Ross Friscia
Prof. B. Herman
DATA 515
14 March 2022

Sportsbook Manager Written Report

Introduction

Sports gambling is going through rapid expansion. Prior to 2018, only Nevada had legalized the activity. Less than four years later and 29 states have legalized the practice with 12 more states either in the process of legalizing it or introducing a bill to legalize it. With the practice expanding, more companies are providing access to sports gambling by creating their own sportsbooks. For people interested in creating models to predict the outcomes of sports this is exciting. However, getting the wagers from the websites is made intentionally difficult. Sportsbook websites engage in various practices to hide wagers and make them difficult to programmatically pull. From requiring logins to view wagers, to hiding relevant information in custom attributes, or storing different types of information in the same attributes, scraping sportsbook information can feel like more trouble than it's worth.

This is where sports-book-manager steps in. Sports-book-manager helps programmers connect their model predictions to the sportsbook of their choice. By inputting the necessary html information into an instance of the BookScraper class, users can get back all wagers of the sportsbook of their choice. This, along with a couple of python modules to convert and compare market odds to model outputs, means sports-book-manager does the tedious work for the user, and saves them time.

Functional Specification

As mentioned above this package is targeted towards people who are creating predictive models for sporting events. This package assumes that the user is familiar with coding as well as inspecting html code and extracting the relevant information from it to web scrape.

The package requires the following python packages to be installed: pandas, SciPy, and Selenium. It also requires ChromeDriver to run Selenium. The package has two use cases:

1. Scrape wagers from the selected sportsbook
    i. USER: creates instance of BookScrape class entering the sportsbook website domain string and directories dictionary
    ii. USER: updates the object attributes with relevant html information to scrape the website
    iii. USER: call the sportsbook page to scrape using a key from directories
    iv. PROGRAM: runs the Selenium scraper to retrieve the information
    v. OUTPUT: pandas dataframe containing the Teams, Lines, and Odds of all events

2. Use modules and pandas' lambda function to convert odds and create new columns

i. USER: use implied_probability_calculator inside a lambda function with market odds
ii. OUTPUT: new column with market's expected probability at the market's line
iii. USER: use model_probability inside a lambda function with model mean, model standard deviation, and market line
iv. OUTPUT: new column with model's expected probability at the market's line

Component Specification

The package's modules are the following:

1. book_scrape_class.py – This module contains the BookScrape class that users employ to store and reference website information, html elements, and ChromeDriver information to run the Selenium web scraper. When an instance is created, the user must set the domain string and the directories dictionary. The class has the following public methods:
    i. update_directories – This method appends key values or creates new dictionary items in the instance's directories attribute.
    ii. update_html_elements – This method is used to update all other attributes inside the instance. It will only update attributes that are set by the user when calling the method. ancestor_container, event_row, team_class, line_class, and odds_class must be set for the following method to run, and each utilize a private method to ensure the values are formatted for a Selenium scraper.
    iii. retrieve_sports_book – This method takes a directory key and runs the Selenium scraper to retrieve the wagers as a pandas dataframe. This is done using several private methods that initialize the driver, finds the team, line, and odds of each event, and return the results.

2. implied_probability_calculator.py – This module takes American odds (it assumes that is how the sportsbook will display them) and converts it into the market's expected probability. This helps the user have an easier time understanding what the sportsbook is saying. One note, you should expect the sum of implied probabilities between all sides in an event to be greater than one. While yes, in the real-world probability cannot be greater than one, sportsbook's weigh events this way because that is how they make their money.

3. model_probability.py – This module takes a model's mean, standard deviation, and the market's line and returns the model's expected probability of a team covering the line set by the market. This will allow users to see what their model thinks at the market's line and quickly draw comparisons against the market's implied probability.

The flow of components is creating an instance of BookScraper establishing the domain and directories, updating the directories dictionary as needed using update_directories, setting necessary scraper information in the object using update_html_elements, and running the scraper and retrieving the sportsbook information using retrieve_sports_book. Once these steps are done, the flow moves from the object and its methods to manipulating pandas dataframes. Run implied_probability_calculator inside a lambdas function to get a column of the implied probabilities, then join a table of the model's outputs table to the dataframe, and finally run

market_probability inside a lambdas function to get a column of the model's expected probability. From there the user can compare the implied probabilities to model expected probabilities to see what bets their model likes best. An example of this flow can be found in the package's example notebook.

Design Decisions

The first design decision I made for this package is utilizing Object-Oriented Programming (OOP) to store the scraper information inside of instances of a class instead of using functional programming. I did this for a few reasons, the first of which being an instance's ability to store attribute information. My experience with scraping is it can take a few times to get everything right, which is more likely to be true here with how sportsbook's store their information. This allows users to keep track of what values they are setting for the scraper and make easier, piecemeal changes with update_html_elements.

The other reason for making the scraper using OOP is because of code organization. Building the functions for the scraper, I was finding a clear relationship between them. Using OOP, I bundled the code into logical units instead of having a package with a long list of modules that appear like separate entities. This bundling allowed me to minimize what methods the user had to interact with by making private methods do a lot of the lifting, as well as made the code more intuitable.

The benefit of OOP I utilized was the ability to reuse code instead of duplicating it. I made private methods that are called multiple times by the public methods. This occurred with the private methods utilized __value_formatter and __get_wager_values. I believe the benefits of OOP still show themselves in my package.

Another design decision I made was the choice of using Selenium instead of BeautifulSoup to scrape the sportsbook. One benefit of Selenium is its ability to locate elements to extract data. BeautifulSoup does this as well, but I found Selenium's CSS_SELECTOR more capable in retrieving most sportsbook's difficult to retrieve information. Another benefit of Selenium is its ability to click and interact with a website. This allows it to circumvent login protected information; a common deterrent sportsbook utilize. While this functionality is still future work to be done, it can be done and will be a valuable addition to the package. BeautifulSoup is known for being a faster scraper than Selenium, but the amount of data being extracted in this use case is minimal. This makes speed, which is usually one of the most important factors in picking a scraper, not a large concern for this use case.

More design decisions are discussed in the following section.

Comparison to PySBR

PySBR is like sports-book-manager's BookScraper in a couple of ways. They are both sportsbook scrapers, and both utilize OOP to scrape. However, that is where the similarity stops. PySBR emphasizes doing all the scraping work for the user, and on the surface appears better than BookScraper. Users can pull events by league, sport, or date, it asks the user to interact far

less with the interface than BookScraper does, and it allows users to skip the trial and error of scraping the websites themselves.

All these things sound great, but this package is meant for a different type of user and has limitations. First, PySBR has only a few sportsbooks that can accessed since it relies on sportsbookreview.com to retrieve the wagers. Currently, the website only tracks four different sportsbooks, so users better hope the one they want is there! BookScraper allows users to retrieve wagers from the sportsbook of their choice because of the class's flexibility.

Another limitation of PySBR is it relies on both a website not changing its structure as well as another website keeping up to date with multiple sportsbooks and their website design changes. These changes occur, and while users of sports-book-manager may be frustrated, they can take the necessary steps to alter their object's attributes to have their code working again. Sportsbookreview.com have removed sportsbooks from their website before due to difficulties pulling the information. This means if they pull the sportsbook a PySBR user is interested in the user will have no recourse (besides downloading sports-book-manager!). Sports-book-manager is designed for users who can build models to handle some interface interaction to scrape websites, and this expectation allows the package to be far more flexible than PySBR.

Extensibility

Sports-book-manager is built to be extensible. Separation of concerns was frequently considered when building methods, meaning it is easier to pinpoint where errors occur. Methods do not function with each other unless they must like in the case of the retrieve_sports_book private methods. The result is a package loosely coupled and built to minimize dependencies. Most of the package is inside of the BookScraper class, but the attribute updaters interact with the scraper methods in no way. All of this makes it easy to add or change aspects of the package without affecting others. Future work to add a login work around was mentioned earlier in this report, but that addition will be easy to make because of how the package is organized.