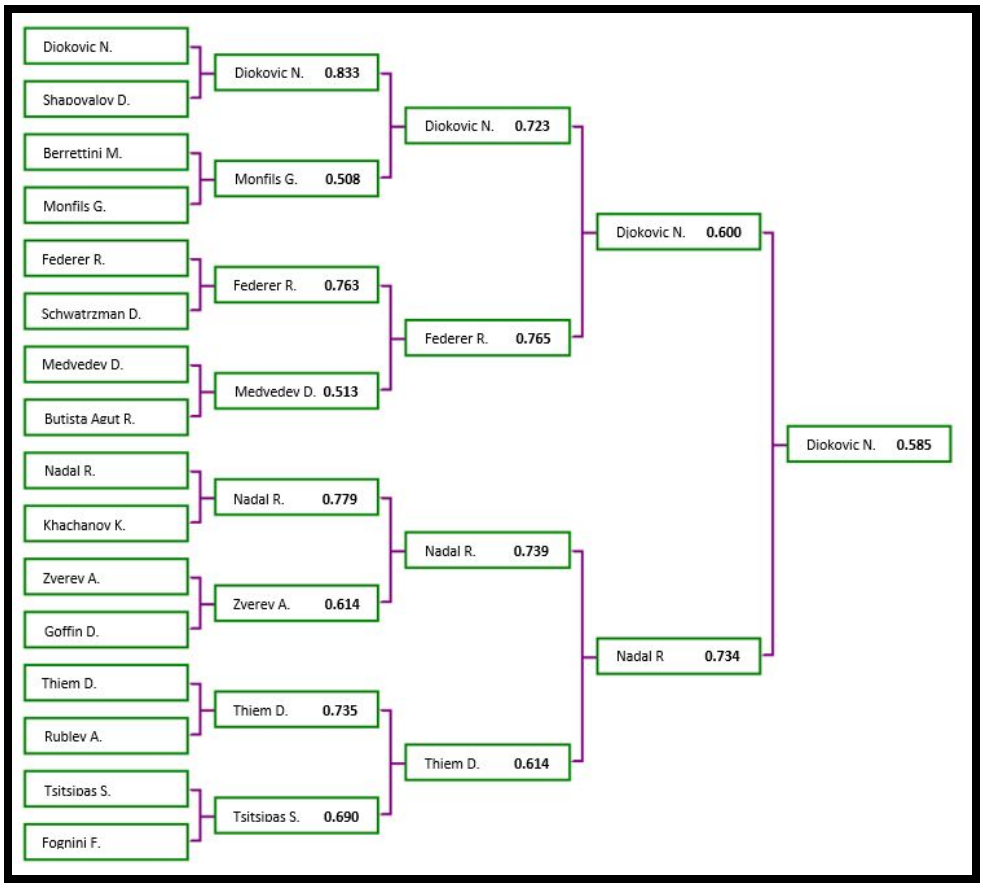
Predictions on US Open 2020

For the simulation of the US Open 2020, we started at the 4th round assuming that the 16 best-ranked players of the moment are going to play. The first match would be the player ranked number 1, Novak Djokovic, against the player ranked 16th, Denis Shapovalov; the second match would be the player ranked 2th, Rafael Nadal, against the player ranked 15th, Karen Khachanov; player ranked 3th against 14th, and so on. The current ATP ranking is available in the oficial ATP Tour website: https://www.atptour.com/en/rankings/singles.

The data that we used in this simulation was created using the same code that we used for the data mining section of the model. The code can be found in Appendix A.3. For each of the top 16 players, we used data from their last match in 2020 (Dubai Open, Mexican Open, or

Australian Open) to feed into our model. The dataset created had 8 rows for the 4th round, 16 rows for all the possible combinations of players that could happen in the Quarterfinals, 32 rows for all possible Semifinals, and 64 rows for all possible Finals.

Predictions for all these 120 matches were made using our PCA model in python because it was the one that showed the highest accuracy. With all the possible outcomes, we manually created the following bracket chart that shows the name of the players and their probability of winning:



As in real life, our model reports that the player with the higher rank is not always the winner. We included in the chart the probability of winning for the player that advanced round. There are some very even matches like Berretini against Monfils, a match for which our model predicts that the french player is winning with 50.8% probability. Additionally, while Medvedev has a much better rank than Schwatzman, we can see that our model is predicting that Roger Federer has higher chances to win Quarterfinals against Medvedev than 4th Round against Schwatzman. That shows us that our model is not only relying in rank, but considering many other variables as well. The big winner of our simulation is the current number 1 of the world, Novak Djokovic, beating Rafael Nadal in the final match with a probability of 58.5%.

References

- *IEOR 242 - Slides Week 3 - Predicting Loan Repayment with Logistic Regression* by George Ng, Spring 2020
- *IEOR 242 - Slides Week 4 - Predicting Loan Repayment and Customer Churn with Logistic Regression and LDA* by George Ng, Spring 2020
- *IEOR 242 - Slides Week 5 – Predicting Parole Violations with CART and Cross Validation* by George Ng, Spring 2020
- *IEOR 242 - Slides Week 6 – CTR Prediction with Random Forests and Boosting* by George Ng, Spring 2020
- *IEOR 242 - Slides Week 12– Regularization Methods for Prediction* by George Ng, Spring 2020
- *IEOR 242 - Slides Week 13 – Large-Scale Learning and Neural Networks* by George Ng, Spring 2020
- *An Introduction to Statistical Learning: with Applications in R* by GarethJames, Daniela Witten, Trevor Hastie and Robert Tibshirani, Springer, 2013.A PDF version of this textbook is available at http://www-bcf.usc.edu/~gareth/ISL/.
- *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data* by Hadley Wickham and Garrett Grolemund, O'Reilly Media, 2017. An online version of this book is available at http://r4ds.had.co.nz/.
- https://www.atptour.com/en/rankings/singles.
- http://www.tennis-data.co.uk/alldata.php

Appendices

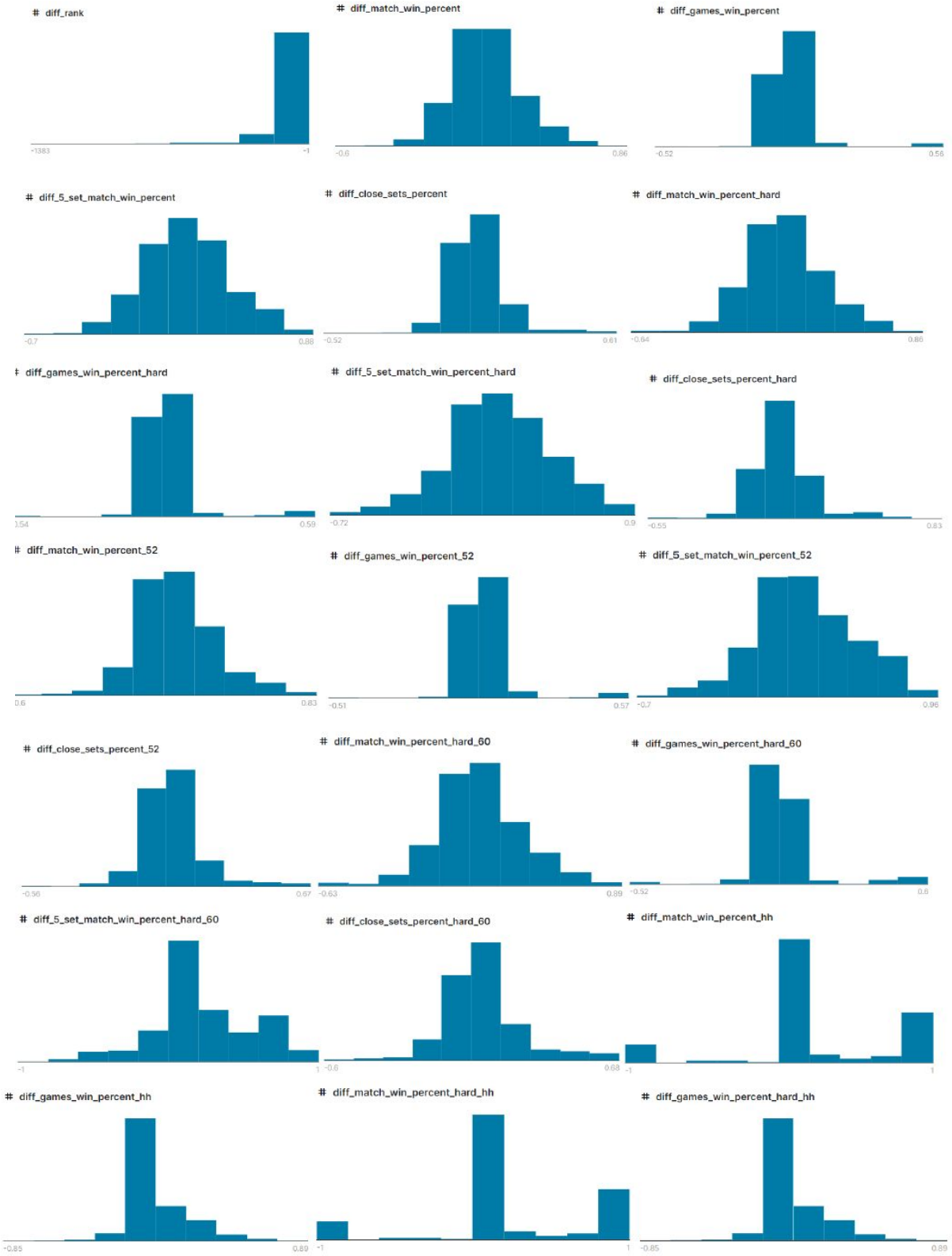
A1: List of Features

1. ATP = Tournament number (men)
2. Location = Venue of tournament
3. Tournament = Name of tournament (including sponsor if relevant)
4. Data = Date of match (note: prior to 2003 the date shown for all matches played in a single tournament is the start date)
5. Series = Name of ATP tennis series (Grand Slam, Masters, International or International Gold)
6. Tier = Tier (tournament ranking) of WTA tennis series.
7. Court = Type of court (outdoors or indoors)
8. Surface = Type of surface (clay, hard, carpet or grass)
9. Round = Round of match
10. Best of = Maximum number of sets playable in match
11. Winner = Match winner
12. Loser = Match loser
13. WRank = ATP Entry ranking of the match winner as of the start of the tournament
14. LRank = ATP Entry ranking of the match loser as of the start of the tournament
15. WPts = ATP Entry points of the match winner as of the start of the tournament
16. LPts = ATP Entry points of the match loser as of the start of the tournament
17. W1 = Number of games won in 1st set by match winner
18. L1 = Number of games won in 1st set by match loser
19. W2 = Number of games won in 2nd set by match winner
20. L2 = Number of games won in 2nd set by match loser
21. W3 = Number of games won in 3rd set by match winner
22. L3 = Number of games won in 3rd set by match loser
23. W4 = Number of games won in 4th set by match winner
24. L4 = Number of games won in 4th set by match loser
25. W5 = Number of games won in 5th set by match winner
26. L5 = Number of games won in 5th set by match loser
27. Wsets = Number of sets won by match winner
28. Lsets = Number of sets won by match loser
29. Comment = Comment on the match (Completed, won through retirement of loser, or via

Walkover)

30. B365W = Bet365 odds of match winner
31. B365L = Bet365 odds of match loser
32. B&WW = Bet&Win odds of match winner
33. B&WL = Bet&Win odds of match loser
34. CBW = Centrebet odds of match winner
35. CBL = Centrebet odds of match loser
36. EXW = Expekt odds of match winner
37. EXL = Expekt odds of match loser
38. LBW = Ladbrokes odds of match winner
39. LBL = Ladbrokes odds of match loser
40. GBW = Gamebookers odds of match winner
41. GBL = Gamebookers odds of match loser
42. IWW = Interwetten odds of match winner
43. IWL = Interwetten odds of match loser
44. PSW = Pinnacles Sports odds of match winner
45. PSL = Pinnacles Sports odds of match loser
46. SBW = Sportingbet odds of match winner
47. SBL = Sportingbet odds of match loser
48. SJW = Stan James odds of match winner
49. SJL = Stan James odds of match loser
50. UBW = Unibet odds of match winner
51. UBL = Unibet odds of match loser
52. MaxW= Maximum odds of match winner (as shown by Oddsportal.com)
53. MaxL= Maximum odds of match loser (as shown by Oddsportal.com)
54. AvgW= Average odds of match winner (as shown by Oddsportal.com)
55. AvgL= Average odds of match loser (as shown by Oddsportal.com)

A2: Data Visualization



A3: Code Data Mining

Feature Definition

```
from datetime import datetime
from datetime import timedelta

def get_player_1_name(winner_name, winner_rank, loser_name, loser_rank):
    """
    :param winner_name: Name of winner
    :param winner_rank: Rank of the winner
    :param loser_name: Name of loser
    :param loser_rank: Rank of the loser
    :return: name of higher ranked player
    """
    if winner_rank < loser_rank:
        return winner_name
    else:
        return loser_name

def get_player_1_rank(winner_rank, loser_rank):
    """
    :param winner_rank: Rank of the winner
    :param loser_rank: Rank of the loser
    :return: rank of higher ranked player
    """
    if winner_rank < loser_rank:
        return winner_rank
    else:
        return loser_rank

def get_player_2_name(winner_name, winner_rank, loser_name, loser_rank):
    """
    :param winner_name: Name of winner
    :param winner_rank: Rank of the winner
    :param loser_name: Name of loser
    :param loser_rank: Rank of the loser
    :return: name of lower ranked player
    """
    if winner_rank > loser_rank:
        return winner_name
    else:
        return loser_name

def get_player_2_rank(winner_rank, loser_rank):
    """
    :param winner_rank: Rank of the winner
```

```
    :param loser_rank: Rank of the loser
    :return: rank of lower ranked player
    """
    if winner_rank > loser_rank:
        return winner_rank
    else:
        return loser_rank

def outcome(winner_rank, loser_rank):
    """
    Returns 0 if higher ranked player won and 0 otherwise
    :param winner_rank: Rank of the winner
    :param loser_rank: Rank of the loser
    :return: Odds of the Higher ranked player
    """
    if winner_rank< loser_rank:
        return 0
    else:
        return 1

def get_player_1_odd(winner_odd, winner_rank, loser_odd, loser_rank):
    """
    Returns the odds of Higher ranked player
    :param winner_odd: Odds of the winner
    :param winner_rank: Rank of the winner
    :param loser_odd: Odds of the Loser
    :param loser_rank: Rank of the Loser
    :return: Odds of the Higher ranked player
    """
    if winner_rank < loser_rank:
        return winner_odd
    else:
        return loser_odd

def get_player_2_odd(winner_odd, winner_rank, loser_odd, loser_rank):
    """
    Returns the odds of Lower ranked player
    :param winner_odd:
    :param winner_rank:
    :param loser_odd:
    :param loser_rank:
    :return:
    """
    if winner_rank > loser_rank:
        return winner_odd
    else:
        return loser_odd
```

```
def subtract_days(date_string, num_days):
    """
    Subtract n days from a specified date
    :param date_string: pass date in format '%Y/%m/%d'
    :param num_days: Number of days to be subtracting from the date
    :return: date in format '%Y/%m/%d'
    """
    date_temp = (datetime.strptime(date_string, '%Y/%m/%d') - timedelta(days=num_days))
    return date_temp.strftime("%Y/%m/%d")


def winning_percentage(player_id, data,  type1='matches', current_date=None, surface='All', last_n_weeks=0):
    """
    Caculate different player stats
    :param player_id: Name or ID of Player
    :param data: The raw dataframe from http://www.tennis-data.co.uk/alldata.php
    :param type1: Options: ['matches', 'total_matches', 'games', 'matches_5_sets', 'win_or_close_sets']
    :param current_date: Date of match
    :param surface: Surface options: ['All', 'Grass', 'Hard', 'Clay']
    :param last_n_weeks: Get stats from the past n weeks
    :return: Returns the players Stat for the specified parameters.
    """
    data = data[data['Date'] < current_date]

    if surface!='All':
        data = data[data['Surface'] == surface]

    if last_n_weeks>0:
        last_date = subtract_days(current_date, (last_n_weeks * 7))
        data = data[data['Date'] >= last_date]

    if type1 == 'matches':
        wins = (data['Winner'] == player_id).sum()
        loses = (data['Loser'] == player_id).sum()

    elif type1 == 'total_matches':
        return (data['Winner'] == player_id).sum() + (data['Loser'] == player_id).sum()


    elif type1 == 'matches_5_sets':
        wins = ((data['Winner'] == player_id) & (data['best_of_5'] == 1)).sum()
        loses = ((data['Loser'] == player_id) & (data['best_of_5'] == 1)).sum()


    elif type1 == 'games':
        winner_set_list = ['W1', 'W2', 'W3', 'W4', 'W5']
        loser_set_list = ['L1', 'L2', 'L3', 'L4', 'L5']
```

```
                wins = data[data['Winner'] == player_id][winner_set_list].values.sum() + data[data['Loser'] == player_id][loser_set_list].values.sum()
                loses = data[data['Loser'] == player_id][winner_set_list].values.sum() + data[data['Winner'] == player_id][loser_set_list].values.sum()


    elif type1 == 'win_or_close_sets':

        wins = 0
        loses = 0

        data_3_set = data[data['best_of_5'] == 0]
        data_5_set = data[data['best_of_5'] == 1]

        for i in range(1, 4):
            wins = wins + ((data_3_set['Winner'] == player_id) & (data_3_set[('W' + str(i))] >= 5)).sum()
            wins = wins + ((data_3_set['Loser'] == player_id) & (data_3_set[('L' + str(i))] >= 5)).sum()
            loses = loses + ((data_3_set['Winner'] == player_id) & (data_3_set[('W' + str(i))] < 5)).sum()
            loses = loses + ((data_3_set['Loser'] == player_id) & (data_3_set[('L' + str(i))] < 5)).sum()

        for i in range(1, 6):
            wins = wins + ((data_5_set['Winner'] == player_id) & (data_5_set[('W' + str(i))] >= 5)).sum()
            wins = wins + ((data_5_set['Loser'] == player_id) & (data_5_set[('L' + str(i))] >= 5)).sum()
            loses = loses + ((data_5_set['Winner'] == player_id) & (data_5_set[('W' + str(i))] < 5)).sum()
            loses = loses + ((data_5_set['Loser'] == player_id) & (data_5_set[('L' + str(i))] < 5)).sum()

    total = wins + loses

    if total <2:
        win_percent = 0

    else:
        win_percent = wins / total
    return win_percent


def winning_percent_hh(player_name, opponent_name, data, type1='matches', current_date=None, surface='All', last_n_weeks=0):
    """
    :param player_name: Name of player
    :param opponent_name: Name of opponent
    :param data: The raw dataframe from http://www.tennis-data.co.uk/alldata.php
    :param type1: Options: ['matches', 'games', ]
    :param current_date: Date of the match
    :param surface: Surface options: ['All', 'Grass', 'Hard', 'Clay']
    :param last_n_weeks: Get stats from the past n weeks
    :return: Returns the players Head to Head Stat.
```

```
    """
    data = data[data['Date'] < current_date]

    if surface!='All':
        data = data[data['Surface'] == surface]

    if last_n_weeks>0:
        last_date = subtract_days(current_date, (last_n_weeks * 7))
        data = data[data['Date'] >= last_date]

    if type1 == 'matches':
        wins = ((data['Winner'] == player_name) & (data['Loser'] == opponent_name)).sum()
        loses = ((data['Winner'] == opponent_name) & (data['Loser'] == player_name)).sum()

    elif type1 == 'games':
        winner_set_list = ['W1', 'W2', 'W3', 'W4', 'W5']
        loser_set_list = ['L1', 'L2', 'L3', 'L4', 'L5']

        wins = data[(data['Winner'] == player_name) & (data['Loser'] == opponent_name)][winner_set_list].values.sum() + \
               data[(data['Winner'] == opponent_name) & (data['Loser'] == player_name)][loser_set_list].values.sum()

        loses = data[(data['Winner'] == opponent_name) & (data['Loser'] == player_name)][winner_set_list].values.sum() + \
                data[(data['Winner'] == player_name) & (data['Loser'] == opponent_name)][loser_set_list].values.sum()

    total = wins + loses

    if total == 0:
        win_percent = 0

    else:
        win_percent = wins / total
    return win_percent

def create_features(df_combined, df):
    """
    :param df_combined: All matched with player_0 as higher ranked player
    :param df: The raw dataframe from http://www.tennis-data.co.uk/alldata.php
    :return: A data_frame with player features for each match
    """

    # Player Career Stats All Surface

print('Creating Player Career Stats All Surface')

    df_combined.loc[:, 'player_0_match_win_percent'] = df_combined.apply(
        lambda row: winning_percentage(row['player_0'], df, type1='matches', current_date=row['Date'], last_n_weeks=0),
        axis=1)
    df_combined.loc[:, 'player_1_match_win_percent'] = df_combined.apply(
```

```
    lambda row: winning_percentage(row['player_1'], df, type1='matches', current_date=row['Date'], last_n_weeks=0),
    axis=1)

df_combined.loc[:, 'player_0_games_win_percent'] = df_combined.apply(
    lambda row: winning_percentage(row['player_0'], df, type1='games', current_date=row['Date'], last_n_weeks=0),
    axis=1)
df_combined.loc[:, 'player_1_games_win_percent'] = df_combined.apply(
    lambda row: winning_percentage(row['player_1'], df, type1='games', current_date=row['Date'], last_n_weeks=0),
    axis=1)

df_combined.loc[:, 'player_0_5_set_match_win_percent'] = df_combined.apply(
    lambda row: winning_percentage(row['player_0'], df, type1='matches_5_sets', current_date=row['Date'],
                                   last_n_weeks=0), axis=1)
df_combined.loc[:, 'player_1_5_set_match_win_percent'] = df_combined.apply(
    lambda row: winning_percentage(row['player_1'], df, type1='matches_5_sets', current_date=row['Date'],
                                   last_n_weeks=0), axis=1)

df_combined.loc[:, 'player_0_close_sets_percent'] = df_combined.apply(
    lambda row: winning_percentage(row['player_0'], df, type1='win_or_close_sets', current_date=row['Date'],
                                   last_n_weeks=0), axis=1)
df_combined.loc[:, 'player_1_close_sets_percent'] = df_combined.apply(
    lambda row: winning_percentage(row['player_1'], df, type1='win_or_close_sets', current_date=row['Date'],
                                   last_n_weeks=0), axis=1)

# Player Career Stats on Grass/Clay/Hard

print('Creating Player Career Stats on Grass/Clay/Hard')

df_combined.loc[:, 'player_0_match_win_percent_hard'] = df_combined.apply(
    lambda row: winning_percentage(row['player_0'], df, type1='matches', current_date=row['Date'],
                                   surface=row['Surface'], last_n_weeks=0), axis=1)
df_combined.loc[:, 'player_1_match_win_percent_hard'] = df_combined.apply(
    lambda row: winning_percentage(row['player_1'], df, type1='matches', current_date=row['Date'],
                                   surface=row['Surface'], last_n_weeks=0), axis=1)

df_combined.loc[:, 'player_0_games_win_percent_hard'] = df_combined.apply(
    lambda row: winning_percentage(row['player_0'], df, type1='games', current_date=row['Date'],
                                   surface=row['Surface'], last_n_weeks=0), axis=1)
df_combined.loc[:, 'player_1_games_win_percent_hard'] = df_combined.apply(
    lambda row: winning_percentage(row['player_1'], df, type1='games', current_date=row['Date'],
                                   surface=row['Surface'], last_n_weeks=0), axis=1)

df_combined.loc[:, 'player_0_5_set_match_win_percent_hard'] = df_combined.apply(
    lambda row: winning_percentage(row['player_0'], df, type1='matches_5_sets', current_date=row['Date'],
                                   surface=row['Surface'], last_n_weeks=0), axis=1)
df_combined.loc[:, 'player_1_5_set_match_win_percent_hard'] = df_combined.apply(
    lambda row: winning_percentage(row['player_1'], df, type1='matches_5_sets', current_date=row['Date'],
                                   surface=row['Surface'], last_n_weeks=0), axis=1)
```

```
df_combined.loc[:, 'player_0_close_sets_percent_hard'] = df_combined.apply(
    lambda row: winning_percentage(row['player_0'], df, type1='win_or_close_sets', current_date=row['Date'],
                                   surface=row['Surface'], last_n_weeks=0), axis=1)
df_combined.loc[:, 'player_1_close_sets_percent_hard'] = df_combined.apply(
    lambda row: winning_percentage(row['player_1'], df, type1='win_or_close_sets', current_date=row['Date'],
                                   surface=row['Surface'], last_n_weeks=0), axis=1)

# Player Career Stats All Surface Last 52 Weeks

print('Creating Player Career Stats All Surface Last 52 Weeks')

df_combined.loc[:, 'player_0_match_win_percent_52'] = df_combined.apply(
    lambda row: winning_percentage(row['player_0'], df, type1='matches', current_date=row['Date'], last_n_weeks=52),
    axis=1)
df_combined.loc[:, 'player_1_match_win_percent_52'] = df_combined.apply(
    lambda row: winning_percentage(row['player_1'], df, type1='matches', current_date=row['Date'], last_n_weeks=52),
    axis=1)

df_combined.loc[:, 'player_0_games_win_percent_52'] = df_combined.apply(
    lambda row: winning_percentage(row['player_0'], df, type1='games', current_date=row['Date'], last_n_weeks=52),
    axis=1)
df_combined.loc[:, 'player_1_games_win_percent_52'] = df_combined.apply(
    lambda row: winning_percentage(row['player_1'], df, type1='games', current_date=row['Date'], last_n_weeks=52),
    axis=1)

df_combined.loc[:, 'player_0_5_set_match_win_percent_52'] = df_combined.apply(
    lambda row: winning_percentage(row['player_0'], df, type1='matches_5_sets', current_date=row['Date'],
                                   last_n_weeks=52), axis=1)
df_combined.loc[:, 'player_1_5_set_match_win_percent_52'] = df_combined.apply(
    lambda row: winning_percentage(row['player_1'], df, type1='matches_5_sets', current_date=row['Date'],
                                   last_n_weeks=52), axis=1)

df_combined.loc[:, 'player_0_close_sets_percent_52'] = df_combined.apply(
    lambda row: winning_percentage(row['player_0'], df, type1='win_or_close_sets', current_date=row['Date'],
                                   last_n_weeks=52), axis=1)
df_combined.loc[:, 'player_1_close_sets_percent_52'] = df_combined.apply(
    lambda row: winning_percentage(row['player_1'], df, type1='win_or_close_sets', current_date=row['Date'],
                                   last_n_weeks=52), axis=1)

# Player Career Stats on Grass/Clay/Hard Last 60 Weeks

print('Creating Player Career Stats on Grass/Clay/Hard Last 60 Weeks')

df_combined.loc[:, 'player_0_match_win_percent_hard_60'] = df_combined.apply(
    lambda row: winning_percentage(row['player_0'], df, type1='matches', current_date=row['Date'],
                                   surface=row['Surface'], last_n_weeks=60), axis=1)
df_combined.loc[:, 'player_1_match_win_percent_hard_60'] = df_combined.apply(
```

```
    lambda row: winning_percentage(row['player_1'], df, type1='matches', current_date=row['Date'],
                                   surface=row['Surface'], last_n_weeks=60), axis=1)

df_combined.loc[:, 'player_0_games_win_percent_hard_60'] = df_combined.apply(
    lambda row: winning_percentage(row['player_0'], df, type1='games', current_date=row['Date'],
                                   surface=row['Surface'], last_n_weeks=60), axis=1)
df_combined.loc[:, 'player_1_games_win_percent_hard_60'] = df_combined.apply(
    lambda row: winning_percentage(row['player_1'], df, type1='games', current_date=row['Date'],
                                   surface=row['Surface'], last_n_weeks=60), axis=1)

df_combined.loc[:, 'player_0_5_set_match_win_percent_hard_60'] = df_combined.apply(
    lambda row: winning_percentage(row['player_0'], df, type1='matches_5_sets', current_date=row['Date'],
                                   surface=row['Surface'], last_n_weeks=60), axis=1)
df_combined.loc[:, 'player_1_5_set_match_win_percent_hard_60'] = df_combined.apply(
    lambda row: winning_percentage(row['player_1'], df, type1='matches_5_sets', current_date=row['Date'],
                                   surface=row['Surface'], last_n_weeks=60), axis=1)

df_combined.loc[:, 'player_0_close_sets_percent_hard_60'] = df_combined.apply(
    lambda row: winning_percentage(row['player_0'], df, type1='win_or_close_sets', current_date=row['Date'],
                                   surface=row['Surface'], last_n_weeks=60), axis=1)
df_combined.loc[:, 'player_1_close_sets_percent_hard_60'] = df_combined.apply(
    lambda row: winning_percentage(row['player_1'], df, type1='win_or_close_sets', current_date=row['Date'],
                                   surface=row['Surface'], last_n_weeks=60), axis=1)

# Player Head to Head Career Stats All Surface

print('Creating Player Head to Head Career Stats All Surface')

df_combined.loc[:, 'player_0_match_win_percent_hh'] = df_combined.apply(
    lambda row: winning_percent_hh(row['player_0'], row['player_1'], df, type1='matches', current_date=row['Date'],
                                   last_n_weeks=0), axis=1)
df_combined.loc[:, 'player_1_match_win_percent_hh'] = df_combined.apply(
    lambda row: winning_percent_hh(row['player_1'], row['player_0'], df, type1='matches', current_date=row['Date'],
                                   last_n_weeks=0), axis=1)

df_combined.loc[:, 'player_0_games_win_percent_hh'] = df_combined.apply(
    lambda row: winning_percent_hh(row['player_0'], row['player_1'], df, type1='games', current_date=row['Date'],
                                   last_n_weeks=0), axis=1)
df_combined.loc[:, 'player_1_games_win_percent_hh'] = df_combined.apply(
    lambda row: winning_percent_hh(row['player_1'], row['player_0'], df, type1='games', current_date=row['Date'],
                                   last_n_weeks=0), axis=1)

# Player Head to Head Career Stats On Hard Court

print('Creating Player Head to Head Career Stats On Hard Court')

df_combined.loc[:, 'player_0_match_win_percent_hard_hh'] = df_combined.apply(
    lambda row: winning_percent_hh(row['player_0'], row['player_1'], df, type1='matches', current_date=row['Date'],
```

```
                                last_n_weeks=0), axis=1)
    df_combined.loc[:, 'player_1_match_win_percent_hard_hh'] = df_combined.apply(
        lambda row: winning_percent_hh(row['player_1'], row['player_0'], df, type1='matches', current_date=row['Date'],
                                last_n_weeks=0), axis=1)

    df_combined.loc[:, 'player_0_games_win_percent_hard_hh'] = df_combined.apply(
        lambda row: winning_percent_hh(row['player_0'], row['player_1'], df, type1='games', current_date=row['Date'],
                                last_n_weeks=0), axis=1)
    df_combined.loc[:, 'player_1_games_win_percent_hard_hh'] = df_combined.apply(
        lambda row: winning_percent_hh(row['player_1'], row['player_0'], df, type1='games', current_date=row['Date'],
                                last_n_weeks=0), axis=1)

    # Difference variables

    print('Creating Difference Variables')

    df_combined.loc[:, 'diff_match_win_percent'] = df_combined['player_0_match_win_percent'] - df_combined[
        'player_1_match_win_percent']
    df_combined.loc[:, 'diff_games_win_percent'] = df_combined['player_0_games_win_percent'] - df_combined[
        'player_1_games_win_percent']
            df_combined.loc[:, 'diff_5_set_match_win_percent'] = df_combined['player_0_5_set_match_win_percent'] -
df_combined[
        'player_1_5_set_match_win_percent']
    df_combined.loc[:, 'diff_close_sets_percent'] = df_combined['player_0_close_sets_percent'] - df_combined[
        'player_1_close_sets_percent']

    df_combined.loc[:, 'diff_match_win_percent_hard'] = df_combined['player_0_match_win_percent_hard'] - df_combined[
        'player_1_match_win_percent_hard']
            df_combined.loc[:, 'diff_games_win_percent_hard'] = df_combined['player_0_games_win_percent_hard'] -
df_combined[
        'player_1_games_win_percent_hard']
    df_combined.loc[:, 'diff_5_set_match_win_percent_hard'] = df_combined['player_0_5_set_match_win_percent_hard'] - \
                                          df_combined['player_1_5_set_match_win_percent_hard']
    df_combined.loc[:, 'diff_close_sets_percent_hard'] = df_combined['player_0_close_sets_percent_hard'] - \
                                         df_combined['player_1_close_sets_percent_hard']

    df_combined.loc[:, 'diff_match_win_percent_52'] = df_combined['player_0_match_win_percent_52'] - df_combined[
        'player_1_match_win_percent_52']
    df_combined.loc[:, 'diff_games_win_percent_52'] = df_combined['player_0_games_win_percent_52'] - df_combined[
        'player_1_games_win_percent_52']
    df_combined.loc[:, 'diff_5_set_match_win_percent_52'] = df_combined['player_0_5_set_match_win_percent_52'] - \
                                          df_combined['player_1_5_set_match_win_percent_52']
    df_combined.loc[:, 'diff_close_sets_percent_52'] = df_combined['player_0_close_sets_percent_52'] - df_combined[
        'player_1_close_sets_percent_52']

    df_combined.loc[:, 'diff_match_win_percent_hard_60'] = df_combined['player_0_match_win_percent_hard_60'] - \
                                          df_combined['player_1_match_win_percent_hard_60']
    df_combined.loc[:, 'diff_games_win_percent_hard_60'] = df_combined['player_0_games_win_percent_hard_60'] - \
```

```
                                                df_combined['player_1_games_win_percent_hard_60']
    df_combined.loc[:, 'diff_5_set_match_win_percent_hard_60'] = df_combined[
                                                    'player_0_5_set_match_win_percent_hard_60'] - \
                                                df_combined[
                                                    'player_1_5_set_match_win_percent_hard_60']
    df_combined.loc[:, 'diff_close_sets_percent_hard_60'] = df_combined['player_0_close_sets_percent_hard_60'] - \
                                                df_combined['player_1_close_sets_percent_hard_60']

    df_combined.loc[:, 'diff_match_win_percent_hh'] = df_combined['player_0_match_win_percent_hh'] - df_combined[
        'player_1_match_win_percent_hh']
    df_combined.loc[:, 'diff_games_win_percent_hh'] = df_combined['player_0_games_win_percent_hh'] - df_combined[
        'player_1_games_win_percent_hh']

    df_combined.loc[:, 'diff_match_win_percent_hard_hh'] = df_combined['player_0_match_win_percent_hard_hh'] - \
                                                df_combined['player_1_match_win_percent_hard_hh']
    df_combined.loc[:, 'diff_games_win_percent_hard_hh'] = df_combined['player_0_games_win_percent_hard_hh'] - \
                                                df_combined['player_1_games_win_percent_hard_hh']

    return df_combined
```

Feature Creation

```
from utils.create_features_utils_USOpen import *
import pandas as pd

df = pd.DataFrame()

for i in range(2005, 2020):
    if i <=2012:
        link = 'data/' + str(i) + '.xls'
    else:
        link = 'data/' + str(i) + '.xlsx'
    df_temp = pd.read_excel(link)
    df = df.append(df_temp, sort=False)

df = df.reset_index()

df = df[df.Date.notnull()]

df['Date'] = df.apply(lambda row: datetime.strptime(str(row['Date']), "%Y-%m-%d %H:%M:%S").strftime("%Y/%m/%d"),
axis=1)

df.reset_index(inplace=True)

del (df['index'])

df = df[df.Comment == 'Completed']
```

```
df.loc[:,'best_of_5'] = (df['Best of'] ==5).astype(int)
df['W3'] = pd.to_numeric(df['W3'], errors='coerce')
df['L3'] = pd.to_numeric(df['L3'], errors='coerce')

df = df.fillna(0)

print(df.dtypes)

df_combined = df[['Tournament',  'Date', 'Surface', 'Round']].copy()

df_combined.loc[:,'player_0'] = df.apply(lambda row: get_player_1_name(row['Winner'], row['WRank'], row['Loser'], row['LRank']), axis=1)
df_combined.loc[:,'player_0_rank'] = df.apply(lambda row: get_player_1_rank(row['WRank'], row['LRank']), axis=1)
df_combined.loc[:,'player_0_odd'] = df.apply(lambda row: get_player_1_odd(row['B365W'], row['WRank'], row['B365L'], row['LRank']), axis=1)


df_combined.loc[:,'player_1'] = df.apply(lambda row: get_player_2_name(row['Winner'], row['WRank'], row['Loser'], row['LRank']), axis=1)
df_combined.loc[:,'player_1_rank'] = df.apply(lambda row: get_player_2_rank(row['WRank'], row['LRank']), axis=1)
df_combined.loc[:,'player_1_odd'] = df.apply(lambda row: get_player_2_odd(row['B365W'], row['WRank'], row['B365L'], row['LRank']), axis=1)

df_combined.loc[:,'outcome'] = df.apply(lambda row: outcome(row['WRank'], row['LRank']), axis=1)

df_combined = df_combined[df_combined.Tournament == 'US Open']

df_combined = df_combined[df_combined.Date > '2010/01/01']

df_combined = create_features(df_combined, df)

print(df_combined.columns)

df_combined.to_csv('data/USOpen_matches_with_feature.csv', index=False)

df.to_csv('data/combined_raw_data.csv', index=False)
```

A4: Code in R

Machine Learning with R

Baseline - Logistic Regression - CART - Random Forest

1) The first step is import the packages that we are going to use:

In [ ]:
```
library(tidyverse)
library(rpart)
library(rpart.plot)
library(caret)
library(caTools)
library(randomForest)
library(gbm)
library(plyr)
library(dplyr)
library(ggplot2)
library(GGally)
library(MASS)
library(ROCit)
library(PRROC)
library(RColorBrewer)
```

2) Import the dataset:

We import the dataset and name it 'usopenData' using the read.csv() function:

In [ ]:
```
usopenData <- read.csv('../input/usopendata/USOpen_matches_with_feature - USOpen_matches_with_feature.csv')
```

Let's take a look at the data:

In [ ]:
```
summary(usopenData)
nrow(usopenData)
ncol(usopenData)
```

As we can see, we have 1181 records and 72 features.

3) Data Manipulation:

Now, we transform the 'outcome' variable (that we are trying to predict) from a 0-1 variable to a factor variable. Otherwise CART may try it as regression.

In [ ]:
```
usopenData$outcome <- as.factor(usopenData$outcome)
```

In [ ]:
```
usopenData2 <- usopenData[, -c(0:10)]
usopenData <- usopenData2[, -c(2:41)]
summary(usopenData)
```

4) Split training data and test data:

The function 'sample.split' splits the dataset smartly for binary outcomes:
It keeps the same ratio of 1s and 0s in the training and test sets.

SplitRatio = 0.7 means that we will put 70% of the data in the training set, 30% in the test set.

In [ ]:
```
set.seed(737)
split <- sample.split(usopenData$outcome, SplitRatio = 0.7)
training <- filter(usopenData, split == TRUE)
test <- filter(usopenData, split == FALSE)
```

Let's see the results:

In [ ]:
```
table(training$outcome)
table(test$outcome)
```

5) Create a Baseline:

Based on the results, we see that there are more 0s than 1s. Therefore, our baseline is a model that considers all of them 0s:

In [ ]:
```
#FOR THE TEST DATASET:
accuracyBaseline <- 255/(255+99)
accuracyBaseline
```

So, we have a dummy model with 72.03% of accuracy. We will try to build a CART model to improve that accuracy.

6) Machine Learning Models:

ML1) Logistic Regression Model:

We first run it with all variables:

In [ ]:
```
modelGLM1 <- glm(outcome ~ ., data = training, family="binomial")
summary(modelGLM1)
```

For the second model we just keep variables that show signf.:
diff_match_win_percent_hard
diff_close_sets_percent_hard
diff_match_win_percent_52

In [ ]:
```
modelGLM2 <- glm(outcome ~diff_match_win_percent_hard+diff_close_sets_percent_hard +diff_match_win_percent_52 , data = training, family="bino
summary(modelGLM2)
```

Performance:

In [ ]:
```
predlog<-predict(modelGLM2, type = 'response', newdata= training)
```

In [ ]:
```
ROCobj<-rocit(score=predlog,class=training$outcome)
plot(ROCobj)
```

In [ ]:
```
ROCobj$TPR[which.max(ROCobj$TPR-ROCobj$FPR)]
ROCobj$FPR[which.max(ROCobj$TPR-ROCobj$FPR)]
threshold <- ROCobj$Cutoff[which.max(ROCobj$TPR-ROCobj$FPR)]
```

In [ ]:
```
pred.logistic <- predict(modelGLM2, type='response', newdata = test)

logtable = table(test$outcome,pred.logistic> threshold)
logtable
```

In [ ]:
```
tableAccuracy <- function(test, pred) {
  t = table(test, pred)
  a = sum(diag(t))/length(test)
  return(a)
}


tableAccuracy(test$outcome, pred.logistic> threshold)
##FPR
FPR = logtable[1,2]/(logtable[1,1]+logtable[2,2])
FPR
TPR = logtable[2,2]/(logtable[2,1]+logtable[2,2])
TPR
```

In [ ]:
```
ROCobj <- rocit(score=pred.logistic,class=test$outcome)
plot(ROCobj)
```

In [ ]:
```
ROCobj$AUC
```

In [ ]:
```
accuracyGLM <- tableAccuracy(test$outcome, pred.logistic> threshold)
print(accuracyGLM)
```

ML2) CART model

For this model:

1. We are going to do Cross Validation (10 fold).

In [ ]:
```
modelCARTcv <- train(outcome ~.,
                     data = training,
                     method = "rpart",
                     tuneGrid = data.frame(cp=seq(0, 0.05, 0.002)),
                     trControl = trainControl(method="cv", number=10))
modelCARTcv
```

In [ ]:
```
ggplot(modelCARTcv$results, aes(x=cp, y=Accuracy)) + geom_point(size=3) +
  xlab("Complexity Parameter (cp)") + geom_line()
```

In [ ]:
```
modelCARTcv$bestTune
```

In [ ]:
```
modelCART <- modelCARTcv$finalModel
```

Performance:

In [ ]:
```
predCART <- predict(modelCART, newdata = test, type = "class")
accuracyCART <- sum(diag(table(test$outcome, predCART)))/sum(table(test$outcome, predCART))
accuracyCART
```

ML3) Random Forest Model:

For this model, we perform 5-fold CV:

In [ ]:
```
modelRFcv <- train(outcome ~ .,
            data = training, method="rf",
            tuneGrid = data.frame(mtry=1:21),
            trControl = trainControl(method="cv", number=5, verboseIter=TRUE),
            metric="Accuracy")
```

In [ ]:
```
modelRFcv$results
```

Final model:

In [ ]:
```
modelRF <- modelRFcv$finalModel
modelRF
```

Variable importance:

In [ ]:
```
rfImp <- varImp(modelRF, scale = FALSE)
rfImp
```

In [ ]:
```
varImpPlot(modelRF)
```

Performance:

In [ ]:
```
predRF <- predict(modelRF, newdata=test)
```

In [ ]:
```
accuracyRF <- sum(diag(table(test$outcome, predRF)))/sum(table(test$outcome, predRF))
accuracyRF
```

7) Accuracy Comparisson:

In [ ]:
```
Accuracies <- list("Baseline" = accuracyBaseline, "Logistic_R" = accuracyGLM, "CART" = accuracyCART, "Random_F" = accuracyRF)
barplot(height=unlist(Accuracies),
        col=brewer.pal(4, "Set1"),
        xlab="Models",
        ylab="Accuracy",
        ylim=c(0,0.8))
```

A5: Code Neural Network

(Certain Pages, i.e Compete “Head List” and “Epochs” we excluded to save space)

Neural_Network_CleanedUp

May 14, 2020

Neural Network

[29]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from utils.create_features_utils_USOpen import *
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras import models, layers
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.models import load_model
sns.set_style("darkgrid")
```

[30]:
```
# Dataset import
df = pd.read_csv('USOpen_matches_with_feature.csv')

# Drop N/A Values
df = df.dropna()

# Return 10 rows of data
df.head(10)
```

[30]:

	Tournament	Date	Surface	Round	player_0	player_0_rank	\
0	US Open	2010/08/30	Hard	1st Round	Cilic M.	13.0	
1	US Open	2010/08/30	Hard	1st Round	Davydenko N.	6.0	
2	US Open	2010/08/30	Hard	1st Round	Ferrero J.C.	24.0	
3	US Open	2010/08/30	Hard	1st Round	Gasquet R.	38.0	
4	US Open	2010/08/30	Hard	1st Round	De Bakker T.	48.0	
5	US Open	2010/08/30	Hard	1st Round	Mello R.	81.0	
6	US Open	2010/08/30	Hard	1st Round	Soderling R.	5.0	
7	US Open	2010/08/30	Hard	1st Round	Monfils G.	19.0	
8	US Open	2010/08/30	Hard	1st Round	Zeballos H.	61.0	
9	US Open	2010/08/30	Hard	1st Round	Melzer J.	15.0	

```
   diff_match_win_percent_hh  diff_games_win_percent_hh  \
0                        0.0                   0.000000
1                        0.0                   0.000000
2                        0.0                   0.000000
3                        0.0                   0.090909
4                       -1.0                  -0.183673
5                        0.0                   0.000000
6                        0.0                   0.000000
7                        1.0                   0.130435
8                        0.0                   0.000000
9                       -1.0                  -0.076923

   diff_match_win_percent_hard_hh  diff_games_win_percent_hard_hh
0                             0.0                        0.000000
1                             0.0                        0.000000
2                             0.0                        0.000000
3                             0.0                        0.090909
4                            -1.0                       -0.183673
5                             0.0                        0.000000
6                             0.0                        0.000000
7                             1.0                        0.130435
8                             0.0                        0.000000
9                            -1.0                       -0.076923

[10 rows x 71 columns]
```

[31]:
```
# Create Diff_Rank Variable
df['diff_rank'] = df['player_0_rank'] - df['player_1_rank']

# List all the features
features_list = [
 'diff_rank',
 'diff_match_win_percent',
 'diff_games_win_percent',
 'diff_5_set_match_win_percent',
 'diff_close_sets_percent',
 'diff_match_win_percent_hard',
 'diff_games_win_percent_hard',
 'diff_5_set_match_win_percent_hard',
 'diff_close_sets_percent_hard',
 'diff_match_win_percent_52',
 'diff_games_win_percent_52',
 'diff_5_set_match_win_percent_52',
 'diff_close_sets_percent_52',
 'diff_match_win_percent_hard_60',
 'diff_games_win_percent_hard_60',
```

```
 'diff_5_set_match_win_percent_hard_60',
 'diff_close_sets_percent_hard_60',
 'diff_match_win_percent_hh',
 'diff_games_win_percent_hh',
 'diff_match_win_percent_hard_hh',
 'diff_games_win_percent_hard_hh']
```

[32]:
```
target = df.outcome
features = df[features_list]

train_features, test_features, train_target, test_target =␣
 ↪train_test_split(features, target, test_size=0.20, random_state=1)
```

[33]:
```
# Building Neural network
network = models.Sequential()
network.add(layers.Dense(units=64, activation='relu', input_shape=(len(features.
 ↪columns),)))
network.add(layers.Dense(units=32, activation='relu'))
network.add(layers.Dense(units=1, activation='sigmoid'))
network.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=500)
mc = ModelCheckpoint('best_model.h5', monitor='val_loss', mode='min',␣
 ↪verbose=1, save_best_only=True)

history = network.fit(train_features, train_target,
            epochs=1000, verbose=0, batch_size=128,
            validation_data=(test_features, test_target), callbacks=[es, mc])

saved_model = load_model('best_model.h5')
```

```
Epoch 00001: val_loss improved from inf to 0.62195, saving model to
best_model.h5

Epoch 00002: val_loss did not improve from 0.62195

Epoch 00003: val_loss improved from 0.62195 to 0.61921, saving model to
best_model.h5

Epoch 00004: val_loss improved from 0.61921 to 0.61167, saving model to
best_model.h5

Epoch 00005: val_loss improved from 0.61167 to 0.60397, saving model to
best_model.h5
```

```
Epoch 00550: val_loss did not improve from 0.52105

Epoch 00551: val_loss did not improve from 0.52105

Epoch 00552: val_loss did not improve from 0.52105

Epoch 00553: val_loss did not improve from 0.52105

Epoch 00554: val_loss did not improve from 0.52105

Epoch 00555: val_loss did not improve from 0.52105

Epoch 00556: val_loss did not improve from 0.52105

Epoch 00557: val_loss did not improve from 0.52105

Epoch 00558: val_loss did not improve from 0.52105

Epoch 00559: val_loss did not improve from 0.52105

Epoch 00560: val_loss did not improve from 0.52105

Epoch 00561: val_loss did not improve from 0.52105

Epoch 00562: val_loss did not improve from 0.52105

Epoch 00563: val_loss did not improve from 0.52105

Epoch 00564: val_loss did not improve from 0.52105
Epoch 00564: early stopping
```

[34]:
```
# Accuracy of Best Model

_, train_acc = saved_model.evaluate(train_features, train_target, verbose=0)
_, test_acc = saved_model.evaluate(test_features, test_target, verbose=0)

print('Train Accuracy: %.3f, Test Accuracy: %.3f' % (train_acc, test_acc))
```

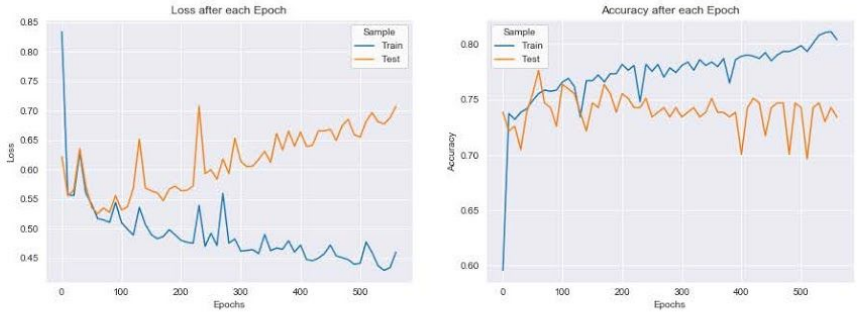
```
Train Accuracy: 0.758, Test Accuracy: 0.776
```

[35]:
```
plt.subplots(figsize=(15, 5))
plt.subplot(1, 2, 1)
plt.title('Loss after each Epoch')
plt.plot(history.epoch[::10], history.history['loss'][::10], label='Train')
plt.plot(history.epoch[::10], history.history['val_loss'][::10], label='Test')
```

```
plt.legend(['Train', 'Test'],loc='upper right', title='Sample',␣
 ↪facecolor='white',fancybox=True)
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.subplot(1, 2, 2)
plt.title('Accuracy after each Epoch')
plt.plot(history.epoch[::10], history.history['accuracy'][::10], label='Train')
plt.plot(history.epoch[::10], history.history['val_accuracy'][::10],␣
 ↪label='Test')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left', title='Sample',␣
 ↪facecolor='white', fancybox=True)

plt.savefig('Results.jpg', quality=100)
```



A6: Code PCA

0.1 Import Packages

[0]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
```

0.2 Import Data

[0]:
```
from google.colab import drive
drive.mount('/content/drive')
```

[0]:
```
data = pd.read_csv('/content/drive/My Drive/242Project/Data/
 ↪USOpen_matches_with_feature.csv')
```

[0]:
```
data.head()
```

0.3 Read Data

[0]:
```
data.player_0_rank = data.player_0_rank.astype(int)
data.player_1_rank = data.player_1_rank.astype(int)
```

[0]:
```
label_encoder = LabelEncoder()
list0 = np.unique(np.concatenate((data['player_0'].unique(),data['player_1'].
 ↪unique())))
label_encoder.fit(list0)
data['player_0']=label_encoder.transform(data['player_0'])
data['player_1']=label_encoder.transform(data['player_1'])
```

[0]:
```
data.head()
```

[0]:
```
data.info()
```

[0]:
```
data.columns
```

0.4 Selecting the columns

[0]:
```
selected_columns = [ 'diff_match_win_percent',
       'diff_games_win_percent', 'diff_5_set_match_win_percent',
       'diff_close_sets_percent', 'diff_match_win_percent_hard',
       'diff_games_win_percent_hard', 'diff_5_set_match_win_percent_hard',
       'diff_close_sets_percent_hard', 'diff_match_win_percent_52',
       'diff_games_win_percent_52', 'diff_5_set_match_win_percent_52',
       'diff_close_sets_percent_52', 'diff_match_win_percent_hard_60',
       'diff_games_win_percent_hard_60',
       'diff_5_set_match_win_percent_hard_60',
       'diff_close_sets_percent_hard_60', 'diff_match_win_percent_hh',
       'diff_games_win_percent_hh', 'diff_match_win_percent_hard_hh',
       'diff_games_win_percent_hard_hh']
print(len(selected_columns))
```

0.5 Split Data

[0]:
```
# test_size: what proportion of original data is used for test set
train_set, test_set, train_target, test_target =␣
 ↪train_test_split(data[selected_columns], data.outcome, test_size=0.2,␣
 ↪random_state=0)
```

0.6 Standardize Data

[0]:
```
scaler = StandardScaler()
# Fit on training set only.
scaler.fit(train_set)
# Apply transform to both the training set and the test set.
train_set = scaler.transform(train_set)
test_set = scaler.transform(test_set)
```

0.7 Apply PCA

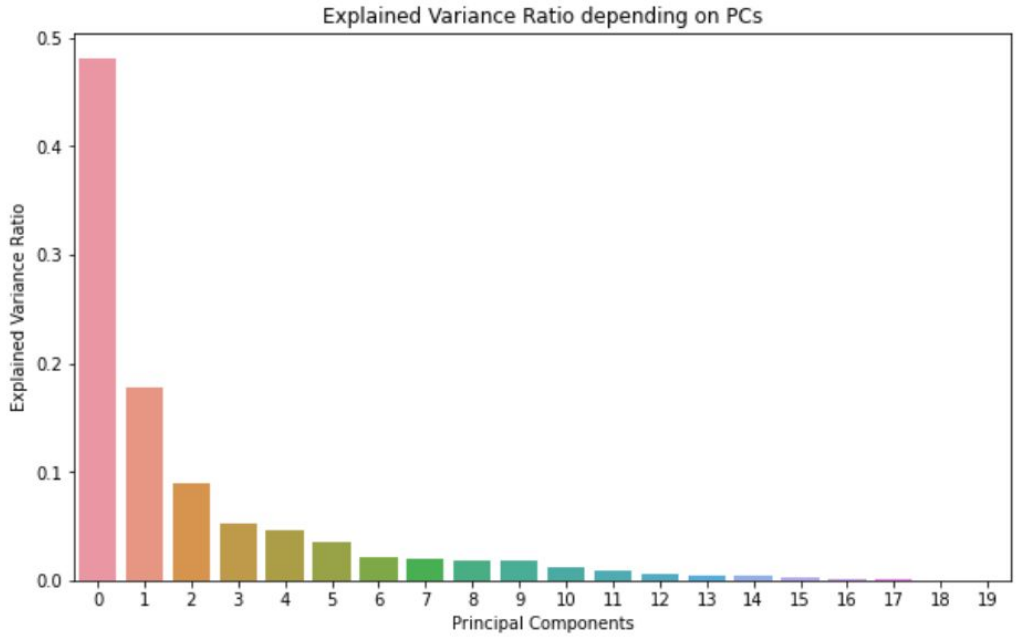
[15]:
```
# Make an instance of the Model
pca = PCA()
pca.fit(train_set)
```

[15]:
```
PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

[16]:
```
pca.explained_variance_ratio_
```

[16]:
```
array([4.80495386e-01, 1.76813743e-01, 8.95178438e-02, 5.15424916e-02,
       4.58591188e-02, 3.47494875e-02, 2.13593704e-02, 2.01510621e-02,
       1.88282620e-02, 1.78855966e-02, 1.22948018e-02, 8.83315753e-03,
       6.61041957e-03, 4.72132282e-03, 4.31717301e-03, 3.22646389e-03,
       1.93316792e-03, 8.61132167e-04, 2.30197534e-32, 6.28816479e-34])
```

[17]:
```
plt.figure(figsize=(10,6))
sns.barplot([i for i in range(20)],pca.explained_variance_ratio_)
plt.title('Explained Variance Ratio depending on PCs')
plt.ylabel('Explained Variance Ratio')
plt.xlabel('Principal Components')
plt.show()
```



[0]:
```
pca = PCA(3)
pca.fit(train_set)
train_set = pca.transform(train_set)
test_set = pca.transform(test_set)
```

[19]:
```
print('Number of selected components:', pca.n_components_)
print(pca.explained_variance_ratio_)
print('Explained variance ratio:',pca.explained_variance_ratio_)
```

```
Number of selected components: 3
```

```
[0.48049539 0.17681374 0.08951784]
Explained variance ratio: [0.48049539 0.17681374 0.08951784]
```

0.8 Apply Logistic Regression to the transformed data

[23]:
```
logisticRegr = LogisticRegression(solver = 'liblinear')
logisticRegr.fit(train_set, train_target)
predictions = logisticRegr.predict(test_set)
test_target = np.array(test_target)
(predictions==test_target).sum()/len(predictions)
```

[23]:
```
0.7890295358649789
```