Comparison of Regressors in Predicting March Madness Brackets

Ashwin Wariar, Raj Singh, Athreyi Badithela, Jacynda Alatoma, Alec Lorimer

June 19, 2024

Abstract

The March Madness basketball tournament, an exciting event in NCAA college basketball, poses a unique challenge for bracket predictions owing to its unpredictability. With the use of statistical insights from NCAA men's basketball, there is very little chance of creating a flawless bracket, especially with the enormous popularity of bracket submissions. In this work, we investigate the difficulties involved in forecasting team victory percentages using machine learning methods, including Random Forest, neural network, and XGBoost. Our research highlights the complex nature of basketball prediction by taking into account both complex variables and data. We use the datasets provided in the "March Madness Machine Learning Mania" competition on Kaggle to assess how well different algorithms predict game outcomes. To determine the best method for bracket prediction, we will compare the Random Forest, Neural Network, and XGBoost models. Our findings offer important implications for sports analytics and tournament forecasting, illuminating the potential of machine learning techniques to improve bracket projections. Our findings reveal that the XGBoost Regressor performs the best with the lowest statistical error values, and the highest accuracy in a simulated tournament of NCAA March Madness games.

1 Introduction and Background

The March Madness basketball tournament is a nickname given to the NCAA mens and womens college basketball championship tournament. In this tournament, 68 teams play a single elimination tournament where they are ranked 1-16 seed based on their regular season record.

Every year, millions of people submit a bracket at the hopes of picking a perfect bracket, though the odds of making a perfect bracket are next to none. If someone were to flip a coin to select the winner for every game, the odds of making a perfect 63 game bracket is 1 in 9.2 quintillion. According to Jeff Bergen, a professor of mathematics at the University of DePaul, with the addition of knowledge of NCAA men's basketball into the processes, Bergen says that the odds of creating a perfect bracket can be dropped to 1 in 28 billion[2], and still no one has created a perfect bracket before[1].

There are many recorded metrics depending on your theory and understanding of basketball that determines what makes a team 'good'. But there are also plenty of unrecorded factors that affect whether or not a team will win the game. Are they playing home or away? What kind of offense/defense does each coach like to play? How many players are injured? How will the referees officiate the game? These are just a few examples of the many questions that someone may take into account for determining the outcome of one game. It also shows how complex of a problem it is to try to predict a perfect bracket.

Each year, Kaggle runs a ”March Madness Machine Learning Mania” [3] code competition where people may submit predicted brackets generated by machine learning algorithms to see what algorithms preformed the best this year upon the conclusion of the NCAA March Madness tournament. Within this challenge, Kaggle provides datasets in which we tested different algorithms on to find what algorithms preformed the best as well as what metrics provided the most impact upon determining the winner or loser for a particular game.

2 Dataset

Like previously mentioned, Kaggle provides a dataset from the March Machine Learning Mania 2024 code competition [3]. This data set contains the current years games and information as well as historical data back to the 2002-2003 season. This dataset is broken up into 5 different data sections.

2.1 Identification Data

The first data section contains 10 files (2 for each gender) that provide general information about the teams and games played.

- `Teams.csv` - Contains the human readable identification for teams. The `TeamID` and the `TeamName` to do a translation from a TeamID into the names that we are familiar with. This also contains the first and last season that the team has been classified as a division 1 team with the last season for current teams always being listed as the current year.
- `Seasons.csv` - A short file containing simple information about the season such as the human readable date in which the corresponding season started. More importantly though, this file contains the region assignments in terms of W,X,Y, and Z. These regions in terms of there naming assignment are arbitrary with the first alphabetical region being assigned W as some regions change name from year to year. After this the region that plays W in the semifinal is assigned region X. This process is repeated for regions Y and Z.
- `NCAATourneySeeds.csv` - Identifies the NCAA tournament seeding for a given team if they made the tournament. This seed value is broken into two parts with the first part being the region character and the second part being the numerical region seed for the given team from 01 being the top seed to 16 being the worst seed.
- `RegularSeasonCompactResults.csv` - This file contains the first look at general data that can be used to build a model from. This file contains the results of games with the winning team ID losing team ID and the score for the winning and losing teams as well as how many overtimes occurred. It also contains the date of the game through which season and day of the season the game took place. Lastly it tells where the the game took place based off if the winning team was the home or away team or if the game was played at a neutral site.

- NCAATourneyCompactResults - Lastly is the tournament results. This information would be best used as testing data for the models as it only contains the similar information to RegularSeasonCompactResults.csv but for the tournament games.

2.2 Team Box Scores

This group of data contains the ”box scores” or more granular information about the game. Here we get to see statistics about the game such as how many rebounds the winning and losing team got. How many field goal attempts were made. This data is best to display the characteristics of a team as if this data is watched throughout a season, we can see if a team is a good shooting team, good rebounders or good defensively. Below is the full list of categories kept in the files. These categories are shown for the winning and losing team but displayed with a ”W” for winning and ”L” for losing team before the shorthand statistic ID.

- FGM - field goals made
- FGA - field goals attempted
- FGM3 - three pointers made
- FGA3 - three pointers attempted
- FTM - free throws made
- FTA - free throws attempted
- OR - offensive rebounds
- DR - defensive rebounds
- Ast - assists
- TO - turnovers committed
- Stl - steals
- Blk - blocks
- PF - personal fouls committed

Some stats can be generated from combining or splitting the listed stats. Field goals made (FGM) is equal to the number of 2 point field goals and 3-point field goals. If we would like to get the 2-point field goals made (FGM2) we could get this data with $FGM2 = FGM - FGM3$. For the games, this information like in the identification data section, the regular season games are split from the tournament results in two separate files RegularSeasonDetailedResults.csv and NCAATourneyDetailedResults.csv.

2.3 Geography

The geography of the games is broken down into two simple files. The first file Cities.csv contains a list of CityIDs that correspond to a City name and State. The second file GameCities.csv contains the combined list of where and when a game was played between two teams, what kind of game it was (Regular, NCAA Tournament, or secondary such as a preseason tournament) and lastly the city that the game was played at given by the CityID.

2.4 Public Rankings

The public rankings file MasseyOrdinals.csv contains the historical data for the public rankings of every team at different points within the season from different ranking outlets/systems. This information is useful as a variety of different ranking systems (Pomeroy, RPI, ESPN, etc.) have their own different metrics on how they rank teams. This information also contains a ranking for every team, not just the top 25-30 like the coaches poll releases. This way we could track the progress of a team throughout the season for something like ”winning momentum”. Within this file, the teams receive an ordinal rank such as #1 through #351 with 1 being the best and the system in which they received the ranking from.

2.5 Supplementary Datasets

The last section of the data are the supplementary datasets. These contain other pieces of information that may encapsulate some of the attributes that are difficult to note.

- `Conferences.csv` and `TeamConferences.csv` - While there are different divisions withing NCAA basketball, the conferences can be a good indication of difficulty of play. These two documents specify what schools belong to what conferences.
- `ConferenceTourneyGames.csv` - Before the NCAA tournament, all conferences have a tournament where the winning team from each conference is given an automatic bid to the NCAA championship tournament. This document provides the name of the conference, season, and day in which a winning team and losing team played each other.
- `SecondaryTounreyteams.csv` and `SecondaryTourneyCompactResults` - These two file contain the information what teams competed in secondary tournaments such as the NIT or Vegas 16 for example. Like the `RegularSeasonCompactResults.csv` document, this file contains the same attributes per entry, though with another column for what secondary tournament this game was played in. It is also noted that games that are played after $DayNum = 132$, these games are not listed in the regular season results file.
- `TeamSpellings.csv` - Contains information about the spelling of a team name based on the team ID. This can help group results such as ”ball st” and ”ball-st” as a direct group and match on these values will not work.
- `NCAATourneySlots` and `NCAATourneySeedRoundSlots.csv` - Define the slots in which games take place within the tournament. These are shorter codes that define a game based off the ranking of the highest seed being played. For example, if two games are played R1W1 and R1W8, these decode as Round 1 (R1) West region seed 1 (W1) and round 1 west region seed 8. We can then tell for the subsequent game R2W1 since we take the higher seed, this game is the round 2 playoff between west seed 1 and west seed 8. In these datasets, the strong and weak seeds are defined, though the higher of the two seeds are displayed in the shorthand. It is noted that in later rounds, this discrepancy may not be as important as potential 1 seeds from different regions play against each other. The second file only specifies the slots and the time(s) of the day that the game can be played. It shows the seed of the teams, game round and slow and lastly the earliest and latest day that the games can be played on.

3 Methodology

For this project, our goal was to apply three different data mining regression techniques to the dataset of March Madness games and seeds with the intention of finding which regressor would perform the best, and studying why that regressor performed better than the other ones. We use regression as we want our models to predict the probability of the home team winning given data on a single match. The regressors chosen for this project were the Artificial Neural Network (more commonly referred to as a Neural Network), the Random Forest tree classifier, and the XGBoost regressor.

Before training the relevant data on the three regressors, it was important to perform preprocessing on the data in order to be suitable for analysis once the models were trained on the data. This involved many steps, which will be explained more in-depth below. Once the data was prepared for training, each of the three regressors were optimized using k-fold cross-validation to find the best hyperparameters to train the data on. The data was split into a train/test split before training, and once the data was trained on the models, the models were evaluated on certain regression metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). These metrics help determine the standard deviation between the predicted win percentage for teams and the actual win percentages. An important note is that for the statistical error metrics, the data used to train the models was based on average statistics for teams compiled over several seasons.

In addition to calculating error metrics on the three regressors, the regressors were also tested on the most recent NCAA tournament seeds by running simulations of March Madness brackets against the regressors to see how accurate or similar the brackets would be compared to the ground truth.

3.1 Data Pre-processing

In the data provided, there are rudimentary statistics such as 3 Pointers Made (3PM) and Field Goals Attempted (FGA), but we also wanted to utilize more advanced statistics such as 3 Point Percentage (FG%) and Effective Field Goal Percentage (eFG%). eFG% is an important statistic since it weights 3PM heavier than other field goals. The calculation is as follows: eFG% = $\frac{(0.5\times 3PM)+FGM}{FGA}$. Another important statistic that we utilized as a feature was turnover differential, which is the amount of times a team turned the ball over subtracted from the number of times they get a turnover from their opponent. We conducted feature engineering to add these statistics based on domain knowledge of the game in order to possibly yield better results when training and testing our models. In addition, missing values were imputed, which is explained more in-depth below, and the data used for training and testing was split into a 90/10 train/test split.

3.2 Random Forest

Random Forest is an ensemble learning method used for both classification and regression tasks. For this problem, we used it for regression purposes to predict a numerical win percentage value as a decimal. It operates by constructing a multitude of decision trees during training and outputs the mode of the classes of the individual trees. Each decision tree is trained on a random subset of the training data, and at each node of the tree, a random subset of features is considered for splitting picking the split that reduces the residual sum of squares value. This randomness injects diversity into the ensemble, preventing overfitting and increasing the robustness of the model.

Since the Python RandomForestRegressor cannot handle missing values in the data, the dataframe is first preprocessed by imputing missing values using the KNNImputer, which utilizes the knearest neighbors algorithm to estimate missing feature values based on the values of neighboring samples. The value of the neighborhood was set to 5 as this was just picked based off prior knowledge and not based on any other computational measures. Next, a Randomized Grid Search is performed to find the optimal hyperparameters for the Random Forest Regressor. The hyperparameters include the number of trees in the forest (n_estimators), the number of features to consider when looking for the best split (max_features), the maximum depth of the tree (max_depth), the minimum number of samples required to split an internal node (min_samples_split), and the minimum number of samples required to be at a leaf node (min_samples_leaf). After finding the best parameters, the model is trained on the training set, and predictions are made on the test set. This process allows for the creation of an optimized Random Forest model capable of accurately predicting the win percentage of basketball teams based on the various features in the dataframe.

3.3 XGBoost

XGBoost, which is short for eXtreme Gradient Boosting, is a gradient boosting library that is used for both classification and regression. In this project, XGBoost was used for regression in order to predict numerical win percentage values for teams, as well as classification in order to classify which teams would win or lose in simulated brackets. XGBoost works by iteratively training an ensemble of decision trees, where each subsequent tree corrects the mistakes that the previous trees made. It also employs a regularized objective function that penalizes complexity so the model doesn't overfit on training data. XGBoost also utilizes an algorithmic approach knwon as gradient boosting, which enhances model performance by capturing interactions between features.

As done with the Random Forest, a randomized Grid Search was performed to find the optimal hyperparameters for the XGB regressor. The hyperparameters include min_child_weight, which is the minimum sum of instance weight or Hessian needed in a child node, *gamma*, the minimum loss reduction required to make a further partition on a leaf node of the tree, *subsample*, which is the subsample ratio of the training instances, $colsample\_bytree$ which is the subsample ratio of columns when constructing each tree, and finally max_depth, which is simply the maximum depth of each tree constructed. After finding the best parameters, the model is trained on the training set, and predictions for win percentages are made on the test set. In addition, the same model is used to run a simulated tournament to see how accurate the bracket it creates is.

3.4 Neural Network

The neural network used was a fully connected network with 30 input features, a single hidden layer with 15 nodes and a single output node. From the neural network, a value between -1 and 1 is returned indicating which team is predicted to be the winner in a head to head match up.

With the historical data being pre-processed, each teams season normalized data was merged with the competitor's team season statistics creating our 30 input matrix with the first 15 elements being the first teams stats, and the last 15 features being the second teams stats. As teams change from year to year, the team's statistics needed to be separated to more accurately capture the features and "type" of a team. From 2010 and later, all match-ups are split into a an 80% training set then a 20% testing set. The testing set is then separated into randomly sorted batches for each epoch of training in the neural network. Lastly, the network has an early stopping point if the `val_loss` has reached a local minima and not improved after 20 epochs. From this point, the neural network should be trained to determine a winner based off the seasonal characteristics of a team to determine a winner.

Like previously mentioned, the neural network is a fully connected or dense network with 30 input features, 1 hidden layer with 15 nodes and a single output. The first hidden layer uses ReLu for its activation function and the final output mode uses ”tanh” for its activation function.

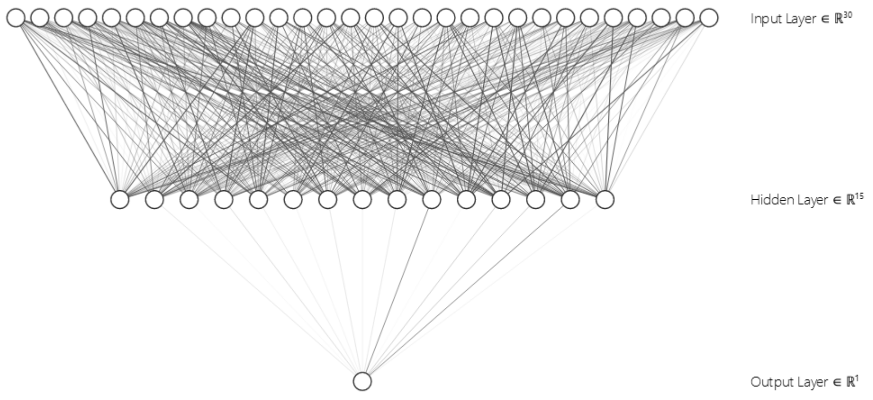
Input Layer ∈ $\mathbb{R}^{30}$
Hidden Layer ∈ $\mathbb{R}^{15}$
Output Layer ∈ $\mathbb{R}^{1}$

Figure 1: Demo Neural Network visualization of similar shape

For the tournament, each of the teams that made the tournament, their 2024 statistics were put into a function that runs the tournament based of the seeding and regions specified in `MNCAATourneySeedRoundSlots.csv` going through the same structure of the real tournament in a single elimination game but without the first four games using the neural network.

4 Results

4.1 Model Performance Comparison

In order to assess the efficacy of various regressors for the given task of predicting win percentages and the accuracy in classifying brackets, we utilized statistical error metrics as a benchmark for predicting win percentages, and the accuracy with which regressors were able to create brackets compared to the ground truth of the 2024 NCAA March Madness Bracket. The statistical error metrics values across the regressors are shown below in Figure 4:

Statistical Error Values by Regressor			
Statistical Error Metric	Random Forest Classifier	XGBoost Regressor	Neural Network
MAE	0.16096	0.00339	0.87436
MSE	0.06171	3.68703e-05	0.87040
RMSE	0.24841	0.00607	0.93295

Figure 2: Statistical Error Metrics for the Regressors

After the tournament was simulated with the different regressors, the accuracy was calculated by comparing the bracket created by the regressors against the ground truth of the 2024 March

Madness bracket. The accuracy across regressors for the simulated tournament is shown below in Figure 5:

Tournament Accuracy % by Classifier			
	Random Forest Classifier	XGBoost Regressor	Neural Network
Acc. (%)	66.6667	66.6667	58.7301

Figure 3: Accuracy % for the regressors on the Simulated Tournament

Further evaluation shows that XGBoost performed the best out of the three in terms of its low statistical error values, whereas the Neural Network performed the worst, with significantly higher error values compared to XGBoost and the Random Forest classifier. Similarly, the neural network showcased the worst accuracy in terms of predicting the 2024 March Madness bracket with a lower percentage of 58.7%, whereas the Random Forest classifier and XGBoost were tied for the best accuracy. The accuracy of a model's accuracy refers to how similar the model's bracket is to the ground truth bracket we created.

We hypothesize that the reason for the poor performance of the neural network is due to its inability to understand complex relationships in the data, and simply overfit on the training data. The Random Forest and XGBoost algorithms are more robust in dealing with overfitting, as they both employ ensemble learning and regularization techniques in order to improve generalization performance. The strong performance of XGBoost was expected, as the winner of the Kaggle competition for the 2023 March Madness Competition used a fine-tuned XGBoost algorithm.

4.2 Limitations and Future Work

The biggest limitation with this project was the availability of the data in order to classify brackets and predict the winners of matches. While there was a plethora of statistics for all NCAA teams spanning many seasons, sports events like the March Madness tournament are inherently dynamic, and are heavily dependent on factors that are not captured in the data. For example, crucial factors that leads to teams winning or losing matches such as player transfers, coaching changes, team chemistry, or injuries, are not captured in the dataset, and so models that are not trained on those features might not be able to accurately predict March Madness brackets. Future work would involve creating features or datasets that are able to capture these factors and input them to certain models in order to better predict March Madness brackets.

Moreover, we can further explore the multiple statistics provided in our data. With further examination of our problem statement, we find our initial assessment on the importance of turnover difference to be incorrect. Instead, Random Forest regressor finds average field goals made to be the most important statistic in determining the winning team. We need to explore if this statistic makes the problem trivial (similar to how a match score eliminates the need for prediction).

Based on existing results from [3], we know that the performance of our models can be further improved. Top-performing models for this challenge have employed the use of ELO ratings in which were provided in the dataset in combination with XGBoost to make predictions. This can be an interesting direction to explore to improve the performance of our models.

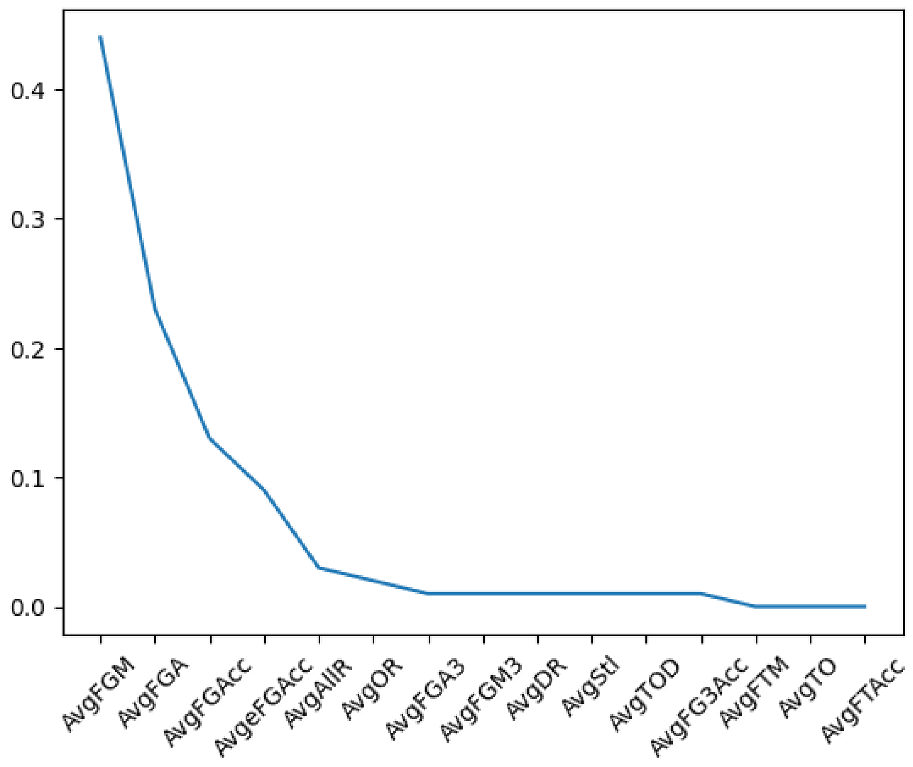

Figure 4: Graph of metric vs metric importance obtained from Random Forest regressor after training.

5 Conclusion

In our March Madness analysis, we utilized past college basketball data, domain knowledge, and data preprocessing techniques to train three different types of regressors, Random Forest, XGBoost Regressor, and a neural network. We found that XGBoost and the Random Forest had the highest percentage accuracy of 66.6667 and the neural network had 58.7301% accuracy. We also utilized mean squared error, root mean squared error, and mean absolute error to measure performance of the models. XGBoost had the least error for all three measures and the neural network had the most error for all three measures. From this performance, we found that XGBoost was the best model due to its resistance to overfitting and good performance with generalization. Future work for the project will include incorporating new features such as team chemistry, player/coaching changes, and injuries to better represent complex relationships in the data and predict March Madness brackets.

6 GitHub

The link to the GitHub and our code is here: https://github.umn.edu/waria012/5523-final-project.

References

[1] Mike Benzie. The longest an ncaa bracket has ever stayed perfect, 2024.

[2] Jeff Bergen. A quantum leap for basketball bracketology. *Inside Science*, 2013.

[3] Addison Howard Anju Kandru Jeff Sonas, Ryan Holbrook. March machine learning mania 2024, 2024.