# English Premier League
# Match Prediction Based on Online Sentiment

Now You Can Bet with Extra Confidence!

(Draft - Dated Mar 25, 2024)

Rishikesan “Rishi” Ravichandran

# Table of Contents

# Executive Summary

Sports betting has been around for a long time, and it's becoming more popular each year. It's not just important for bookies, but also for people who bet. Before we make bets, we need to know more information. We should also consider betting on sports we're not familiar with to have a wider range of options.

Think of it like this: instead of just one person's opinion, we look at what a lot of people think. This is similar to the Delphi method. If many people feel strongly about something, there's a good chance it's true. We can use this idea to predict sports outcomes.

To find out what people think, we gather data from places like Reddit and ESPN. The more different kinds of data we have, the better we can understand what people think about upcoming matches.

We used NLP to analyze this text data and give scores for each team playing. This helps us predict which team might win and what are the odds for each match. We will in future try to automate all these processes and use the most appropriate model for each league.

We collected data from Reddit and ESPN using web scraping in a recent time span before a match, stored it in MongoDB, and analyzed it using Python. These analyses give us new ways to understand sports sentiment and help us make better predictions. In the future, we will also look to incorporate tools like Grafana to see prediction reports in real-time.

# Background Research and Industry Relevance

We try to use the concept of crowdsourcing i.e when we use the opinions of a lot of people to guess what might happen in the future. In sports, this often means betting on the teams or players that most people think will win. It turns out that this method is usually pretty good at predicting what will happen. For example, in a study of UFC fights, the opinions of many people were better at predicting who would win (85.7%) than the opinions of bookies (67.6%).

With all the new technology and tons of stats available from past games and players, many people are using these numbers to make predictions. That's okay, but sometimes we wonder if that's all there is to it. Fans believe they understand sports better than anyone else because it's not just about the numbers. There's also a feeling and connection between players and fans that goes beyond data. Even though we can't see it directly, we can sense it collectively on a large scale. That is the fundamental idea of this project.

If we do it carefully, keep learning, and make changes as needed, we can use these model odds to improve our sports betting. This could help us make more money from our bets in the long run.

In the past, sentiment research on sports like the Premier League and baseball was mostly based on Twitter. But now, with more advanced systems like LLMs available, we can do even better sentiment analysis, especially if we fine-tune these systems properly. This helps us understand what a group of people thinks about a particular event, which can be useful for betting.

While we originally wanted to use Twitter data, it's now expensive due to regulations. So, we're focusing on Reddit for public discussions and ESPN for more critical threads. Reddit is a big forum where lots of people talk about different topics in specific subreddits, which is perfect for each type of league.

# Introduction to the Data

The main goal of our current analysis was to predict games in the English Premier League (EPL). The code for scraping data and an example of analysis are available in the current directory of this document. Additionally, some important functions are included directly in the document's appendix.

## Data Sources

Our main sources of data come from Reddit and ESPN. To access Reddit data, we use Reddit APIs. This involves creating a Reddit account and generating an API token with that account. As for ESPN data, we're still working on it. We'll use standard web scraping methods like Selenium and Beautiful Soup to get the information we need.

## Web-Scraping Routine

We've created a Jupyter notebook file for scraping data from Reddit and ESPN. Users only need to provide two pieces of information:

1. Specify the league you want to analyze. Make sure to use the exact name used in the subreddit you're interested in. This name will be used as the base name for the MongoDB database as well. Different collections like Reddit and ESPN will be created within this database for each match.

2. Provide the match details. Just use a single word for each team that best describes them. For example, instead of "New York Yankees vs New York Mets," we can simply enter "Yankees vs Mets."

```
league_name = "PremierLeague"
match = "Liverpool vs United"
```

Note. We can replace "PremierLeaguel" with the league we're interested in and specify the teams for the match accordingly. For instance, here “PremierLeague” which is the subreddit name for English Premier League, and the match could read as “Liverpool vs United”.

The data will be scraped and saved automatically in a local MongoDB database. The database name will be based on the league name you provided. Both Reddit and ESPN data will be stored within this same database. Each match will have its own collection within the database.

## Database Design

The database design is straightforward. We're using MongoDB to store the text data because it's flexible and useful for our needs. Since we're scraping from two sources and may add more in the future, MongoDB can handle them all easily. It's also good for storage. So, considering complexity, design, and storage, MongoDB is better than SQL databases.

First, we create a database in MongoDB based on the league_name value. If it already exists, the documents will be added under the existing database collection. Each match will have its own unique collection with a date. So, if the same teams play again in the future, it'll be stored as a different collection with a new date.

To distinguish between sources, we include a source field in our document.

```
data = {
    "title":title, #Title of the post
    "description": description, # Description of the post
    "comments": comments, #An array of possible strings
    "source": "reddit",
    "time":post.created_utc
}
```

Each document in a collection contains unique information about a Reddit post's title, its description, or an array of its comments. If the source is ESPN, the document will be labeled as ESPN, but other fields could vary. In the next section, we'll learn how to convert this raw data into a cleaner format for different sources.

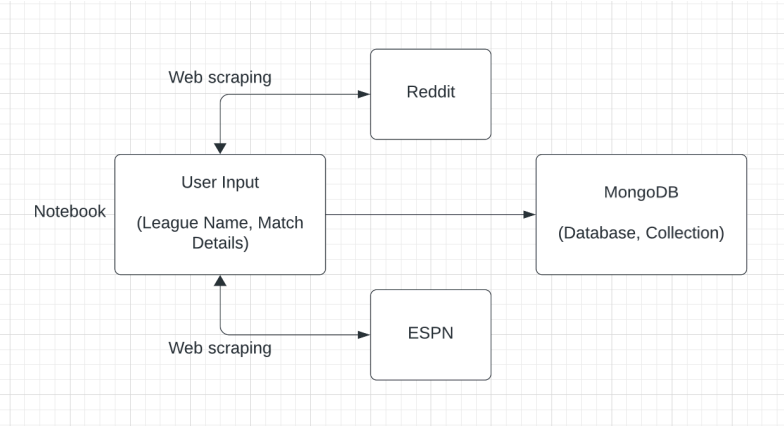



Fig. Flowchart of Data Scraping and Data Storage

# Data Analysis and Business Impact

## Data Retrieval and Transformation

If a user wants to get details about a specific match from the MongoDB database, they can provide the collection name (which includes the match name and date) and the league name (the database name). If the data retrieval happens on the same day as the scraping, it's done automatically based on the initial inputs.

Once the data is fetched from MongoDB, some transformations need to be made, and initial labeling is required. Based on the match information, we'll create two labels for the two different teams.

When the data source is Reddit, each document (which represents a Reddit post) will be examined. We'll look at the post's title, description, and comments, treating each as a separate text string. We'll follow a simple rule: if the text string mentions only one of the teams from the match, we'll assign it to that team. If it mentions both teams or neither, we'll ignore it. In the end, we'll have a dataframe with rows representing titles, descriptions, or comments of all posts related to the match in the past few days, with each text string assigned to a team. We'll also develop a similar transformation method for ESPN in the future.

| | team | message |
|---|---|---|
| 0 | liverpool | [john cross] fixture list in chaos... tottenha... |
| 1 | liverpool | how come liverpool get *this many* early sat... |
| 2 | liverpool | didn't liverpool play 3 games in 7 days and 7 ... |
| 3 | liverpool | the whole “put liverpool on at saturday 12:30”... |
| 4 | liverpool | acting like everton will pose any threat to li... |
| 5 | liverpool | here we go arsenal and liverpool getting their... |
| 6 | united | don't remember much sympathy when this was hap... |
| 7 | liverpool | the 12.30 slot is popular in asia because it i... |
| 8 | liverpool | it does seem silly though that liverpool play ... |
| 9 | liverpool | just seems a bit unfair when there’s no game f... |

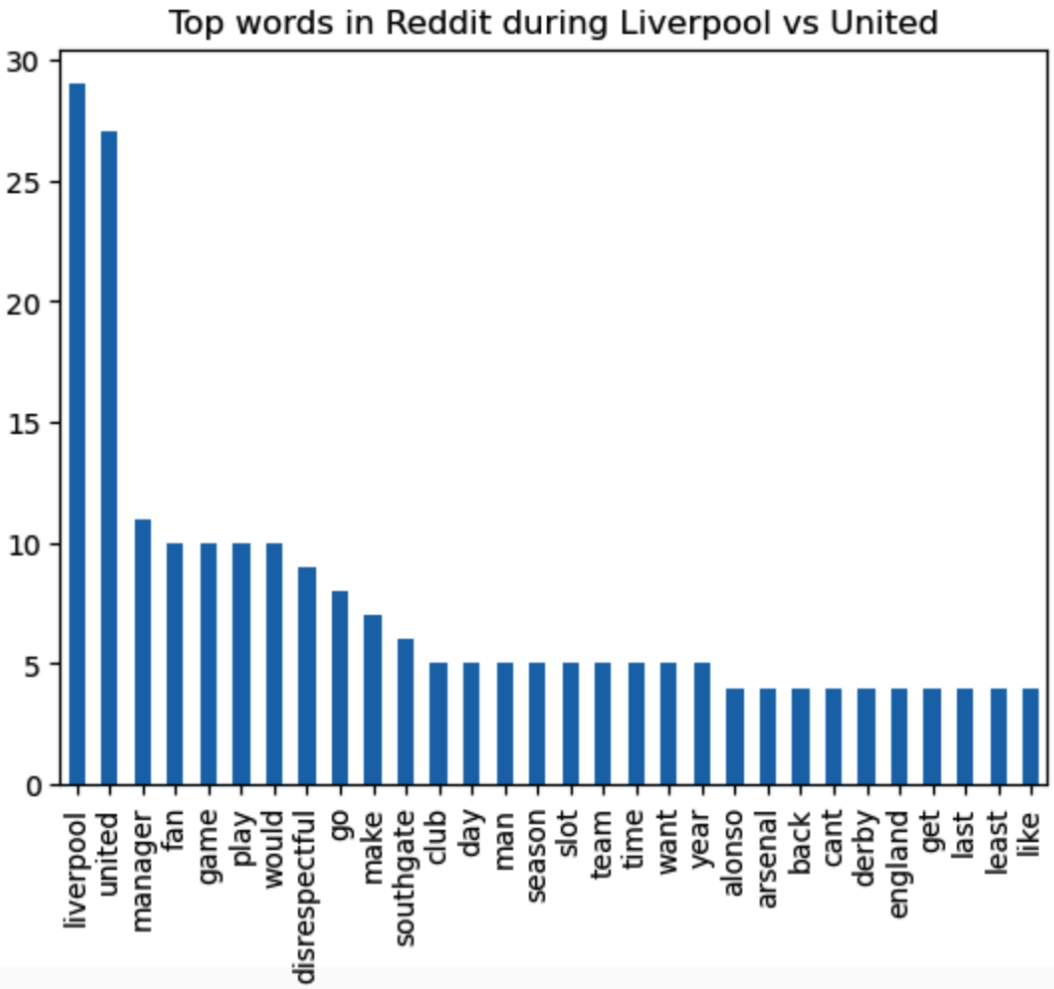
Fig. Transformed Reddit Data

## Data Preprocessing

Before analyzing the data, we're making some changes to it. We're converting all the messages (text strings) to lowercase and removing punctuation using NLTK libraries for language processing. This makes it easier for us to analyze sentiment. Since preprocessing comes after labeling all our texts (from sources like Reddit, ESPN, etc.), we only need one function for this task.

## Data Modeling

We're using a simple modeling method called "bag of words." After preprocessing the data, we start by finding the words that appear most frequently in the scraped texts.




We noticed that not all words express sentiment, and the texts are not equally balanced for each team (though this is just a sample). So, based on some common words and our experience, we created a sentiment dictionary. In this dictionary, a word is labeled positive if it's assigned a +1 and negative if it's assigned a -1. We'll refine this dictionary in the future.

```
{'disrespectful':1,'cant':-1,'last':-1,'least':-1,'like':1,'strength':1,'win':1,'lose':-1}
```

Using this approach, we calculate a total score for each text. If the total score is greater than 0, we label it as positive; if it's less than 0, we label it as negative; and if it's 0, we label it as

neutral. Then, we calculate the proportion of positive sentiments for each team, considering it as the winning proportion among fans for that team. Later, we normalize this proportion to determine the odds of each team winning.

## Results

Based on what we've learned, we have odds on the teams based primarily on discussions among the public or fans in online forums, as well as critics' discussions or predictions from ESPN channels. This will help us make smarter decisions when betting and increase our chances of winning and getting a return on our bet before a match.

| sentiment_label<br>team | %positive | %negative | %neutral | win_percentage | normalized_win_percentage |
|---|---|---|---|---|---|
| **liverpool** | 7.407407 | 14.814815 | 77.777778 | 27.434842 | 14.935823 |
| **united** | 37.500000 | 29.166667 | 33.333333 | 156.250000 | 85.064177 |

(Fig. Sample Prediction Table)

# Summary and Conclusion

We successfully collected data from Reddit, stored it in a database, retrieved it, transformed it, and preprocessed it for analysis. Then, using a simple model like Bag of Words, we predicted the probability of an EPL match based solely on data from online forums. This helps us understand the odds of a match before betting.

As we move forward, it's crucial to make sure that the results we get are really reliable. We need to keep a close eye on how these results affect our business, especially the returns we get from bets after using the information from our text analysis. Additionally, we'll discuss in the next section how we can improve the methods we've been using so far.

# Future Work

We need to improve how we label teams in each text, possibly by considering factors like the number of times a word appears or if a specific team's name is mentioned.

It's important to compare our predictions with the actual match results and store both for future adjustments to our model.

We can try different modeling approaches:

- Model 2: Using a similar method as explained in Model 1, but with ESPN data.
- Model 3: Combining predictions from both Model 1 (Reddit) and Model 2 (ESPN) for better accuracy.
- Model 4: Fine-tuning advanced language models like llama2 or GPT-3 with our data. It would be best to label the data before training.

A more refined approach to Bag of Words can also be followed.

# References


1. Surowiecki, J., 2004. *The Wisdom of Crowds* New York: Doubleday.

2. Lewis, M., 2003. *Moneyball: The Art of Winning an Unfair Game*: WW Norton & Company.

3. Predicting Wins and Spread in the Premier League Using a Sentiment Analysis of Twitter

Robert P. Schumaker[1], A. Tomasz Jarmoszko[2] and Chester S. Labedz Jr.[3]


# Appendix

Function to scrape data from Reddit:

```
def scrape_reddit(match,collection):
    """
    Function to scrape data from reddit through Praw API.
    """

    keywords = [team.lower() for team in match.split(' vs ')]

    #Get the current time and subtract 3 days from it, as we'll only be
scraping the last 3 days of data
    three_days_ago = (datetime.now() - timedelta(days=3)).timestamp()

    # #Create a dictionary to store all the data
    # data = {}

    #Get the posts based on the keywords generated from the match, i.e only
the recent 300 which are also within last 3 days
    for post in subreddit.new(limit=300):

        #Stop scraping if the post is older
```

```
        if post.created_utc < three_days_ago:
            break

        #Get the title information for the post
        title = post.title.lower()

        #Check for the post titles that contains our keywords
        if any(key in title for key in keywords):
            # sleep for 5 seconds to avoid hitting the API too hard, this
will be only before fetching details of a post
            time.sleep(5)

            #Get access to the post with it's id through the API
            post_obj = reddit.submission(id=post.id)

            #Get the post's title
            title = post_obj.title

            #Get the post's description
            description = post_obj.selftext

            #Now we'll try to fetch all the comments of the post with no
limit for the 'More Comments' in Reddit
            post_obj.comments.replace_more(limit=None)

            comments = []

            #Get all the comments of the post
            for comment in post_obj.comments.list():
                comments.append(comment.body)

            #Add the post's data to the dictionary
            data = {}
            data = {
                "title":title, #Title of the post
                "description": description, # Description of the post
                "comments": comments, #An array of possible strings
                "source": "reddit",
                "time":post.created_utc
            }

            doc_insert = collection.insert_one(data)
```

Function to scrape from ESPN:

```
def scrape_espn(match,collection):
    '''
    Function to scrape data from ESPN.
    '''

    #Create a driver object for the chrome
    driver = webdriver.Chrome()

    #We'll wait for 5 sec before sending each request

    #Go to ebay.com
    time.sleep(5)
    driver.get('https://espn.com/')

    #search for "Cell Phones"
    time.sleep(5)
    search =
driver.find_element(By.CSS_SELECTOR,'a[id="global-search-trigger"]')
    search.click()

    time.sleep(5)

input=driver.find_element(By.CSS_SELECTOR,'input[id="global-search-input"]'
)
    input.send_keys(f'{match} \n')

    #Click the articles filter
    time.sleep(5)
    articles = driver.find_element(By.XPATH, '//a[text()="Articles" and
@data-track-filtername="articles"]')
    articles.click()

    #Check for the recent articles based on similar timeframe as reddit,
default: 3 days
    #li.class_='time-elapsed', re.compile(r'(\d{1,24}h|[1-3]d)',li.text)

    #Get the page source
```

```
    page_html = driver.page_source

    soup = BeautifulSoup(page_html,'html.parser')

    #Just take the first article for now. This will be ideally done based
on the timeframe given globally i.e for both reddit and espn.

article_url="https://www.espn.com"+soup.find_all('ul',class_='article__Resu
lts')[0].find_all('a')[0].get('href')

    time.sleep(5)
    page_article = requests.get(article_url, headers = {'User-agent':
'Mozilla/5.0'})
    soup_article = BeautifulSoup(page_article.content,'html.parser')

    #Title of the article
    title = soup_article.find_all('h1')[0].text

    #Description of the article
    description=""
    for p in range(len(soup_article.find_all('p'))):
        description+=soup_article.find_all('p')[p].text

    #Timestamp of the published article
    article_time =
soup_article.find_all('span',class_='timestamp')[-2].text

    #Add the article's data to the dictionary
    data = {}
    data = {
        "title":title, #Title of the article
        "description": description, # Description of the article
        "source": "espn",
        "time": article_time
    }
    print("Sample Article Document shown below:\n")
    print(data)
    doc_insert = collection.insert_one(data)
```

Reddit Transformer for Analysis:

```
def transform_label_reddit_data(match,data):
    '''
    This function will transform the raw data into a dataframe, where every
row will be a english sentence from the title/comments/
    description of a post, labeling to the keywords from the match string.

    The rules are based on assigning a sentence text to a keyword only if
one of the keywords exists in that sentence. If both keywords
    exists in a sentence, that sentence will be ignored for now.

    However, in future, a more appropriate method can be enforced to
accommodate those as well.
    '''

    keywords = [team.lower() for team in match.split(' vs ')]

    #Structure the raw data into a dataframe, every sentence will be a row
in the dataframe
    row=[]

    for doc in doc_list:
        title=doc['title']
        description = doc['description']
        comments = doc['comments']

        #Combine all the messages of a post i.e it's title, description,
and comments into an array
        messages=[title,description]+comments

        for message in messages:
            message=message.lower()

            #Check if the message contains any of the keywords
            if keywords[0] in message and keywords[1] not in message:
                row.append({"team":keywords[0],"message":message})
            elif keywords[1] in message and keywords[0] not in message:
                row.append({"team":keywords[1],"message":message})

    #Now create the data frame
    return pd.DataFrame(row)
```

Bag of Words Implementation:

```
#We'll define the sentiment score manually here, ideally this should be
automated or have a pre-fixed dict in future
#if Bag of words is implemented

#Assign the +ve or -ve sentiments to words observed and typical words in
sports, ad-hoc method**
sentiment_words =
{'money':1,'trade':-1,'minor':-1,'contract':-1,'deal':1,'last':-1,'strength
':1,'win':1,'lose':-1}

#Count only the words in sentiment_words
vectorizer_sentiment_scores =
CountVectorizer(vocabulary=sentiment_words.keys())
X_sentiment_scores =
vectorizer_sentiment_scores.fit_transform(data_fans['message'])

#Now convert to dataframe
df_sentiment_scores = pd.DataFrame(X_sentiment_scores.toarray(),
columns=vectorizer_sentiment_scores.get_feature_names_out())

#Now multiply each word by its corresponding sentiment score
for word, score in sentiment_scores.items():
    df_sentiment_scores[word] *= score

#Add the columns to our original dataframe
data_fans = pd.concat([data_fans, df_sentiment_scores], axis=1)
```