Kshitij Agarwal

Professor Barbara

CS 484: Data Mining

May 13, 2020

NBA Game Prediction

Intro

For this final project, I chose to see how accurately I could predict the winner of an NBA game by using the statistics by the end of the third quarter. I found multiple datasets that had many different stats that are usually collected throughout a game. However, most of these datasets had it by game instead of by quarter or by play. This would not allow me to predict games by the quarter. Since this project was very open ended, it was a struggle to get started and even decide on topic. However, I decided to stick with the game prediction, as it was something that interested me.

The first task of this assignment was to find some viable data. I turned to the best data mining source I found: Kaggle. A Wikipedia explanation of Kaggle is an online community of data scientists and machine learning practitioners. Kaggle offers its users a place to share and explore different datasets, methods, and models. I used this resource to search for a large dataset regarding professional level sports. I found data for just about every sport and league from European soccer to Indian cricket.

Looking through the different sports, I decided to stick to basketball because, statistically, it is the sport that is least based on luck. Unlike soccer, football, and hockey, basketball is constantly scoring points, and therefore, the team that has better stats usually wins. This would allow predicting to be a lot easier especially when you don't have all the data from every quarter. After finding a play by play record of every game, I knew that it would be a great project to do, especially since I had no direction of where to continue.

Reviewing Data

Choosing that dataset, I downloaded it so that I could open the file and start to understand the copious amounts of data in front of me. The file was easily accessible in Microsoft Excel, and I was able to start digging into every line. Across the top, every attribute was listed in terms of what information was found in that column. The first column contained a URL to a website from where the data was scrapped. I used this link to try to get a better understanding of what I was reading. From there, I was able to match the first few plays, and I saw exactly how the data was broken down, and I started to get an idea of how I can piece it together for my own use.

I started off by highlighting the different stats that I would want to record. This started off as the score, the opponent, assists, steals, blocks, rebounds, and fouls. In order to focus on this part of the data, I deleted anything that was not pertinent to those. I then read the first few data points per column in order to find a pattern that could help me extract some information. At first, it

looked as if it would not be possible to add up the stats because every game was thrown together. Trying to separate each game from one another, stop at the 3rd quarter for each game, and iterate through all the data points for one game and combine them into a single array was not clear.

Extracting Data

My first goal in extracting data had to be separating each game. This would then be followed by adding all the relevant data for that game in an array and storing it in a larger array. The first thing I thought about using was just the quarter; every time the quarter changed, I could add it in as a "new game" and then combine the first, second, and third quarter while discarding the forth and any overtime quarters. As I started this process for the first few lines, I printed each line of data. Doing this, I saw that every game note that was taken down, and I noticed each game had a note for the end of each quarter. This allowed me to add an if statement and collect data for each game until I reached that point. I also added an extra if statement that ensured I only looked at first, second, and third quarter data. This separated each game perfectly, and it gave me an option to change two lines of code and stop collecting data at the half if I wanted to do so in the future.

Once I was able to separate out the different games, I had to start collecting data that I wanted keep. Since there was a column that kept a running tally of the score for each team, I was able to add that data whenever I checked for the end of the third period. At that point, I realized that I could add any constant data at the same time. The winner, type of game, and opponent were all easily recorded at that point. This constant point allowed me to only add those data once, and it gave me a chance to reset the data to count the next game.

Since steals and blockers weren't a running tally, I knew that I would have to iterate through every data point. This also allowed me to continuously check for the "End of 3rd Quarter" message. This ongoing iteration also ensure that calculating the remaining stats to be much easier than I expected. The column spaces dedicated to blocks, steals, and assists were always empty unless there was a respective block, steal, or assist that took place per play. And since I know that you can't have a steal and a block on the same play, I was able to set up an easy if-else-if loop that would continuously check to see one of the stats column had any information for every play. If I was able to find any text in the column, I knew that there was data that I would need to take out. To ensure I gave a point to the right time, I checked if there was information under "Home Team Play" as that would indicate if the home team made the play.

After getting to this point, extracting any data I wanted became a repetitive process, and I was easily able to add a lot of information in the matrix. Taking a look at the matrix, I quickly realized how much data was in it, and I also noticed there were two different columns for each stat: one for the home team, and one for away team. Even though these values were correlated, it did not directly show how one team is better than the other. To solve this, I knew that the two stats needed to be combined in one to show if any team was able to outnumber the other team. Every duplicated stat was set to the home's stat minus the away team's stat. This worked really well with the whole dataset because only each team's home games were logged. This ensured that no duplicated game data was added to the set, and also calculated each game in the perspective of being at home.

Libraries Used

In order to quickly process the data and make it as accurate as possible, I knew that libraries were essential. Throughout this course, I have learned that SciKitLearn has almost every essential library for data mining, and I used the following libraries from them: TruncatedSVD, LogisticRegression, KNeighborsClassifier, DecisionTreeClassifier, RandomForestClassifier, and KMeans. These libraries were used to either make models or make the data more relevant and useful for the models.

TruncatedSVD is a dimensionality reduction library. It is designed to cut down the empty features across a sparse matrix in order to make the data more relevant to the application. In my case, I used this library to cut down from 15 features I extracted to just 4. LogisticRegression and KNeighborsClassifier are tests that compare one point to every other point. They, then, use their respective algorithm to use that and make prediction for new data points. DecisionTreeClassifier and RandomForestClassifier are learning models that produce a set of rules. They both build a series of "questions" based off some index, such as Gini, before asking the same "questions" to each test data point. Kmeans is clustering algorithm library. It graphs every point and tries to find a centroid for each of the different groups that form in the data. Whichever centroid is closest to a data point will be labeled as such.

Furthermore, I also had to use a few extra libraries to help manage the data that was collected. I used test_train_split to divide all of the data into two separate, random groups: one for training, and the other for testing. Evaluating KMeans is very different from evaluating the other models because KMeans evaluates and groups all of the data together. To analyze the accuracy of that model, I used the classification_report library, which gave me access to the F1 score. Finally, the last library I used was pandas. This library gave me the ability to quickly alter the matrix I created to run different tests.

Predictions

Prior to testing, I made two predictions that I thought I would be able to see throughout my tests. I predicted that the score of the game would be sufficient to accurately predict the which team had the best chance of winning. This is very likely because the chances of a fourth quarter comeback are not that high. However, the reason I decided to analyze basketball is because it is a sport where the better skilled team usually comes out on top. This means that the team that functions better together, has more assists and blocks, and is not fouling as much should have a better chance at winning a game even if they are down by a few points in the third quarter.

Testing and Results

To test the data and view the accuracy of different models, I simply had to feed my matrix into the different SciKitLearn libraries. Throughout the semester, we learned about different models and have written our own code to replicate them. So I knew that if I wanted to use models that I could implement and understand, the best ones to start with would be KNN, DecisionTrees, and KMeans. Throughout the semester, while working on previous homework assignments, I also researched about similar methods to see how I could improve my accuracy. I

found libraries such as LogisticRegression and RandomForestClassifier. These 5 metrics together allowed me to test my data in multiple ways and see how accurate I could make it.

Each of the libraries has its own modifier that adjusts how the algorithm calculates its data. Using different values for these, we can significantly improve the metrics. Using all of the data, I was able to get around 70% accuracy without making any adjustments. After making these adjustments, I was able to comfortably reach 79-80% for KNN, Logistic, and KMeans, and I was able to achieve nearly 84% for DecisionTrees and RandomForest. Adding the final library from SciKitLearn, TruncatedSVD, I was able to improve the quality of the data. This allowed KNN, Logistic, and KMeans to improve and match the levels of the other two at 84%. However, DecisionTrees and RandomForest both fell in accuracy. Due to having less features to compare and build an effective set of rules, both of these metrics became less accurate by a significant amount.

Using 84% as a benchmark, I adjusted the matrix to have only the difference in scores before running it through the models. The results came back and the accuracy was only 0.5% less accurate than the benchmark. However, due to a lack of dimensionality, KMeans suffered greatly. The accuracy dropped to 31%. Using the all of the data but the score still did drop the score. Some metrics were lowered by as much as 9%. However, at 75%, the accuracy of the prediction is still impressive. And this confirms my hypothesis that a stronger team does have better stats.

Benchmark Using All of the data:

```
Logistic Regression: 83.323%
KNN: 83.020%
Decision Trees: 82.899%
Random Forest: 83.323%

Kmeans:
              precision    recall  f1-score   support

           0       0.87      0.70      0.78      2607
           1       0.73      0.88      0.80      2387

    accuracy                           0.79      4994
   macro avg       0.80      0.79      0.79      4994
weighted avg       0.80      0.79      0.79      4994
```

Test Using only the score:

```
Logistic Regression: 82.838%
KNN: 82.838%
Decision Trees: 82.717%
Random Forest: 82.838%

Kmeans:
              precision    recall  f1-score   support

           0       0.28      0.24      0.26      2542
           1       0.33      0.39      0.35      2452

    accuracy                           0.31      4994
   macro avg       0.31      0.31      0.31      4994
weighted avg       0.31      0.31      0.31      4994
```

Test Using all of the Data but the Score:

```
Logistic Regression: 73.984%
KNN: 73.802%
Decision Trees: 73.681%
Random Forest: 79.442%

Kmeans:
              precision    recall  f1-score   support

           0       0.72      0.61      0.66      2487
           1       0.66      0.77      0.71      2507

    accuracy                           0.69      4994
   macro avg       0.69      0.69      0.69      4994
weighted avg       0.69      0.69      0.69      4994
```

Real World Usage

The main advantage of using data like this is useful in sport betting. Watching a game, you usually see who the better team from the very beginning. However, translating official statistics to predicting the winner of a game is still a challenge. And more importantly, there is always a chance to see if the underdog can pull an upset. Getting the chance of winning into a percentage is very important for companies that make wagers. However, this also can be used to show how well a team is performing even they the scoreboard isn't in their favor. The third test I ran showed that without the score, I was able to accurately predict who would win a game around 74% of the time. This could show that a team with better stats might still have a chance to win the game.