‘

Course: CPE 495

Spring 2024

Date May 2, 2024

Final Project

Submitted by: Ifenna Ekwenem

Introduction

The ability to accurately predict a fight in sports has always been difficult. This is because many variables come into play when predicting a fight. All UFC fight statistics are accessible through various websites like UFC Stats.com. Machine learning algorithms can be effectively trained and tested for predicting outcomes using versatile tools like Weka, renowned for its robust capabilities in data analysis and modeling. Exploring the predictive capabilities of machine learning algorithms using tools like Weka is essential for me as it offers insights into the potential applications and limitations of AI in decision-making especially in the field of martial arts. Throughout this endeavor, we aim to gain a deeper understanding of how these algorithms perform in real-world scenarios and their potential impact on various domains. We organized our research goals into three research questions:

1. Can any interesting information be found through statistical analysis of mixed martial data openly available on the web?
2. Are there strong relationships between certain variables in the data and the outcome of the fight?
3. Can machine learning algorithms be trained on data scraped from the web to predict the outcome of fights?

To explore these research questions, we developed web scraping software to extract relevant UFC fight stats from websites. We used the Axios and Cheerio libraries to achieve this in typescript and incorporated this logic in Python using BeautifulSoup. We then compiled the stats into Comma Separated Values (CSV) and ran machine learning experiments on the CSV file to see if the fight stats could be used to predict the outcome of the bout. Our results showed that the

eight algorithms tested had recall rates ranging from 0.65 - to 1.00 — the rules. ZeroR algorithm has a recall rate of 1.00. The meta. Ads had a rate of around 0.53. Overall, we can conclude that prediction algorithms can be accurate on web-scraped statistical data.

Related Works

Douglas Johnson did similar work using logical regression for his predictive analysis. Johnson did this with a similar cross-validation technique: “Specifically, cross validation will help verify our model’s usefulness, and discriminant analysis will be employed to support the model choice. Discriminant analysis attempts to classify data into categories based on independent variables. It is similar to logistic regression in some ways, but uses different methods and requires different assumptions” (Johnson 20). Another author who achieved a similar feat was LP James. This study uses decision tree analysis to reveal key actions for winning in high-level MMA. Ground strikes, takedown accuracy, and strike precision are vital, emphasizing grappling and striking proficiency.

Background

The objective was to develop a machine-learning approach for predicting fight outcomes in Mixed Martial Arts. Typically, data to train an algorithm and uncover patterns is structured as a list of instances with defined features and a corresponding class label. Algorithms using machine learning can be classified into Supervised, Unsupervised, and Semi-Supervised. Unsupervised algorithms cluster similar data without class labels. Supervised algorithms predict unlabeled data using labeled training sets. Semi-supervised algorithms leverage both labeled and unlabeled data for training. We used a combination of unsupervised and supervised learning algorithms to run our experiments while using certain classifiers and validation techniques. We refer to a classifier as a machine learning algorithm trained on a specific benchmark dataset.

For evaluation, we opted for five-fold cross-validation, dividing each benchmark version into five folds. These folds contain stratified random samples, ensuring an equal distribution of class values among them. The recall is a performance metric that assesses the classifier's ability to classify positive training instances. A perfect recall, or true positive rate (TPR) of 1, signifies that all positively labeled instances are correctly classified. The false negative rate (FNR) is the complement of the TPR, representing the probability of misclassifying actual pulsars as non-pulsars, which is highly undesirable. Precision quantifies the fraction of positive classifications that are relevant. A perfect precision of 1 indicates that every predicted positive instance is indeed positive. The false positive rate measures how often the classifier incorrectly labels negative instances. The F-measure is a metric used to evaluate the performance of a classification algorithm. It's the harmonic mean of precision and recall, providing a single score that balances both false positives and false negatives.

Before we could implement and run the machine learning algorithms, the data had to be scraped from the web using the logic of web scraping. Web scraping can be defined as the ability to extract data sets from the internet using code. This was done in two different programming languages to show the replicability of the concept. In JavaScript, it was done using Axios and Cheerios libraries. Axios is a JavaScript library used for making HTTP requests from browser-based and Node.js applications. It provides a simple API for sending asynchronous HTTP requests to REST endpoints and handling responses. Axios supports various features such as promise-based requests, interceptors for request and response transformations, automatic JSON data parsing, and even extracting data by HTML elements.

As for Cheerio, it's a lightweight and fast library for parsing and manipulating HTML content using jQuery-like syntax. Cheerio allows developers to traverse and manipulate HTML documents using familiar jQuery methods, making it easy to extract specific elements, attributes, or text content from web pages.

To replicate this in Python, we utilized the BeautifulSoup library. This Python library is for web scraping and parsing HTML/XML files. It constructs a parse tree from documents, enabling traversal, search, and modification. Navigational methods like .find(), .find_all(), and .select() elements based on tags, attributes, or text Modification options include adding, removing, or altering elements/attributes. It simplifies web scraping, it abstracts HTML parsing complexities.

We wrote the files to the CSV format using CSV.py. It handles operations with CSV (Comma-Separated Values) files in Python. It reads data from CSV files into Python data structures, writes data from these structures back into CSV files, and parses CSV data for easier

manipulation. The library performs tasks such as filtering rows, transforming values, or aggregating data within CSV files.

The Naive Bayes Algorithm, rooted in Bayes' theorem, operates under the assumption of feature independence, making it a simple yet effective probabilistic classifier. Particularly prominent in text classification and sentiment analysis, it's valued for its computational efficiency and ability to handle large datasets with many features. Despite its "naive" assumption, Naive Bayes often performs admirably well, especially in scenarios where independence among features holds reasonably true.

SMO, or Sequential Minimal Optimization, serves as a critical tool for training Support Vector Machines (SVMs), a popular class of supervised learning models utilized for classification and regression tasks. SMO addresses the optimization problem inherent in SVM training, efficiently navigating the complexities of large-scale datasets. By iteratively selecting pairs of variables to optimize, SMO streamlines the process of constructing an optimal hyperplane to separate data points into distinct classes.

The Meta AdaBoost Algorithm extends the capabilities of AdaBoost, a renowned ensemble learning method. AdaBoost, or Adaptive Boosting, combines multiple weak classifiers to form a robust, high-performing model. The "meta" designation suggests a variation or enhancement of AdaBoost, possibly incorporating additional techniques or refining its underlying mechanisms.

Rules Part refers to a rule-based classification approach, where decisions are made based on predefined rules or conditions. These systems are advantageous for their transparency and interpretability, making them suitable for tasks where understanding the decision-making process is essential.

Finally, "Trees and Tree Random Forest" likely alludes to decision tree algorithms and the Random Forest ensemble method, which leverages multiple decision trees for improved accuracy and generalization. Decision trees recursively partition data based on feature values, while Random Forest aggregates the predictions of multiple trees to mitigate overfitting and enhance predictive performance.

Methodology

To successfully web scrape, we wrote logic that parsed through table row elements and returned the data in the HTML elements. After we check for the elements it is stored and written to the CSV file, this is the basic logic for scraping the contents of the page. The web crawling logic was written by getting each UFC event and parsing it through for the fights by checking HTML elements that contain the information we needed, this can be seen in Figures 6 and 7. The repository is linked here and here are some examples. The reason why we did not discuss the Javascript logic in this methodology is that it did not yield the CSV file that was used to train our data. Nonetheless, it is appended here. The data we scraped was a CSV file with 15,120 Entries. This totals out to 7560 fights. It contained: Name, Knockdowns, Significant Strikes Landed, Significant Strikes Attempted, Significant Strike %, Total Strikes Landed, Total Strikes Attempted, Takedowns Landed, Takedowns Attempted, Takedowns Percent, Submission Attempts, REV (Reversals), Control Time, Head Landed, Head Attempted, Body Landed, Body Attempted, Leg Landed, Leg Attempted, Distance Landed, Distance Attempted, Clinch Landed, Clinch Attempted, Ground Landed, Ground Attempted, Winner. This is also contained in the repository.

```
10
11  # Send a GET request to the URL l
12          for url in urls:
13              response = requests.get(url)
14              print(response)
15
16              # Parse the HTML content
17              soup = BeautifulSoup(response.content, "html.parser")
18
19              # Find the table body element & for the Significant Strikes table
20              table_body = soup.find("tbody", class_="b-fight-details__table-body")
21
22              sig_table_body = soup.find("tbody", class_="b-fight-details__table-body")
23
24
25              # Significant Strikes
26
27              head_sig_forfighter1 = soup.select_one(
28                  "body > section > div > div > table > tbody > tr > td:nth-child(4) > p:nth-child(1)"
29              ).get_text(strip=True)
30              head_sig_forfighter2 = soup.select_one(
31                  "body > section > div > div > table > tbody > tr > td:nth-child(4) > p:nth-child(2)"
32              ).get_text(strip=True)
33
```

Figure 1: Scraping the Entire page to check for a table on the page

```
34              body_sig_forfighter1 = soup.select_one(
35                  "body > section > div > div > table > tbody > tr > td:nth-child(5) > p:nth-child(1)"
36              ).get_text(strip=True)
37              body_sig_forfighter2 = soup.select_one(
38                  "body > section > div > div > table > tbody > tr > td:nth-child(5) > p:nth-child(2)"
39              ).get_text(strip=True)
40              leg_sig_forfighter1 = soup.select_one(
41                  "body > section > div > div > table > tbody > tr > td:nth-child(6) > p:nth-child(1)"
42              ).get_text(strip=True)
43              leg_sig_forfighter2 = soup.select_one(
44                  "body > section > div > div > table > tbody > tr > td:nth-child(6) > p:nth-child(2)"
45              ).get_text(strip=True)
46              distance_sig_forfighter1 = soup.select_one(
47                  "body > section > div > div > table > tbody > tr > td:nth-child(7) > p:nth-child(1)"
48              ).get_text(strip=True)
49              distance_sig_forfighter2 = soup.select_one(
50                  "body > section > div > div > table > tbody > tr > td:nth-child(7) > p:nth-child(2)"
51              ).get_text(strip=True)
52              clinch_sig_forfighter1 = soup.select_one(
53                  "body > section > div > div > table > tbody > tr > td:nth-child(8) > p:nth-child(1)"
54              ).get_text(strip=True)
55              clinch_sig_forfighter2 = soup.select_one(
56                  "body > section > div > div > table > tbody > tr > td:nth-child(8) > p:nth-child(2)"
57              ).get_text(strip=True)
58              ground_sig_forfighter1 = soup.select_one(
59                  "body > section > div > div > table > tbody > tr > td:nth-child(9) > p:nth-child(1)"
60              ).get_text(strip=True)
61              ground_sig_forfighter2 = soup.select_one(
62                  "body > section > div > div > table > tbody > tr > td:nth-child(9) > p:nth-child(2)"
63              ).get_text(strip=True)
64              winforfighter1 = soup.select_one(
65                  "body > section > div > div > div.b-fight-details__persons.clearfix > div:nth-child(1) > i"
66              ).get_text(strip=True)
67              winforfighter2 = soup.select_one(
68                  "body > section > div > div > div.b-fight-details__persons.clearfix > div:nth-child(2) > i"
69              ).get_text(strip=True)
70
```

Figure 2: The table is scraped for stats that are contained in the table

```
102
103         # Combine the extracted data
104         fighter1data = [
105             stats[0],  # Name (with double quotes)
106             stats[2],  # Knockdown
107             stats[4].split()[0],  # Fighter 1 Significant Strikes Landed
108             stats[4].split()[2],  # Fighter 1 Significant Strikes Attempted
109             stats[6],  # Significant Strikes %
110             stats[8].split()[0],  # Total Strikes Landed
111             stats[8].split()[2],  # total strikes Attempted
112             stats[10].split()[0],  # Fighter 1 Takedowns
113             stats[10].split()[2],  # take down attempted
114             stats[12],  # TD %
115             stats[14],  # Submission Attempts
116             stats[16],  # REV
117             ctrl_time_seconds_fighter1,  # CTRL TIME
118             head_sig_forfighter1.split()[0],
119             head_sig_forfighter1.split()[2],
120             body_sig_forfighter1.split()[0],
121             body_sig_forfighter1.split()[2],
122             leg_sig_forfighter1.split()[0],
123             leg_sig_forfighter1.split()[2],
124             distance_sig_forfighter1.split()[0],
125             distance_sig_forfighter1.split()[2],
126             clinch_sig_forfighter1.split()[0],
127             clinch_sig_forfighter1.split()[2],
128             ground_sig_forfighter1.split()[0],
129             ground_sig_forfighter1.split()[2],
130             winforfighter1,
131         ]
```

Figure 3: The data is stored separately and split per fighter

```
164
165 ##############################################################################
166 # Write the Header row
167 with open("fight_data.csv", "w", newline="") as csv_file:
168     writer = csv.writer(csv_file)
169
170     # Write the header row
171     header_row = [
172         "Name",
173         "Knockdown",
174         "Significant_Strikes_Landed",
175         "Significant_Strikes_Attempted",
176         "Significant_Strike_%",
177         "Total_Strikes_Landed",
178         "Total_Strikes_Attempted",
179         "Takedowns_Landed",
180         "Takedowns_Attempted",
181         "Takedowns_Percent",
182         "Submission_Attempts",
183         "REV",
184         "Control_Time",
185         "Head_Landed",
186         "Head_Attempted",
187         "Body_Landed",
188         "Body_Attempted",
189         "Leg_Landed",
190         "Leg_Attempted",
191         "Distance_Landed",
192         "Distance_Attempted",
193         "Clinch_Landed",
194         "Clinch_Attempted",
195         "Ground_Landed",
196         "Ground_Attempted",
197         "Winner",
198     ]
199     writer.writerow(header_row)
```

Figure 4: The data is written to the CSV file and the above parameters are the header row

```python
#Scrape the URL for each anchor text http://ufcstats.com/statistics/events/completed?page=all
#for each URL scrape the fight details anchor tags nested in the data link =
# This will generate muultiple URLs that will be inserted in the array of URL
# URL of the webpage containing the table
##########################################################################
event_details = [
    "http://ufcstats.com/event-details/e4a9dbade7c7e1a7"
]

response = requests.get("http://ufcstats.com/statistics/events/completed?page=all")

    # Parse the HTML content
soup = BeautifulSoup(response.content, "html.parser")
anchor_tags = soup.find_all("a")
hrefs = []

for tag in anchor_tags:
    if 'href' in tag.attrs:
        href = tag['href']
        hrefs.append(href)
    else:
        print("Anchor tag without href attribute:", tag)



# Filter hrefs that start with the desired prefix
event_details = [tag['href'] for tag in anchor_tags if tag.has_attr('href') and tag['href'].startswith("http://ufcstats.com/event-

eventScraper(event_details)
```

Figure 5: The logic that allows scraping for event details that contain fights from each UFC event

```python
def eventScraper(event_details):


    for event in event_details:
        response = requests.get(event)

    # Parse the HTML content
        soup = BeautifulSoup(response.content, "html.parser")
        anchor_tags = soup.find_all("a")
        hrefs = []

        for tag in anchor_tags:
            if 'href' in tag.attrs:
                href = tag['href']
                hrefs.append(href)
            else:
                print("Anchor tag without href attribute:", tag)



        # Filter hrefs that start with the desired prefix
        fight_details = [tag['href'] for tag in anchor_tags if tag.has_attr('href') and tag['href'].startswith("http://ufcstats.co

        multiScraper(fight_details)
```

Figure 6: This logic allows us to scrape for events from the page that contains all the stats

Results

Upon completion of our experiments. We were surprised that certain variables like name were a determining factor in predicting the outcome of a fight. The knockdown attribute was the highest ranking by the classifier. This makes sense because when someone is knocked down in the UFC there are no standing counts hence allowing fighters to get finished into a knockout. The ground landed, ground attempted and control time attributes were next with attribute rankings of (0.0673, 0.05771, 0.04311) respectively. This makes sense because the UFC has been dominated mostly by grapplers such as Chael Sonnen, George St Pierre, Khabib Nurmagomedov, Jon Jones, Charles Oliveria, and many others whose primary fight style relies on the ground game. The leg kicks and reversal seemed to have the lowest attribute rankings as seen in figure 7 below. The attribute rankings were obtained using the Gain ratio attribute evaluator.

The F Measure for the algorithms can be seen in Figure 8. The Naive Bayes Algorithm had an F measure of about 0.73, the function SMO about 0.78, the meta ada boost algorithm had (0.6-0.7), and the rules Part had an F measure of (0.6). While the trees and tree random forest had an F measure of about 0.74. Typically, higher F-measure values indicate better overall performance of the algorithm in terms of both precision and recall. The recall rates for the algorithms can be seen in Figure 9. The Naive Bayes Algorithm had a recall of about 0.79, the function SMO about 0.78, the meta ada boost algorithm had (0.5-0.79), and the rules Part had a recall of (0.74). While the trees and tree random forest had a recall of about 0.73. Higher recall values indicate that the algorithm is better at capturing all relevant instances, which is generally desirable in tasks where identifying as many positives as possible is crucial. The accuracy of all the algorithms can be seen in Figure 10. The Naive Bayes Algorithm had an accuracy of about 58, the function SMO about 74, the meta ada boost algorithm had (58-73), and the rules Part had

an accuracy of (74). While the trees and tree random forest had an accuracy of about 76. There are also CSV files for the results that will be appended to the submission.

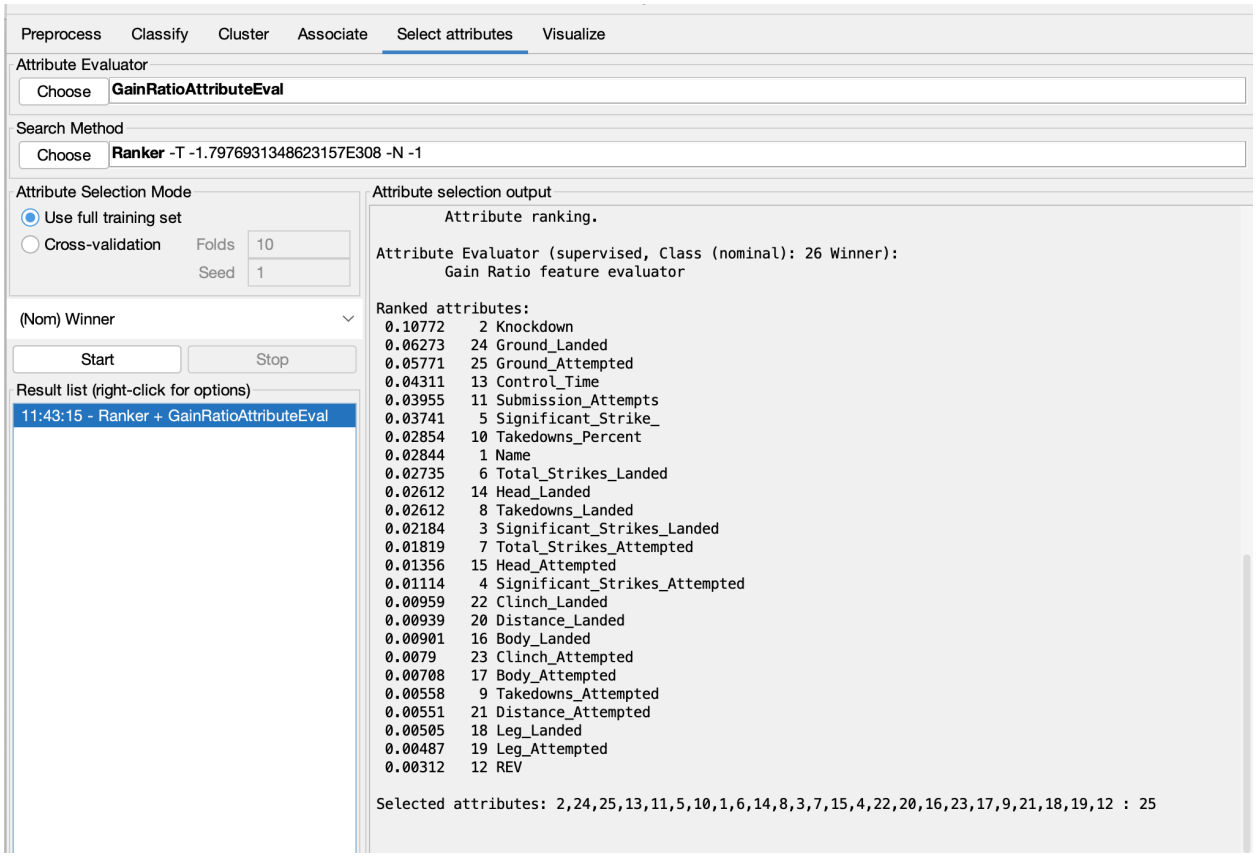

Figure 7: Attribute Rankings

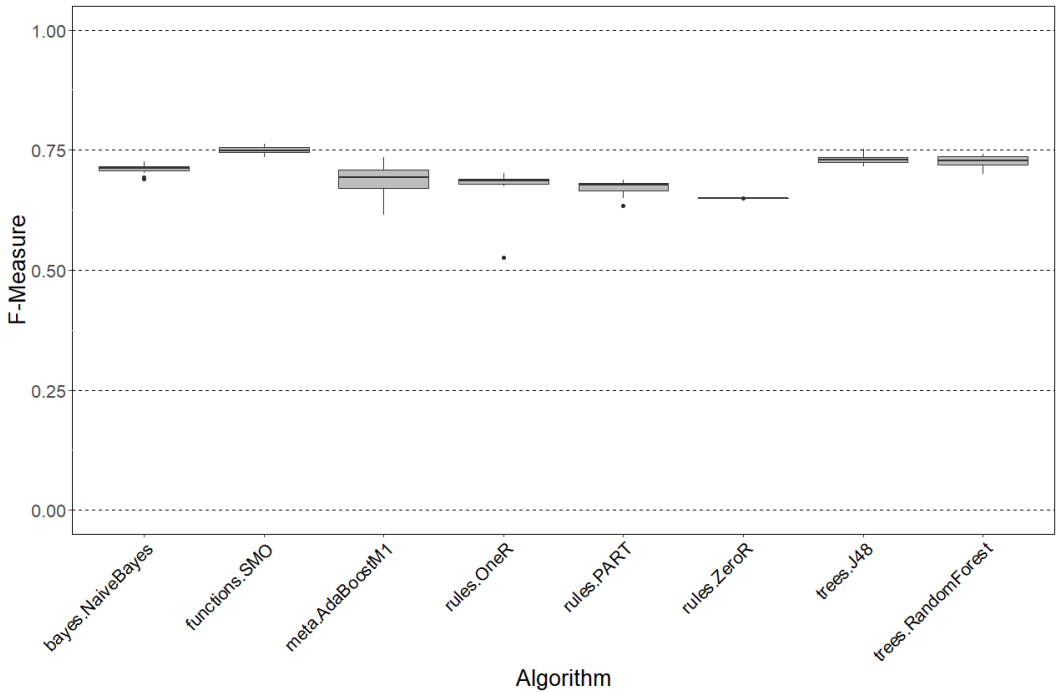

Figure 8: F Measure for Algorithms

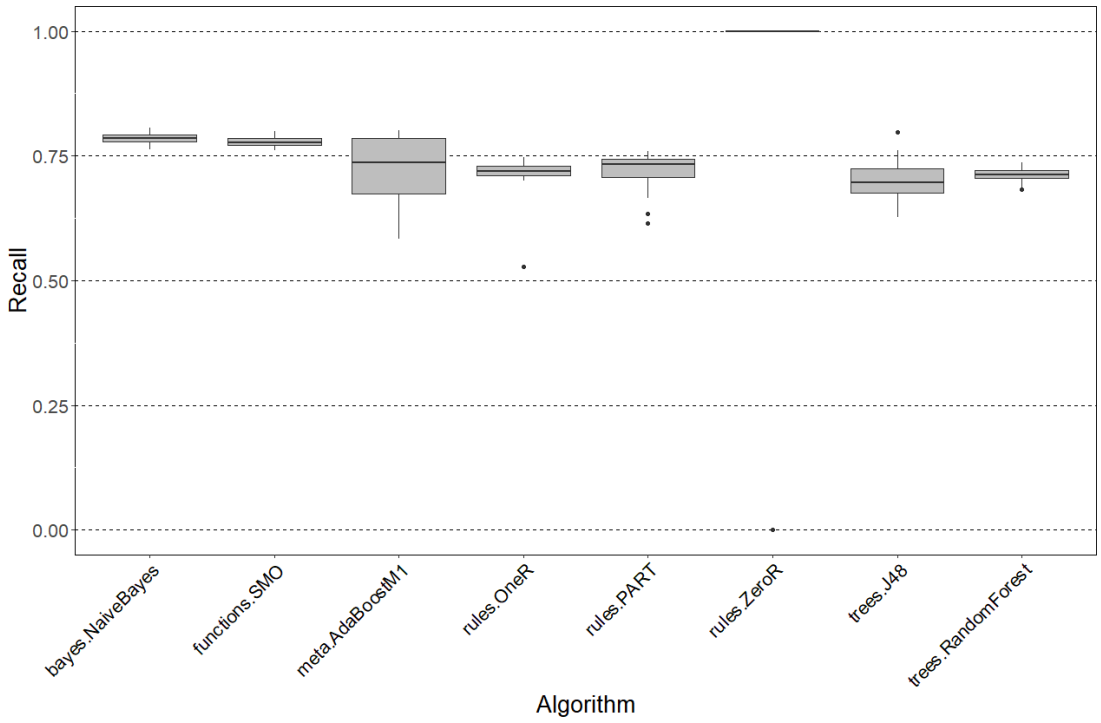

Figure 9: Recall rates of the algorithms

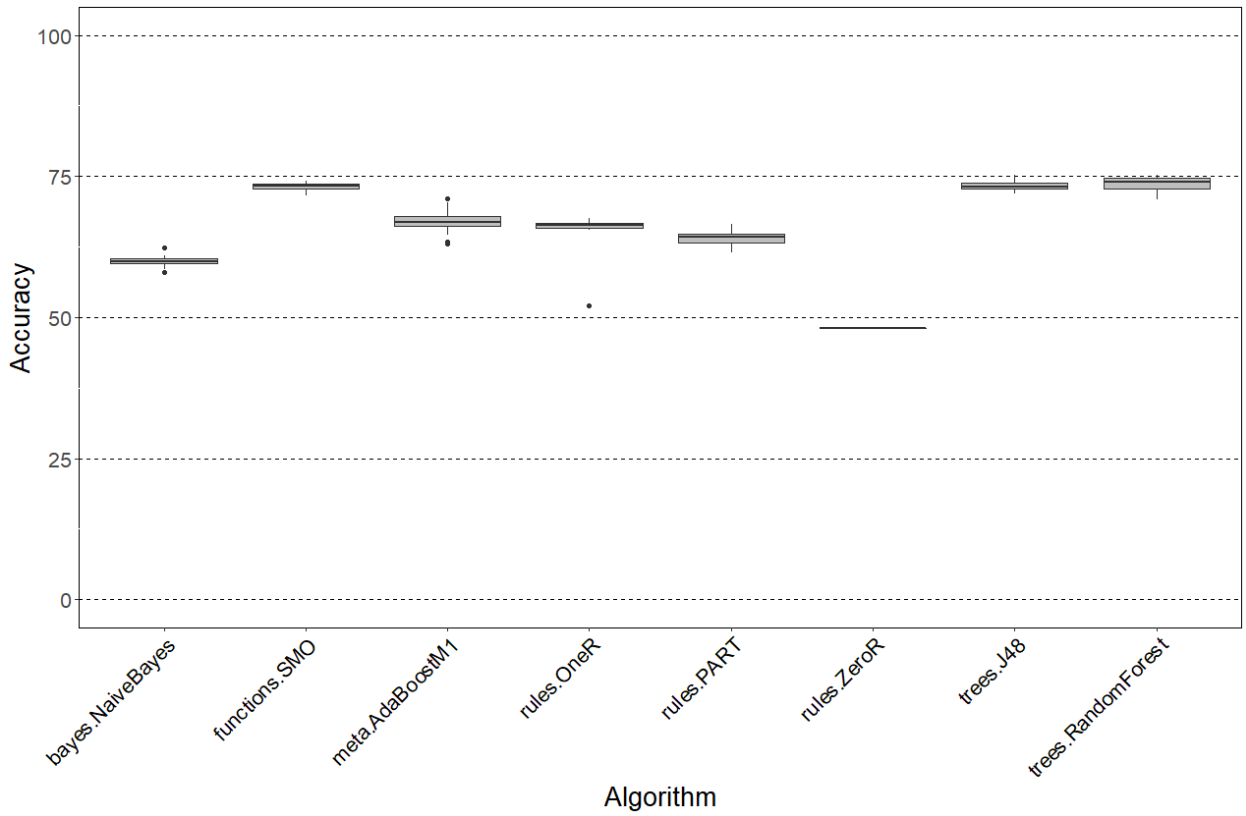

Figure 10: Accuracy of the Algorithms tested

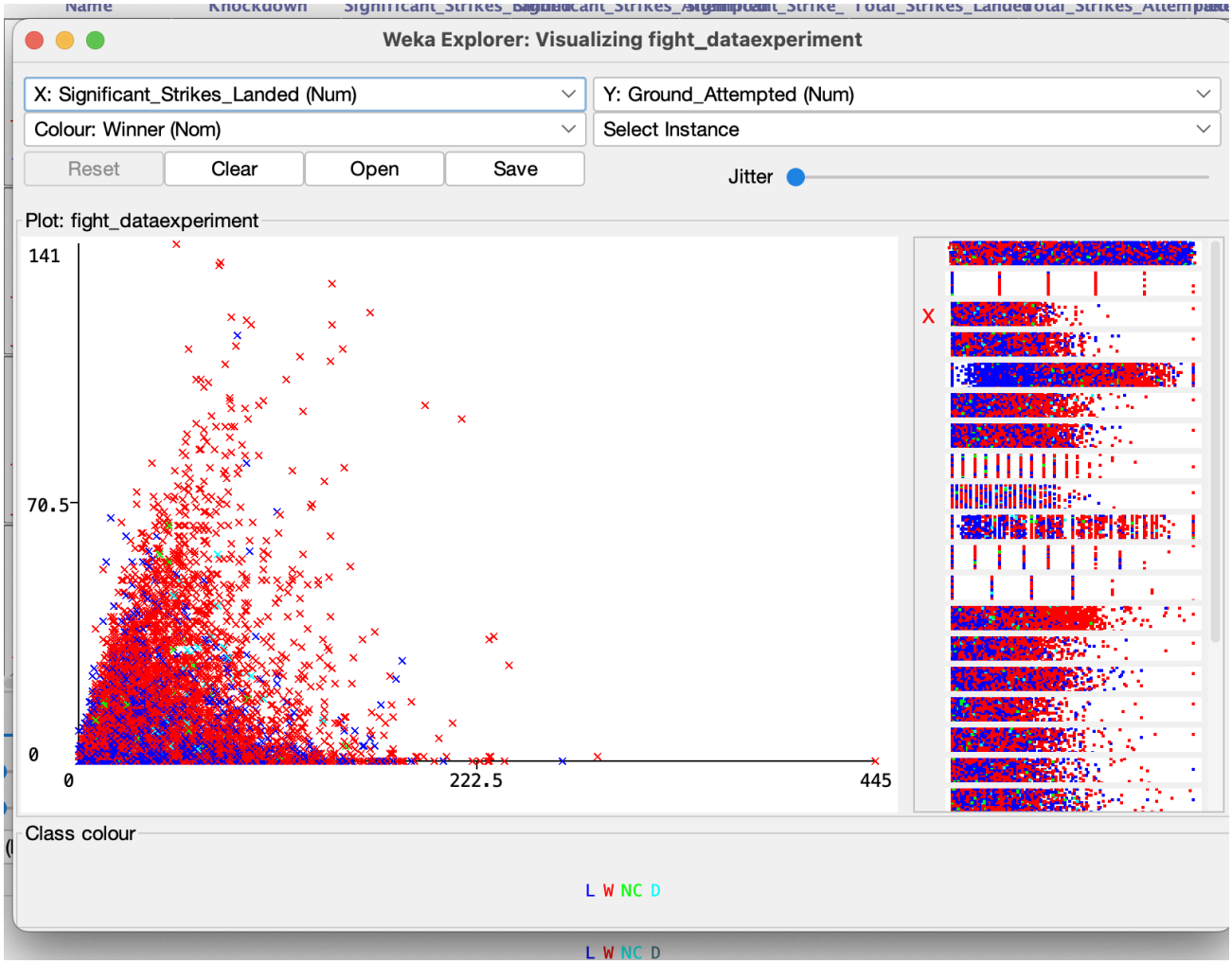

Figure 11: Visualization of the 5 fold, cross-validation using the random trees algorithm for significant strike and ground attempted

Conclusion

In conclusion, we explored the predictive capabilities of machine learning algorithms using UFC fight statistics. This provided practical insights into the realm of machine learning in the world of mixed martial arts. Through web scraping and data analysis, we aimed to address three key research questions: Can any interesting information be found through statistical analysis of mixed martial data openly available on the web? Are there strong relationships between certain variables in the data and the outcome of the fight? Can machine learning algorithms be trained on data scraped from the web to predict the outcome of fights?

Our methodology involved the development of web scraping software to extract relevant UFC fight stats, followed by the utilization of machine learning algorithms to analyze and predict fight outcomes based on the compiled data. Despite encountering challenges such as the perfect recall rate of the ZeroR algorithm, our results demonstrated promising outcomes, with recall rates ranging from 0.65 to 1.00 across different algorithms. We found that prediction algorithms can indeed achieve accuracy when applied to web-scraped statistical data. t

Furthermore, we identified certain variables, such as knockdowns, ground game statistics, and control time, as significant determinants in predicting fight outcomes. This underscores the importance of grappling proficiency and strategic maneuvers in MMA competitions.

While our study contributes to the existing body of research in sports analytics, it is essential to acknowledge its limitations, including the need for further validation and refinement of the prediction models using more data to make better associations such as judge's decision and also factors like weight cuts. Soon we hope to research endeavors that could explore additional variables, refine algorithmic approaches, and incorporate real-time data for enhanced predictive accuracy.

Works Cited

1. Hitkul, et al. "A comparative study of machine learning algorithms for prior prediction of UFC fights." Harmony Search and Nature Inspired Optimization Algorithms: Theory and Applications, ICHSA 2018. Springer Singapore, 2019.
2. Johnson, Jeremiah Douglas. Predicting outcomes of mixed martial arts fights with novel fight variables. Diss. University of Georgia, 2012.
3. James, Lachlan P., et al. "Identifying the performance characteristics of a winning outcome in elite mixed martial arts competition." Journal of science and medicine in sport 20.3 (2017): 296-301.
4. “UFC.” *UFC Stats*, ufcstats.com/statistics/events/completed?page=all. Accessed 29 Apr. 2024.
5. Axios. “Axios/Axios: Promise Based HTTP Client for the Browser and Node.Js.” GitHub, github.com/axios/axios. Accessed 2 May 2024.
6. Cheeriojs. “Cheeriojs/Cheerio: The Fast, Flexible, and Elegant Library for Parsing and Manipulating HTML and XML.” GitHub, github.com/cheeriojs/cheerio. Accessed 2 May 2024.