CS5830 Project 5 Report

Gavin Murdock and Jacob Johns

Introduction

This report outlines our analysis of men's NCAA basketball and our attempt to predict the winners of the annual March Madness tournament. To do this, we implemented a Naive Bayes classifier using real-world data containing detailed team stats and final outcomes of men's NCAA basketball games over the course of many years. This report will discuss us finding and preprocessing our dataset, building our naive bayes classifier, and testing the performance of our model through a variety of metrics. We hope this analysis can help March Madness fans, such as ourselves, make better bracket predictions in the future. The links to our Github project folder and presentation slides can be found below.

GitHub and Presentation.

Dataset

The dataset we used was made available by the NCAA and Sportradar LLC. It originally contained 132 data features on 29.8 thousand men's NCAA basketball games between 2013 and 2018. In processing this data, we took away almost 60 data features that we deemed unimportant, such as team logos and various venue information. The remaining features contained important game data on both teams, such as shooting percentages, turnovers, fouls, rebounds, and final scores. Based on final scores, we then added features for the home team winning and the away team winning. We then split our data into postseason and regular season games, using the regular season games to average each team's stats per season. Our goal was then to look at the postseason matchups and the differences in those team's season averages to predict the winner.

Analysis Technique

We used a Gaussian Naive Bayes classifier to predict which team will win a given March Madness tournament game. We chose a Naive Bayes classifier for a couple reasons. First, Naive Bayes is both popular and simple. Because of this, many libraries have provided free and easy-to-use implementations of the Naive Bayes algorithm, such as the one we used by scikit-learn. Second, Naive Bayes is fast. We could've used a classifier like k-nearest neighbor (kNN) to possibly get better results, but kNN is a lot slower when working with large amounts of data. Finally, after looking over our dataset we decided to use a Gaussian Naive Bayes classifier because we had mostly quantitative variables.

Results

To evaluate our classifier's performance, we first split our data into a training set containing 75% of our dataset and a testing set containing the other 25%. We trained our Gaussian Naive Bayes classifier using the training data and then used the classifier to predict the outcomes of all the games in the testing set. We then compared our model's predicted outcomes of all the games in the testing dataset against their actual outcomes in order to calculate the f-score of our model. F-score is a popular metric that takes

into account the precision and recall of a model in order to give an overall performance rating between 0 and 1, 0 being the worst and 1 being the best. Our model had an f-score of 0.493. That means our model isn't great at predicting the outcomes of NCAA basketball games, and our model wouldn't be very helpful to anyone using it to help fill out their March Madness brackets. One reason our model may not have done very well is because Naive Bayes assumes all variables are independent of each other, but many of our variables, such as turnovers and points off turnovers, are not independent. If we had more time, it would be interesting to test another classifier, such as kNN, on the same data to see if it would get better results.

Technical

To create our dataset, we pulled data using Google's BigQuery and the google-cloud-bigquery Python package. We then saved this data as a csv file we used to construct a Pandas DataFrame object. We then dropped columns we felt weren't helpful for classification using the .drop method and added columns for which team won each game. We finally split the data into regular season games and postseason games. For the postseason games, we extracted the season year, the home team alias, and the away team alias to use as our input data, and we extracted whether the home team won to use as our target data. To get our training and testing sets, we used the train_test_split function from scikit-learn with a test_size equal to 0.25 on our input and target data. Then, for each matchup in our X_train and X_test sets, we replaced the few features we had with the difference in means of various team stats that season for both teams. When calculating the means of stats, we replaced NaN values with 0. Finally, our input data was all quantitative and ready to be used by our Gaussian Naive Bayes Classifier to predict whether the home team won, which is represented by true or false in our y_train and y_test sets. We used a Gaussian Naive Bayes classifier for this project because it's easy to implement (we used just three lines of code to fit our model and generate a list of predictions) and is fast compared to other classifiers, like kNN. After using our model to generate a list of predictions, we evaluated our model using scikit-learn's precision_recall_fscore_support function on our y_test set and our generated list of predictions.