Emmett Krochock
CS480
2 December 2021

Senior Project Paper for MMA Fighter Odds Calculator

Advisor: John Sarkela. Committee: Dr. Shafei, Dr. Poe

The objective of my senior project was to create an application where the user can generate betting odds for matches between mixed martial arts (MMA) fighters. To achieve this, I compiled fighter data using web-scrapers, ranked the fighters using a rating system, and created betting algorithms that generate odds between hypothetical matches between MMA fighters. My whole project is written in Python, which was the key piece of technology I had to learn for this project.

The three main components of the project are two web-scrapers, the files that rank and analyze the fighters in the database, and the UI where the user can choose fighters from the database and view the betting odds between them.

The first of the two web-scrapers collects the “basic” stats for all the fighters. The basic stats are a fighter’s name, weight, divisions competed in, and a list of all their fights. The web-scraper takes a list of fighter names (where each fighter has a valid Wikipedia page), generates the Wikipedia URL, scrapes the relevant data from the page and stores that data in a dictionary where each fighter’s name is the key. This basic stats dictionary is then written to a JSON file.

To assemble my initial list of fighter names who have a valid Wikipedia page, I used the Wikipedia API. I created a query for the Wikipedia page “List of male mixed martial artists” that gave me all of the hyperlinks on the page in JSON format. This collection of links, though, contained irrelevant links like to a country’s page, or to a fighting organization’s page. After

filtering those irrelevant links, I could manually remove the last few entries that were not pages of MMA fighters. Then, I wrote a script that converted all the untranslated Unicode characters (like “\u00e9”) with their literal characters (“é”).

The web-scraper URL-encodes all the fighter page links, visits each page, and scrapes the data. To parse the Wikipedia pages and pull out the relevant stats, I used the Python library BeautifulSoup4. BeautifulSoup4 takes an HTML page and can easily pull out specific information. I made great use of the find_all(a_tag) method that takes an HTML tag and returns a list of all matching elements. Without having a large knowledge of HTML, I was able to leverage the power of BeautifulSoup4 to code the HTML parsing part of my web-scraper. Making a query using the Wikipedia API, getting all the relevant fighter links, and using those links to collect the basic fighter data was the first thing I did in my project.

The other data that a fighter has is a dictionary of his “detailed stats”. The detailed stats are SLpM (strikes landed per minute), StrAcc (striking accuracy), SApM (strikes absorbed per minute), StrDef (strike defense %), TDAvg (takedowns per 15 minutes), TDAcc (takedown accuracy %), TDDef (takedown defense %), and SubAvg (submissions attempted per 15 minutes). The second web-scraper visits every fighter’s page on UFCstats.com and, like the basic stats web-scraper, uses BeautifulSoup4 to extract the stats from each page. The detailed stats dictionary for each fighter is stored in a dictionary where the fighter name is the key. Then, the entire detailed stats dictionary is written to a JSON file. When the basic web-scraper collects basic data for a fighter, it creates a Fighter object containing the basic and detailed stats of a fighter.

Emmett Krochock
CS480
2 December 2021

The JSON file containing the basic and detailed stats is used to build the fighter object dictionary to be used throughout the rest of the program. A Fighter object contains the name, weight, detailed stats dictionary, and a list of their fights (stored in Fight objects). This fighter object dictionary acts as the database for all the fighters.

My fighter database consisted of 1700 fighters initially. I realized that many of the fighters in my database were not very relevant because they either they hadn't fought in ten years or they had short careers of only two of three fights. I also realized that many active, relevant fighters were not in my database. I expanded my database by first creating a list of all existing fighters in my database and then adding each of their opponents to the list. This list of potential fighters that could be added to my database amounted to 16,000 fighters. To select the relevant fighters from this massive list, I created a simple web-scraper that first checks if a fighter's Wikipedia page exists, and then checks the first paragraph for the words "MMA" or "martial". If a potential fighter has an existing Wikipedia page and their description describes them as an MMA fighter, they are considered a valid fighter. This web-scraper took about six hours to finish running and I ended up with 2,000 total relevant fighters in my database. A use case of this project is to select two fighters that will be competing in an upcoming fight, so is important to have a database that contains fighters that are competing in the present day.

To compute the odds between two fighters, I needed a way to rank the fighters relative to each other. I knew that the Elo ranking system is used commonly for competitive games like chess, so I decided to implement an Elo ranking of the fighters in my database. To compute the rankings of the fighters I created a script that creates a list containing every fight in the

database sorted by chronological order. For every fight, the two fighter's rankings are updated based on their current ranking and the outcome of the match. The default rating for a fighter is 1500 and the range of rankings I ended up with was 1100-2300.

One problem with the Elo rating system is that the time between matches is not relevant. Activity or inactivity over a time period has no effect on a player's ranking. In fighting, inactivity is especially relevant because fighters only compete 2-3 times a year on average. In a year's time, the rankings in a division (the set of all fighters in a weight class) can change drastically. After a year or two of inactivity for a fighter, it can be unclear whether a fighter "deserves" their exact ranking. To remedy this, I switched to using the Glicko2 ranking system. Glicko2 is like Elo, but activity over a ranking period (300 days for example) does have a bearing on a fighter's ranking. To implement Glicko2, I used a Glicko2 Python library. To calculate the Glicko2 for all the fighters, I used my same Elo algorithm but instead evaluated the rankings of the fighters in 300-day intervals (the ranking period) and used the Glicko2 Python class.

My "basic" betting algorithm simply finds the ratio between two fighter's Glicko2 rankings and multiplies it by some factor. The next algorithm I implemented was the basic algorithm * the common opponent advantage. If FighterA lost to FighterC and FighterB beat FighterC, this is certainly relevant when predicting the winner of a fight. To implement the common opponent advantage, I implemented a function that takes two fighters, finds their common opponents, and returns a factor based on which fighter has had more success against their common opponents. The third algorithm I implemented was the previous two algorithms * the stylistic advantage. The detailed stats that a fighter has provides some insight into their

fighting style. A fighter with 11.0 submissions attempted per 15 minutes is most certainly a grappling style of fighter, while a fighter with over 4.0 strikes landed a minute is most likely a striker (likes to throw punches and kicks). The stylistic advantage is determined by taking the total percent difference of all the detailed stats between the two fighters. The four betting algorithms I ended up with were the “basic” algorithm, “basic + common opponent adv.”, “basic + stylistic adv.”, and “basic + common opponent adv. + stylistic adv”.

The UI component allows the user to browse the database of fighters, sort by weight class, and add fighters to generate the odds for. Fighters can also be added via the search bar. When a fighter is added, it displays their Wikipedia picture and all their stats. When two fighters are added, the odds are generated in the middle of the screen along with a comparison of all their stats. A dropdown menu lets the user pick which algorithm is used to generate the odds. The user can also view the history of a fighter’s fights. The help button opens a window which explains how to use the program.

A problem this project solves is the problem of having no easy way to view all the relevant stats of two fighters when deciding who to bet on. To view two fighter’s fight history and statistics, I would need to have at least four browser tabs open. This project allows easy access to all of the 2000 fighter’s stats in one location. The betting odds that are generated can give an estimation of what the odds for a hypothetical matchup would look like. This can be especially useful when comparing the generated odds for an upcoming fight to the actual betting odds that bookies have. A disparity in odds is an opportunity for someone to potentially make money, especially for lesser known fighters.

Emmett Krochock
CS480
2 December 2021

The key piece of technology I had to learn to complete this project was to learn Python. Before I began writing code, I went through the first section of a free online Python book that taught all the necessary topics to do all the basic programming activities (types, conditionals, loops, I/O etc.). I had never used a dynamically typed language before, so a large part of learning Python for me was getting used to the fact that the type of a variable is inferred when declaring it. Python is designed to be an accessible language for non-programmers, so coding in Python seemed pretty intuitive to me as someone who already knows Java and C++. I chose to do all my coding for this project in the IDE PyCharm, which didn't cause any particular difficulties.

To compile the initial list of fighter names, I decided to use the Wikipedia API to extract all the links from a page of all notable male MMA fighters. I knew what an API was, but I had never used one before. Wikipedia's API is pretty straightforward, so creating a query that returned all the links on a page didn't take much time. The fighter names contained in my query were not encoded in UTF-8 format so I also learned how to encode these characters correctly in Python. Once I had the UTF-8 encoded fighter names, I could URL encode those names to create a list of links that my web-scraper would visit.

The most difficult thing to learn in my project was learning how to use the BeautifulSoup4 Python module to build my first web-scraper. BeautifulSoup4 is a widely used web-scraping module that I came across when researching the topic of web-scraping for this project. After reading the documentation for the module, I learned how find specific elements in an HTML file. The examples in the documentation were based on basic HTML pages, but

Emmett Krochock
CS480
2 December 2021

Wikipedia's HTML files are much larger and more detailed. Wikipedia pages contain many tables and nested elements, so most of the difficulty in creating my web-scraper came from analyzing the Wikipedia HTML files and understanding the type of elements used in the page. Specifically, learning how to traverse an HTML table required me to learn some basics of HTML. I learned that an HTML element contains a start tag, end tag, attributes, and the contents. After I learned how to access the text contained in an HTML element, I had all the knowledge needed to build my web-scraper.

The last piece of technology I had to learn was PyQt5, the set of Python bindings for Qt, a widget toolkit for creating GUIs. I had all the OOP knowledge necessary to create a UI, so I just needed to read the Qt documentation to learn the types of widgets that Qt contains. Starting my UI was not difficult, I just created the main window that contained a single label. The two challenges I encountered when building my UI were displaying an image and finding out how to connect buttons to functions. The image displayed below a fighter's name is their picture on Wikipedia. To convert image data from a webpage to a pixmap that could be displayed in the UI, I ended up using the PIL (Python Image Library) module. To connect functions to buttons, I passed the functions as lambda expressions. I realized that to connect a function to a listener, I needed to pass the function itself, a lambda, not what is returned by the function.

The parts of my project that went especially well were coding the UI and coding in Python generally. After I went through the Python tutorial course that I used, the syntax of Python didn't provide me much trouble. By the time I wrote my web-scrapers and wrote my program to rank all the fighters, I had a good enough grasp on Python to where the UI came

together quickly. I am also satisfied with the final size of my database, 2,000 fighters. After expanding my database from 1,700 fighters to 16,000 and filtering those fighters down to 2,000, I noticed that that the amount of data missed by my web-scrapers went down. Once I expanded the database, many notable fighters that were absent, were now present.

The parts of my project that didn't go as well were creating the first web-scraper and creating algorithms to generate betting odds. When creating the first web-scraper, I was learning Python, BeautifulSoup4, and HTML in the process. The code used in my web-scraper could be structured much better. There is also some percentage of data missing from certain fighters in my database that my web-scraper didn't collect. If I were re-do anything in my project, it would absolutely be my first web-scraper. By the time I wrote my second web-scraper, I had a better handle on BeautifulSoup4 and there wasn't any missing data when the web-scraper visited a page. I could certainly reduce the amount of code in my first web-scraper by about half, but I think I could almost eliminate any missing data by using another website instead of Wikipedia. There is a ton of data on every Wikipedia page and the relevant data that I needed could be found elsewhere. Because of some data that the web-scraper missed, a lot of the odds that can be generated are not very useful. If a fighter's fights aren't collected, they are treated as if they are a fighter with the default rating.

I initially viewed the goal of my project as generating the most accurate betting odds possible for MMA fighters. By the end of the project though, I viewed my main objective as creating a comprehensive database of fighters that includes a UI component to browse fighter stats. Coding the first web-scraper took a long time, and at that point, I knew I still had to create

a UI. After coding the web-scrapers and UI, there wasn't nearly enough time to create sophisticated algorithms for the betting odds. I was fine with this, though, because I didn't have enough time to research different algorithms to use in my project. I also realized that I don't know much about data science or data analytics, which is a skillset that would have been useful in creating useful betting algorithms. Generating useful, accurate betting odds for the fighters in my database would be a whole other project. Without having to create a UI component, I would have had much more time to generate more accurate betting odds. However, I am still pleased with how my project turned out. I would rather have web-scrapers and a functioning UI as opposed to dedicating my project just to creating betting algorithms.

In completing my project, I not only learned how to use tools like Python, Qt, and BeautifulSoup4, but I learned that the best way to learn new skills in computer science is to create a project from scratch like I did. With a goal in mind of creating a fighter database and odds generator in Python, I discovered new tools and concepts that I hadn't encountered before. I now feel confident that given a coding project, I have the skills to research and learn what is needed in order to complete it.