Probability Analysis with Web Scraping and Linear Regression

Javier A. García Matos
Master of Engineering in Computer Engineering
Dr. Nelliud Torres
Electrical and Computer Engineering and Computer Science Department
Polytechnic University of Puerto Rico

Abstract *— Since the dawn of sports, probability and trends have played an important role in predicting the outcome of an event. It gives the public a general idea of how the match will occur, providing a powerful tool in analytics. This project intends to extract raw tennis match data from verified sources and apply mathematical equations to predict the probability of the outcome of a particular match. Firstly, with the help of the programming language Python and a popular technique known as* web scraping*, the data can be extracted from a verified source, such as the Association of Tennis Professionals (ATP), and validated. After the data is extracted, in this project, three different algorithms will be applied, with the goal of predicting the outcome of a particular tennis match. These algorithms are known as* linear regression *(decision tree, ridge, and lasso) and are made with the programming language Python. There will be a section describing each equation in detail and how it works. Finally, the results will show which of the equations best predicted the outcome of the different matches and conclusions will be drawn based on the results of each equation.*

Key Terms *— ATP, linear regression, Python, web scraping*

INTRODUCTION

Since the sixteenth century, probability has played a large role in the world as we know it. From complex mathematical equations to a simple roll of the dice, the chance of an event occurring is present in everyday life. The world of tennis is not exempt from this. In every match, the odds are stacked against or for a player, depending on a variety of factors. In this project, we will gather the raw data from verified sources with web scraping and apply linear regression to understand the probabilities of a tennis match.

Basic Rules of Tennis

To investigate the probabilities of a tennis match, some basic rules of the sport must be explained. First, the sport is played by either two or four players, with the same number on each team, that is, one player against another or two players against two. Players use rackets to hit the ball back and forth across the court. The points mainly come from three situations: when the ball gets stuck in the net, goes out of bounds, or bounces twice on the court; or when the opposing player does not hit the ball accordingly. The game is organized into sets, which are made up of games. Matches are typically played as a best-of-three or best-of-five sets, with each set consisting of games and potentially tiebreakers (different format games to decide the result of a set or match). Each set has a max total of ten games with the possibility of extending the games to 12 with a famous tiebreaker. In this setup, each game has a scoring pattern of 15, 30, 40, and 60 points. When a game gets to a tie at 40-40, a player needs to win by two points in a row.

The court has two parts marked by white lines. These lines show where the game is played and where the serve happens (figure 1). There is a net in the middle of the court that sets the lowest point the ball must reach to go to the other side.

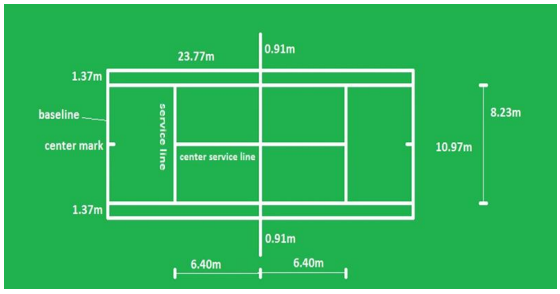

Figure 1
Regulation Tennis Court [1]

Motivation

From the genesis of tennis on July 9th, 1877, at the Wimbledon Courts (figure 2), the word *upset* (a match result that did not go according to prediction) has been used in the sport. For example, a lower-seed player in a tournament defeats the top-seed of the same tournament. Bearing this in mind, with technology constantly improving and changing, newer terms such as *machine learning* can be applied to this world, with the goal to improve the overall accuracy of tennis prediction models and not to avoid upsets, but to know when they will happen. This project will focus on gathering data about tennis players with modern tools such as web scraping and then teaching the computer how to correctly predict a match via machine learning algorithms (ridge, lasso, and decision tree). This article will focus on an in-depth explanation of the implementation of such tools and the results of the different models of prediction. The result will be a very educated and informed prediction or, in simpler terms, a guess.

Figure 1
First Tennis Match [2]

PROBLEM STATEMENT

With modern-day tennis and a variety of factors, the prediction of the outcome of a particular match has grown to be more difficult as time progresses. With newer techniques, players are surprising the outcome of tennis matches around the world. Tennis matches are one of the most difficult sports to predict in the world; with factors, such as surface, player attributes, head-to-head scores, and tournament format, among others, the outcome of a match is influenced severely [3]. Consequently, a more effective solution to properly predict a match is needed. With this goal in mind, this project will concentrate on using those factors to produce an accurate prediction.

Web Scraping with Python

To correctly predict a tennis match using linear regression, historical data must be extracted from a verified source. Firstly, we shall define the difference between structured data and unstructured data. Structured data refers to data that follows a specific format, such as database entities or Excel files. Unstructured data refers to data that does not follow any specific format, such as videos and audio files. The data that will be extracted in this project is categorized as structured data.

In this section, the first step is to implement web scraping (figure 3). Like a web API, web scraping provides a tool for interaction with a website or application, yet the main difference is in the approach used. An API is given by the developers of the application, while web scraping is stand-alone, that is, it extracts the data it sees in the front-end of the website. Consequently, with web scraping, tennis match historical data can be extracted based on tournaments or years, among other factors. To implement this, firstly, the verified source is found [4]. Then, the following libraries are imported: Beautiful Soup (made for parsing the HTML), Selenium (web driver to automate the front-end of a page, i.e., interactions), Pandas (a library for handling different data structures in Python) and CSV (tool for dealing with files in CSV or comma-separated format). After this, the different static variables are listed: tournaments and years. Consequently, a loop is made for each tournament and each year, to modify the existing atptour.com URL (Uniform Resource Locator). Then, the web driver is started (an instance of Google Chrome) and the page is parsed looking for the table with the results. After it is found, the data is stored in a variable and then cleaned (eliminate unnecessary characters or text). Finally, the data is exported into a CSV file for analysis. The structure of the data is as follows: Year, Tournament,

Ranking Winner, Winner, Result, Ranking Loser, Loser, Final Score and Comments.

```
tournament_names,tournaments_ids  = ['australian-open','roland-
garros','wimbledon','us-open'],['580','520','540','560']
#Australian,Roland Garros,Wimbledon,US Open
start_year,end_year=2017,2023 #Year Range 2017 to 2022
f = open("out.csv", "w+") #Open Output CSV to append
f.close()
for i in range(start_year,end_year): #Loop through years
    for j in range(len(tournaments_ids)):
        url =
'https://www.atptour.com/en/scores/archive/'+tournament_names[j]+'
/'+tournaments_ids[j]+'/'+str(i)+'/results'
        driver = webdriver.Chrome()
        time.sleep(1) #Pause for one second
        driver.get(url)
        soup = BeautifulSoup(driver.page_source, 'html.parser')
        parent_div = soup.find('div' , {'id':
'scoresResultsContent'}) #Get enclosing container
        if parent_div:
            table = parent_div.find('table') #Find table within
            data = []
            for row in table.find_all('tr'): #Get data from table
                row_data = []
                row_data.append(i)
                row_data.append(tournament_names[j])
                counter=0
                for cell in row.find_all('td'):
                    if counter == 7 : #Score
                        row_data.append(re.sub("\s+", ",",
cell.text.strip())) #Trim and replace unnecessary characters
                    else: #Everything else
                        row_data.append(re.sub("\s+", ",",
cell.text.replace("\n", "").strip())) #Trim and replace
unnecessary characters
                    counter+=1
                data.append(row_data)
            data = [i for i in data if len(i)>2] #Eliminate
headings
            df = pd.DataFrame(data) #Convert to data frame
            df.to_csv('out.csv',mode='a',index=False,
header=False) #Output data extracted
```

Figure 3
Web Scraping Algorithm

Linear Regression

To discuss the implementation and results of the project, we shall describe linear regression and the different types used in this project. Firstly, a linear regression is defined as a prediction of the value of a variable based on the value of another variable [5]. In other words, it evaluates the different attributes (features) a particular value has and then uses the values to predict the result (target) of a particular variable. The word feature is very important in this definition, since it is a synonym for an object's characteristics. For example, in a tennis match, feature columns may be the year a match took place, the tournament that was played, the winner and loser of the match, the duration of the match, etc. These columns help predict the results of a variety of datasets based on patterns found in the characteristics. On the other hand, a target column refers to the result which needs to be predicted. It is the outcome that needs to be guessed based on the feature columns that are provided. In this example, the result of a tennis match, i.e., who wins, is the target column. With these columns, a dynamic relationship of the values is created. The next question that arises is: What happens behind the curtain? In other words, what technique is used to perform such a prediction? The result of a variable is done by advanced mathematics with the help of vectors and slope equations, defined in the next subsection.

Vectors and Slope Equations

A vector is a quantity that has both magnitude and direction [6]. In other words, it is a dynamic value used to define an object. For example, let's use a vector as a position in a mathematical graph. As shown in figure 4, if many vectors are used, a line can be made in the middle of the graph to separate the objects and therefore create a slope equation, i.e., the mathematical equation that describes the line in the middle. The more precise the line is, the more accurate the prediction will be. The result of the formula would be the dependent variable (the prediction) and the formula would be the characteristics (feature columns). Within the graph, the green dots represent a particular classification while the blue dots refer to another classification. If the example of the feature and target columns above is used, a result could be predicted accordingly, where the green dots could represent player 1 winning the match and the blue dots could represent player 1 losing the match.

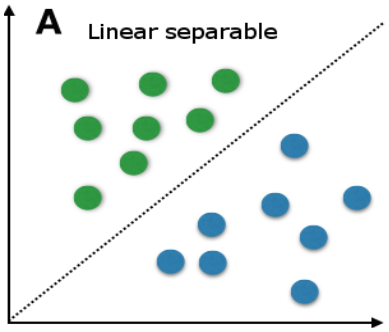

Figure 4
Linear Regression Graph [7]

Decision Tree

As discussed in the previous section, linear regression helps determine the result of a variable based on another variable. As such, this model has different types, which will be covered in this article. Firstly, a decision tree is a type of supervised machine learning used to categorize or make predictions based on how a previous set of questions were answered [8]. In other words, it is an algorithm that uses past values to make a prediction. For example, following the example of the previous section, some questions and answers may be drawn from historical tennis data, such as: Was the tournament Wimbledon? Did they play against each other last year? Did player 1 win against player 2 by many points? As can be observed, the pattern with the questions is the possible answer: yes or no. After each question, a virtual drill down occurs until a prediction is drawn. As shown in figure 5, starting off in the root node, i.e., the first question, it drills down to the "splitting," which can be defined as the *Yes* answer and the *No* answer. After the split occurs, a decision node is created (a sub-question of the root question), which can have a direct prediction (yes or no) or can be split into another decision node. Hence, after the last question in the branch (a subsection of the decision tree) is answered, the prediction is drawn. This prediction is known as the leaf node. This follows the linear regression model, since it functions by predicting an outcome based on the values of other variables. It does it in a way that is indirect. In other words, instead of evaluating the characteristics and their values, it uses questions. For example, it is equivalent to saying: Did player 1 win over player 2 in Wimbledon? To: player 1 wins over player 2 in Wimbledon or player 1 loses to player 2 in Wimbledon.

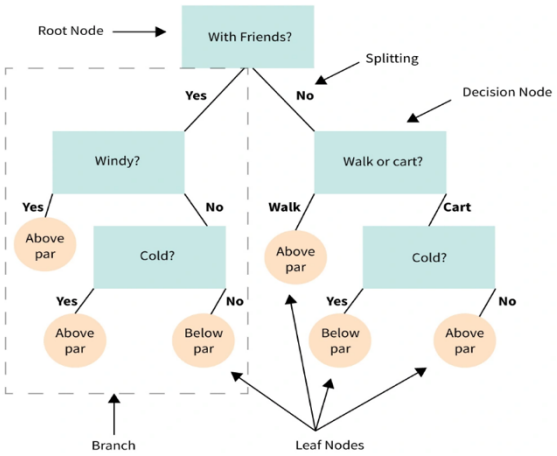

Figure 5 [8]
Decision Tree Graph

Lasso

Having discussed the decision tree in the previous section, another type of linear regression shall be discussed. Lasso, or Least Absolute Shrinkage and Selection Operator, is defined as a popular technique used in statistical modeling and machine learning to estimate the relationships between variables and make predictions [9]. In other words, like linear regression, it establishes virtual relationships between columns or variables and uses it to make an educated guess of the result. Consequently, the process for applying this technique can be divided into steps. In the first step, the traditional linear regression model, an initial prediction is calculated based on the values of the features and target columns as explained in the section of linear regression. Consequently, it does a regularization of the values by eliminating the different repeated values in the data set. After this step, it focuses on shrinking coefficients, which means eliminating the values that are close to zero, since they would negatively affect the prediction of the result. Thus, the values considered for prediction are more common cases instead of the outliers that would affect the result. By eliminating the outliers from the data set, it can correctly make a prediction of a particular result.

Ridge

Finally, the last model in this project is the ridge regression model. A ridge regression algorithm is a model-tuning method that is used to analyze any data that suffers from multicollinearity [10]. In other words, this model applies linear regression and has specific tools to battle data sets with smaller samples and more parameters (features). In this case, since the data set is not large, it is a good model to apply to the test data. The process used by this algorithm follows a step-by-step procedure. Firstly, a regular linear regression model is applied. As discussed in the previous section of this project. After this step, a standardization technique is applied. This part of the process consists of assuring the feature and target values achieve a mathematical mean (average) of 0 and a standard deviation (how much the numbers vary) of 1. With these numbers, a more standard dataset is achieved to be analyzed. Finally, with the data set to a "standard" form, a more accurate prediction can be made.

IMPLEMENTATION

Non-numeric Values

After defining linear regression and its various types, this section will focus on the implementation of linear regression models. Firstly, to correctly apply mathematical equations to a particular data set, it must be "cleaned," that is, it needs to be in the format required by the models for it to function. Consequently, the first step is to convert alpha values (words) into numbers. This is done with a term known as *dictionaries*. Dictionaries are mutable data structures that allow you to store key-value pairs [11], which means that the data in a dictionary is stored with a key (unique identifier) that references a particular value. For example, in the case of tennis players, a dictionary could be the number of the player (key) and the name of the player (value). These keys help give a number that represents the value, allowing for the numeric constraint to be achieved. Figure 6 shows the first function.

```
def is_numeric(value): #Function to determine if value is numeric
    return_value = True #Default value
    try:
        int(value)
    except ValueError:#Cannot convert to integer
        return_value=False#Changes value
    return return_value
def dict_to_csv_keys(dictionary,filename):#Outputs dictionary to
CSV
    f = open(filename+'_keys.csv', "w+") #Open Output CSV to clear
it
    f.close()
    with open(filename+'_keys.csv', 'a') as csv_file:#Open Output
CSV to Append to it
        csv_file.write('Value,Key\n')#Header Line
        for key in dictionary.keys():#Iterates over every key in
the dictionary i.e. unique values

csv_file.write("\'%s\',%s\n"%(key,dictionary[key]))#Detail Line, a
value and its corresponding identifier
def string_to_numeric(data_frame):#Converts string values in a
data frame to identifiers
    for column in data_frame.columns:#For every column
        if not is_numeric(data_frame[column].iat[0]): #Check if
first value is not numeric

unique_values,dictionary=data_frame[column].unique(),{} #Get
distinct values in data frame column and assigns a variable
            for i in range(len(unique_values)): #Counter with
total length of unique values array
                dictionary[unique_values[i]] = i #Assign
identifier to value and value
            dict_to_csv_keys(dictionary,column) #Outputs to CSV
results
            data_frame[column] =
data_frame[column].map(dictionary) #Maps data frame column to
dictionary created
    return data_frame
```

Figure 6
Non-numeric Values Algorithm

A function is a "container" of code for completing a particular job. In other words, it is used to segment the Python script for reusability and organization. The function "is numeric" is a tool for verifying whether a value is numeric. If it is not, it will simply return the value false and if it is a number, it will return the value true. This function is used to determine if the value of a column is already numeric; if it is, then it does not require a dictionary. In the case that the value is alpha (words), the function "string_to_numeric" is used. This function oversees applying dictionaries to a specific column in a data frame. A data frame is a data structure that consists of columns and rows, where each column can be a different data type. In other words, it is a table with different values in it containing values that can be numbers, words, dates, etc. For each column, it will first list the unique values (eliminate the repeated values) and create a dictionary with these values, assigning each value a number or key. After it creates the dictionaries, it exports them into a file

for future reference (keys and values) and assigns the column in the data frame to their respective dictionary (mapping), thus eliminating alpha values from the data frame and finishing the "cleaning" of the data set.

Application of Types of Linear Regression

After the dataset is cleaned, as discussed in the previous section, the different linear regression models can be applied. Firstly, the process begins in the "main" function. This function oversees all the code that is run in the Python script. It starts off with reading the CSV file that was created in the previous section of web scraping (historical tennis match data) and continues to "clean" the dataset (words to numbers). After the dataset is converted to numeric values, the feature columns and target columns are established. This is done with a simple loop that establishes every column except the result that is wished to be predicted as the features columns. On the other hand, it only establishes the result as the target column. With these columns defined, they can be passed on to the "print output" as parameters. A parameter is a value that is supplied to function when it is called. For example, if an "add" function was made, it could accept two numbers as parameters and return their sum. Consequently, the "print output" function begins with the output document "output.csv" being opened to clear any previous results the computer had calculated. After this, the header of the files, i.e., the columns, are written as the first line of the file: Year, Tournament, Ranking Player 1, Player 1, Ranking Player 2, Player 2, Decision Tree Result, Lasso Result, Ridge Result, Lasso Probability of Player 1 Winning, and Ridge Probability of Player 1 Winning. It then opens the testing document, which contains the different results that need to be predicted. Consequently, it loops through these tests and creates an object for each type of linear regression model (figure 7). An object is defined as a data field that has unique attributes and behavior [12]. In other words, it is an instance of the data field. For example, if a student data type were

```
def print_output(feature_columns,target_column):#Prints results to
csv file
    f = open('output.csv', "w+") #Open Output CSV to clear it
    f.close()
    with open('output.csv', 'a') as csv_file:#Open Output CSV to
Append to it
        csv_file.write('Year,Tournament,Ranking Player 1,Player
1,Ranking Player 2,Player 2,Decision Tree Result,Lasso
Result,Ridge Result,Lasso Probability of Player 1 Winning,Ridge
Probability of Player 1 Winning\n')#Header Line
    with open("testing.csv", 'r') as tests:#Open testing file for
reading
        next(tests)#Skip header line
        test = csv.reader(tests)#Initialize reader object
        for row in test:#For each line in csv file
            test_row = [[int(x) for x in row]]#Create test row
            decision_tree = DecisionTreeClassifier() #Initialize
the Decision Tree object
            Lasso_model = Lasso(alpha=1.0) #Initialize the Lasso
object
            ridge_model = Ridge(alpha=1.0) #Initialize the Ridge
object
            decision_tree =
decision_tree.fit(feature_columns.values, target_column)#Assign
corresponding columns
            Lasso_model= lasso_model.fit(feature_columns.values,
target_column)#Assign corresponding columns
            ridge_model= ridge_model.fit(feature_columns.values,
target_column)#Assign corresponding columns

decision_tree_prediction=decision_tree.predict(test_row)#Predict
the result
            lasso_model_prediction =
lasso_model.predict(test_row)#Predict the result
            ridge_model_prediction =
ridge_model.predict(test_row)#Predict the result
            with open('output.csv', 'a') as csv_file:#Open Output
CSV to Append to it
                line=''#Detail Line
                counter=0 #Counter for test values
                for column in feature_columns:#Iteration over
columns that are not the result
                    if os.path.isfile(column+'_keys.csv'):#Check
if column is a Key Based Column
                        df = pd.read_csv(column+'_keys.csv',
quotechar="'")#Read csv file into data frame
                        line+=df[df['Key'] ==
test_row[0][counter]].Value.values[0]+',#Print value
                    else: line+=str(test_row[0][counter])+','#The
opposite
                    counter+=1#Increment counter for test values
                line+='Player 2 Wins,' if
decision_tree_prediction[0] == 0 else 'Player 1 Wins,' #[0]
represents the given parameters would be false, i.e. player 1
loses, [1] The opposite, i.e. player 1 wins
                line+='Player 2 Wins,' if
lasso_model_prediction[0] < 0.51 else 'Player 1 Wins,' #If the
probability is less than 50% then player 2 wins,The opposite, i.e.
player 1 wins
                line+='Player 2 Wins,' if
ridge_model_prediction[0] < 0.51 else 'Player 1 Wins,' #If the
probability is less than 50% then player 2 wins,The opposite, i.e.
player 1 wins
                line+=str("{:.2f}".format(
lasso_model_prediction[0]*100))+','#Probability of player 1
winning
                line+=str("{:.2f}".format(
ridge_model_prediction[0]*100))#Probability of player 1 winning
                csv_file.write(line)#Write Detail Line to file
    print('Output done.')
def main():
    tennis_matches=pd.read_csv('in.csv') #Read Dataset
    tennis_matches=string_to_numeric(tennis_matches) #Clean string
values
    feature_columns = tennis_matches[[i for i in
tennis_matches.columns if i!='Result']] #The columns that affect
the result
    target_column = tennis_matches['Result'] #The result that
needs to be predicted
    print_output(feature_columns,target_column)
```

Figure 7
Linear Regression Algorithm

created, each student would be an object of the data type (an instance). After each object is created, the objects are given the feature and target columns along with their corresponding values (fit function). With these values, the predictions are made with each linear regression model. These values are written to the corresponding document (output.csv). Additionally, with the keys and values document discussed in a previous section, the values are changed for a more user-friendly design. In other words, if the key for the value "Rafael Nadal" were 1, then the 1 is changed for "Rafael Nadal." In the case of the ridge and lasso models, since the output is given in percentage, rules were written to give the output as a specific result. Which means that if the percentage of player 1 winning is greater than 49%, then it registers player 1 winning the match and vice versa.

RESULTS

In this section, particular test cases shall be applied to the project to see how accurate the prediction was. Matches that are not part of the historical data will be considered, that is, tournaments that occurred after the year 2022: Wimbledon, the US Open, Roland Garros, and the Australian Open. Table 1 shows the test cases that were applied.

Table 1
Actual Results of Test Cases

Test #	Winner (Ranking) vs. Loser (Ranking)	Tournament
1	Daniil Medvedev (3) vs. Andrey Rublev (8)	US Open
2	Ben Shelton vs. Tommy Paul (14)	US Open
3	Alexander Zverev (12) vs. Jannik Sinner (6)	US Open
4	Sebastian Korda (29) vs. Hubert Hurkacz (10)	Aus. Open
5	Novak Djokovic (4) vs. Alex de Minaur (22)	Aus. Open
6	Casper Ruud (4) vs. Holger Rune (6)	Roland Garros
7	Stefanos Tsitsipas (5) vs. Sebastian Ofner (Q)	Roland Garros
8	Francisco Cerundolo (23) vs. Taylor Fritz (9)	Roland Garros
9	Carlos Alcaraz (1) vs. Daniil Medvedev (3)	Wimbledon
10	Hubert Hurkacz (17) vs. Lorenzo Musetti (14)	Wimbledon
11	Carlos Alcaraz (1) vs. Novak Djokovic (2)	Wimbledon

Table 2 shows the results of the different algorithms with their respective test cases.

Table 2
Prediction Results of Test Cases

Test #	Decision Tree	Lasso	Ridge
1	Player 1 Wins	Player 2 Wins	Player 1 Wins
2	Player 1 Wins	Player 2 Wins	Player 2 Wins
3	Player 2 Wins	Player 1 Wins	Player 1 Wins
4	Player 1 Wins	Player 2 Wins	Player 1 Wins
5	Player 2 Wins	Player 2 Wins	Player 2 Wins
6	Player 2 Wins	Player 1 Wins	Player 1 Wins
7	Player 1 Wins	Player 1 Wins	Player 1 Wins
8	Player 2 Wins	Player 2 Wins	Player 2 Wins
9	Player 1 Wins	Player 2 Wins	Player 2 Wins
10	Player 1 Wins	Player 1 Wins	Player 2 Wins
11	Player 1 Wins	Player 2 Wins	Player 1 Wins

Table 3 shows the percentage of player 1 winning for the models lasso and ridge.

Table 3
Probabilities of Prediction of Test Cases

Test #	Lasso	Ridge
1	49.24%	52.91%
2	43.58%	38.63%
3	55.61%	55.79%
4	48.24%	52.03%
5	50.98%	46.88%
6	58.86%	60.73%
7	53.83%	52.66%
8	44.13%	47.36%
9	42.84%	43.40%
10	51.95%	47.59%
11	48.56%	58.42%

CONCLUSIONS

After concluding the testing for the linear regression implementation, the evaluation of the test cases showed conclusive results. Table 4 shows the accuracy of the results.

Table 4
Conclusions of Test Cases

Model	Total Tests	Correct Tests	Accuracy %
Decision Tree	11	7	63%
Lasso	11	4	36%
Ridge	11	6	55%
Average	11	5.7	51%

The table shows that the algorithms were pretty accurate overall, with the average accuracy of a correct prediction being 51%. The best algorithm, decision tree, showed an impressive prediction rate of 7 out of the 11 test cases produced. On the other hand, the ridge algorithm demonstrated to be close in the race for the best overall prediction, with an accuracy of 55%. In other words, 6 out of the 11 test cases were correct. In sharp contrast, the last algorithm (lasso) showed a decline in prediction accuracy, with 36%, thus only predicting correctly 4 out of the 11 test cases. With these results, it can be confirmed that, with the implementation of these algorithms, a tennis match may be more accurately predicted. The sample test cases were produced with different players, tournaments, and rankings, demonstrating that the python script can adapt to different scenarios.

In conclusion, the project of web scraping and prediction with linear regression was an overall success, providing an overall accuracy of 51% and a useful tool for data gathering and collection. This project provides a useful tool for analysts, fans, and any person interested in the world of tennis, allowing the worlds of programming and sports to join forces and focus on producing more accurate predictions. In other words, this project represents the complete extinction of tennis upsets.

FUTURE WORK

Soon, many additions can be made to this project, specifically in two major areas: more features and other areas of implementation. Firstly, the more features collected via the web scraping algorithm, the more accurate the results will be. Columns such as surface of play, current tournament layout, age of players, climate, etc., can be added Additionally, this project could be implemented in other areas in which prediction plays a crucial role. For example, in the world of firewalls and cybersecurity, these algorithms can be applied to historical malware attacks to help in the prevention of attacks based on the prediction they will occur, therefore allowing security staff to stop these attacks.

REFERENCES

[1] M. Schnur, “What size is a regulation tennis court?", Metro League, September 5, 2022. Available: https://www.metroleague.org/what-size-is-a-regulation-tennis-court/

[2] History of Tennis, “The first official tennis match,” September 17, 2017. Available: https://thehistoryoftennisblog.wordpress.com/2017/09/21/introduction/

[3] R. Vora, “How to predict a tennis match?”, MatchStat, June 29, 2023. Available: https://matchstat.com/predictions-tips/how-to-predict-a-a-tennis-match/

[4] ATP Tour, “Homepage.” ATP Tour. Accessed Aug. 20, 2023. Available: https://www.atptour.com/

[5] IBM, “About linear regression.” Accessed Oct. 7, 2023. Available: https://www.ibm.com/topics/linear-regression#:~:text=Resources-,What%20is%20linear%20regression%3F,is%20called%20the%20independent%20variable

[6] “Vector,” *Encyclopædia Britannica*, accessed Oct. 7, 2023. Available: https://www.britannica.com/science/vector-physics

[7] Artificial Intelligence, “Linear Classification." Accessed Oct. 7, 2023. Available: https://leonardoaraujosantos.gitbook.io/artificial-inteligence/machine_learning/supervised_learning/linear_classification

[8] Master's in Data Science with edX, “What is a decision tree." Accessed Oct. 7, 2023. Available: https://www.mastersindatascience.org/learning/machine-learning-algorithms/decision-tree/

[9] D. Kumar, “A complete understanding of LASSO regression,” Great Learning, May 30, 2023. Available: https://www.mygreatlearning.com/blog/understanding-of-lasso-regression/

[10] Great Learning, “What is ridge regression?” Accessed Oct. 7, 2023. Available: https://www.mygreatlearning.com/blog/what-is-ridge-regression/#:~:text=Ridge%20regression%20is%20a%20model,away%20from%20the%20actual%20values

[11] Simplilearn, “What is a dictionary in Python?" Accessed Oct. 7, 2023. Available: https://www.simplilearn.com/dictionary-in-python-article#:~:text=In%20Python%2C%20dictionaries%20are%20mutable,update()%2C%20dict

[12] A. S. Gillis and S. Lewis, “What is object-oriented programming?,” TechTarget, July 2021. Available: https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP#:~:text=An%20object%20can%20be%20defined,and%20actively%20updated%20or%20maintained