A Machine Learning Approach to March Madness*

Anonymized

Abstract

The aim of this experiment was to learn which learning model, feature selection technique, ordering process, and distance criteria provided the best classification accuracy in predicting the outcome of any NCAA Men's Basketball Tournament match. Ninety-four features were selected from ESPN.com for each team accepted into the tournament for the last four years. Ordering processes were tested against the baseline and the random ordering increased classification accuracy from 0.61 to 0.63. Random ordering was used to test a variety of feature reduction techniques. Random forest feature reduction performed best and increased accuracy from 0.63 to 0.7322 when used in conjunction with kNN and limiting the number of features to five. Using Manahattan distances as opposed to Euclidean distance further increased accuracy from 0.7322 to 0.7362.

1 Introduction

Since 1939, the best colleges and universities across the United States have participated in a yearly tournament called the NCAA Men's Basketball Championship. This basketball tournament has become one of the most popular and famous sporting tournaments in the United States. Millions of people around the country participate in contests in which the participants make their best guesses on who will win each game throughout the tournament. These types of contests have become so popular that more than 11 million brackets were filled out on ESPN.com in 2014. One billion dollars was even being rewarded to anyone who achieved a "perfect bracket" (guessed every game correctly).

Every game is unpredictable, and the teams that are supposedly the "better team" sometimes end up losing. This is called an upset in basketball lingo, and happens regularly throughout the tournament. Because of these upsets, it can be difficult to correctly guess the winner of each game. The tournament can be so unpredictable that the time period over which the tournament runs has been termed March Madness.

Since there are 64 games played in the NCAA tournament, it is nearly impossible to predict a perfect bracket. High Point Enterprise, a morning paper from North Carolina, stated that "you have a much greater chance of winning the lottery, shooting a hole-in-one in golf or being struck by lightning". They estimated that the chances of predicting a perfect bracket are 1 in 9.2 quintillion.

It became clear that developing a model that provided perfect win/loss classification was unrealistic, so instead we focused on improving the prediction accuracy of individual games. Data was collected from a number of sources to help with the learning. Certain organizations such as ESPN.com and NCAA.com keep mounds of statistical information on every team throughout the regular season. By collecting these statistical measurements and running them through numerous learning models, prediction accuracy could drastically improve.

Note that the problem at hand is not classification of individual teams, but rather predicting the outcome of a match between any two teams. We discuss the impact of this distinction and our steps to deal with it in the methods section.

2 Methods

2.1 Data Source

We selected our data from the ESPN.com website, one of the largest and most popular sources for sports news in general, and particularly for information about March Madness. ESPN.com is listed as the third search result in Google (after two ncaa.com results) for the query "march madness". ESPN.com not only publishes the results of the March Madness tournament, but also keeps track of a vast number of statistics about each team and player that participates in the NCAA tournament. Each team is associated with a unique ID on ESPN.com, making associating match results with team statistics trivial.

Though the NCAA March Madness competition has existed for many decades, ESPN changed the way that they calculate the statistics in 2010, leaving us with 4 years of data. Each championship contains 64 matches, yielding nearly 300 matches for evaluation, each match representing a single instance in our machine learning problem. Also included in the instance are the statistics provided by ESPN.com for the two competing teams, of which there are 47 per team. This re-

*These match the formatting instructions of IJCAI-07. The support of IJCAI, Inc. is acknowledged.

sults in 94 features per instance, of which several features are derived from other features.

We considered growing the data set by including matches that occurred during the season preceding the tournament, but given the specificity of the problem, we decided that the addition of such instances would not help the algorithm to generalize on championship games. Additionally, many of the features taken from ESPN are calculated based on the team's performance during the preceding season, which would not apply to matches that occurred during that season.

2.2 Selected Models

Given that the comparison classification problem is sufficiently different from typical problems, we wanted to evaluate several different learning algorithms to see which would fare the best. All of the input features are continuous, with a boolean output class, which fits many common algorithms well.

Accordingly, we selected Naive Bayes, kNN, Decision Tree, SVM, CN2 Induction, Random Forest, Logistic Regresison, and Backpropagated Neural Network to form our base set of models. Additionally, we ran all tests with a simple learner that output the majority class (stochastically) to form a baseline.

Table 1: Feature Descriptions

Feature Name	Feature Description
2P	Two point field goals made for season
2PA	Two point field goal attempts for season
2PM	Two point field goals made for season
3P	Total three point fields for season
3PA	Three point attempts
3PM	Three pointers made for season
APG	Assists per game
ASM	Average scoring margin
AST	Assists for season
AST/TO	Assists to turnover ratio
BLK	Blocks for season
BLK/PF	Blocks to personal foul ratio
BLKPG	Blocks per game
CFRP	Conference RPI
CFSS	Conference strength of schedule
DEF	Defensive rating
DEFQ	Defensive Quotient
DRPG	Defensive rebounds per game
FG	Field goal percentage
FT	Total free throws made for season
FTA	Total free throw attempts for season
FTM	Free throws made for season
GP	Games played for season
LRPI	Road and neutral game RPI
NCRP	Non-conference RPI
NCSS	Non conference strength of schedule
OFF	Offensive rating
OFFQ	Offensive quotient
ORPG	Average offensive rebounds per game
PF	Personal fouls for season
PPG	Points per game
PPS	Points per shot
RK	Rank at the end of the season
RPG	Rebounds per game
RPI	Rating percentage index
SOS	Strength of schedule
ST/TO	Steals to turnover ratio
STPG	Steals per game
TOPG	Turnovers per game

These statistics refer to teams' performances as a whole and do not pertain to the performance of individual players.

3 Initial Results

For our initial results, we ran all of the learners with their default parameters (as supplied by the orange computation platform) using all 94 features and a feature-based ordering scheme where the "team1" was the team with the lower "strength of schedule" score. This yielded abysmal results; the best learner was kNN, which achieved 61% accuracy, barely above the baseline of 60%. All of the other learners tied with the baseline or did significantly worse.

4 Improvements

4.1 The Problem of Comparison

One of the features of this machine learning task is that its goal is comparison between two items (in this case teams), where for each team there are a number of features associated with it. The problem of classification, and specifically binary classification, is to determine whether an instance A is one of two classes, for example X or Y. What's unique about the

situation of comparison is that each instance A is comprised of an ordered pair of "teams" B and C. For every instance $BC => X$, the reversed pair CB will always Y, and vice versa.

This adds a layer of complexity to what the learner must learn; for every logical input feature f, f_A and f_B are implicitly linked and their reverse is linked to the reverse of the outcome.

The setup of the learners forces B and C to be ordered, though the data set is not inherently ordered; for each game there are just two teams and an indication of which team won. An obvious option would be to put the winner first and the loser second, but then all classifiers would only classify the outcome as "WIN"; the ordering must be something that can be trivially determined for novel data.

We investigated several methods of approaching this issue, including feature-based ordering, arbitrary ordering, reverse duplication, and random ordering.

Feature-based Ordering

Initially, we just sorted the teams based on a single feature that we thought was most useful, the "strength of schedule" feature that is decided by ESPN.com based on a secret algorithm known only to them. This feature generally has a lot of weight when people build their brackets by hand, and so we tried ordering the teams such that the team with a lower "strength of schedule" came first.

Arbitrary Ordering

Our second approach was to order the teams alphabetically, such that the team to come "first" was the one whose name came first alphabetically. This was arbitrary, but not random; it introduced a bias that was completely unrelated to the problem at hand.

Random Ordering

A third approach was to order the teams randomly – essentially remove any bias from the ordering. Removing that bias would mean one less thing that a learner has to discover. This doesn't help them to understand that there is reversability in the data set, however, which could potentially damage results.

Data Doubling

The final method was to insert both orderings; both $BC => X$ and $CB => Y$. This would give learners the most possible information and hopefully also remove any bias. While testing learners, pairs from the test set were excluded from the test set. If BC were left in the training set when testing on CB, the learners achieved nearly 100% accuracy.

Comparison of Orderings

When tested with the default settings of eight different machine learning algorithms, the different ordering schemes resulted in significant differences in accuracy. Figure 1 shows the absolute accuracy of each learner for each of the four different ordering schemes, and Figure 2 shows the relative accuracy compared with a baseline learner (majority) for that ordering. Accuracy was determined using 10-fold cross validation.

In this comparison, difference from the baseline is a more useful metric than absolute performance. In a case where the ordering resulted 90% of the instances being the same class, many learners would doubtless do very well, perhaps in the 80-90% range. This is far less impressive, however, than a learner that achieves 60% accuracy on a data set that is split 50-50.

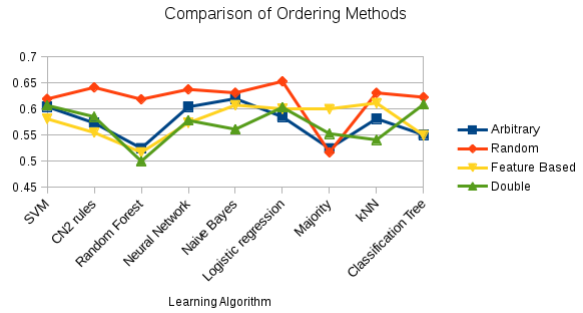

Figure 1: Performance of Various Learners on Different Orderings

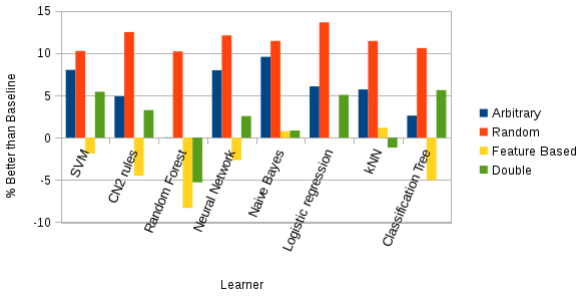

Figure 2: Performance of Various Learners against the Baseline

The large amount of variance for a given learner on the different orderings indicates that ordering does have a large impact on a learner's ability to learn the problem. The various learners reacted differently to each of the orderings, as some were better able to cope with or take advantage of the bias introduced by a given scheme.

For the arbitrary (alphabetical by team name) ordering, almost all of the learners were more accurate than the baseline. Because the ordering was fairly uncorrelated with the learning goal, there wasn't much of an extra bias that had to be overcome. However, it still fared worse than the random ordering, which was (by definition) uncorrelated and unbiased.

Feature-based ordering was least helpful; almost all of the learners did worse than the baseline. The "strength of schedule" feature was more correlated with the learning goal than any other ordering; it resulted in 60-40% win-loss split in our data set. The majority learner therefor achieved 60% accuracy, making all others comparatively much worse.

Random ordering was most effective across the board, both in terms of absolute accuracy and in terms of improvement

over the baseline. Given that none of the algorithms used are designed to take advantage of ordering bias in a comparison instance, the best method was to just remove the bias.

Data doubling was surprisingly ineffective. Even though there were twice as many instances for a learner to learn from and the ordering bias was effectively neutralized, the problem was apparently just as hard to learn as when there was ordering bias.

4.2 Feature Reduction

From an initial feature space of 94 dimensions, we tested several different methods of reduction, in which we took the top n attributes as ranked by four different algorithms: Gain Ratio, ReliefF, Linear SVM Weights, and Random Forests. For each algorithm we tested taking different numbers of attributes (always the top n) for each of the learners. Figure 3 shows the best accuracy achieved by each learner for each reduction technique, for $n < 45$.

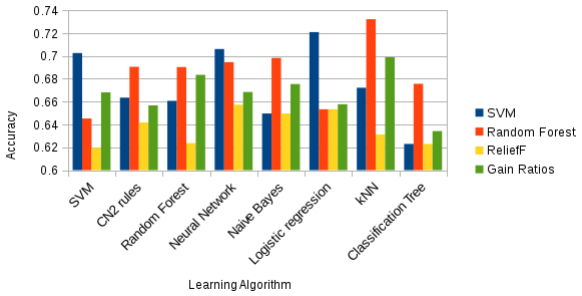

Figure 3: Comparison of Reduction Methods

ReliefF scoring proved to be the least helpful, and was the only method that actually decreased accuracy when the number of features was reduced; the other three methods managed to achieve significant gains over the full feature set.

Using Gain Ratio to reduce feature size resulted in modest accuracy improvements for over half of the learners. However, it was only able to reduce the feature set size down to about 15 before all the learners began to lose accuracy dramatically.

The two reduction methods that involved more complex computation resulted in the most impressive accuracy gains. They also succeeded in reducing the number of features by over 80%.

Reduction using the Support Vector Machine resulted in dramatic improvements for several learners. The SVM learner improved by 8% and the neural network and logistic regression algorithms both improved by 6%. In each case, the feature space was reduced from 94 features to between 5 and 10, dramatically speeding up learning time as well. It is interesting that the SVM learner improved when using SVM preprocessing. The second learner was apparently able to benefit from the work done by the previous one, while at the same time exploring new territory to achieve better accuracy.

Random forest feature reduction proved to be the most successful, both in terms of accuracy and in the amount of reduction achieved. The kNN algorithm was boosted by 10% when run on only the top 5 features from the random forest weights—a 95% reduction in feature space size. All except logistic regression and SVM achieved similar improvements in accuracy. As was the case with SVM reduction, the Random Forest reduction resulted in a significant improvement for the Random Forest algorithm.

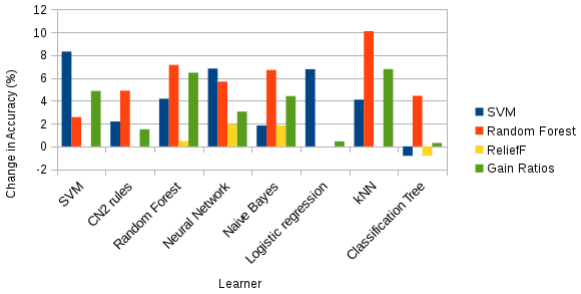

Figure 4: Benefit of Reduction to Individual Learners

4.3 Model Tweaks

The kNN learner achieved the greatest accuracy after applying feature reduction, so we chose to focus on it for more granular parameter adjustment. We experimented with two different distance metrics, Euclidean and Manhattan, at varying values of k to isolate the ideal parameters. Results are displayed in Figure 5. The greatest accuracy (73.62%) was reached by using the Manhattan distance metric and a k value of 130.

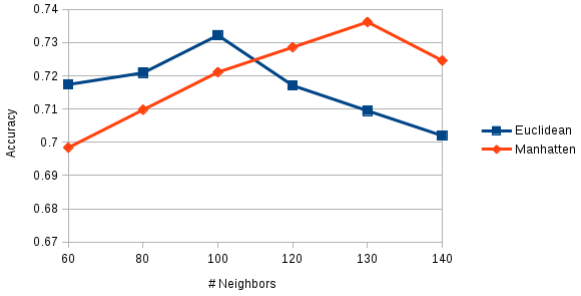

Figure 5: Nearest Neighbor Accuracies by # Neighbors

5 Results

Numerous tests were performed altering the ordering sequence, training model, features, and type of distance (Euclidean vs. Manhattan). The data was trained using SVM, CN2 rules, neural network, classification tree, logistic regression, naive bayes, random forest, k-nearest neighbor, or simple majority. The features were selected according to those that provided the highest SVM weights. In other words, if the data set was training using 10 features, then the 10 features with the highest SVM weights were used. Increasing classification accuracy (CA) was our primary goal.

5.1 Ordering

Each ordering technique was run with each learning model. The results can be seen on table 2.

	Arbitrary	Random	Feature	Double
SVM	0.6047	0.6195	0.5813	0.6071
CN2 rules	0.5734	0.6417	0.555	0.5854
Random Forest	0.5249	0.6189	0.5171	0.5
Neural Network	0.6043	0.6379	0.5736	0.5783
Naive Bayes	0.6201	0.6313	0.6077	0.5613
Logistic regression	0.5852	0.6533	0.6004	0.6034
Majority	0.5246	0.517	0.6001	0.5528
kNN	0.5816	0.6312	0.6117	0.5411
Classification Tree	0.5506	0.6229	0.55	0.609

Table 2: Ordering influence on training models

Random ordering gave us our best results for every training model. The classification accuracy increased from the baseline of 0.61 to 0.63. Other ordering techniques introduced a type of bias that random ordering did not introduce. These results were attained without implementing any feature reduction techniques.

5.2 Feature Reduction

SVM

Features were chosen according to their SVM weights. If only ten features were selected, the top ten features with the highest SVM weights were used. Random ordering was used in creating each instance.

Model	Top N Features				
	All	35	10	9	5
SVM	0.619	0.679	0.702	0.698	0.588
CN2 rules	0.641	0.663	0.562	0.592	0.569
Random Forest	0.618	0.660	0.611	0.641	0.551
Neural Network	0.637	0.667	0.687	0.706	0.641
Naive Bayes	0.631	0.645	0.634	0.623	0.6
Logistic regression	0.653	0.668	0.691	0.720	0.607
kNN	0.631	0.672	0.656	0.657	0.603
Classification Tree	0.622	0.555	0.547	0.570	0.572

Table 3: Features were reduced using the best features according to SVM weights.

Random Forest

Features were chosen by constructing a large number of decision trees and choosing the features that averaged the most importance among the decision trees. These results can be viewed in Table 4.

ReliefF

Instances are chosen at random and changes the weights of feature relevance according to its nearest neighbor. Table 7 shows the results using ReliefF scoring.

Model	Top N Features			
	All	15	5	4
SVM	0.6195	0.6452	0.6228	0.6226
CN2 rules	0.6417	0.6687	0.6835	0.6905
Random Forest	0.6189	0.6903	0.6719	0.6571
Neural Network	0.6379	0.6493	0.6942	0.6946
Naive Bayes	0.6313	0.6755	0.6909	0.6835
Logistic regression	0.6533	0.6379	0.6187	0.6187
kNN	0.6312	0.6872	0.7322	0.7024
Classification Tree	0.6229	0.5774	0.6756	0.5936

Table 4: Features were selected by selecting those features most important in a random forest

Model	Top N Features			
	All	65	15	5
SVM	0.6195	0.6085	0.5991	0.5651
CN2 rules	0.6417	0.5813	0.5969	0.4828
Random Forest	0.6189	0.616	0.5665	0.5132
Neural Network	0.6379	0.6303	0.6305	0.5694
Naive Bayes	0.6313	0.6496	0.6074	0.566
Logistic regression	0.6533	0.6528	0.6377	0.5991
kNN	0.6312	0.6199	0.5701	0.5963
Classification Tree	0.6229	0.5595	0.5661	0.5131

Table 5: Features selected using ReliefF scoring

Gain Ratio

Calculates features that provide the most gain, but does not take combination of features into account. Table 8 shows feature selection using gain ratio.

Model	Top N Features			
	All	35	15	5
SVM	0.6195	0.6566	0.6298	0.6148
CN2 rules	0.6417	0.6342	0.6567	0.5698
Random Forest	0.6189	0.6386	0.6835	0.5922
Neural Network	0.6379	0.6224	0.6684	0.615
Naive Bayes	0.6313	0.6642	0.6528	0.6225
Logistic regression	0.6533	0.6349	0.6115	0.6074
kNN	0.6312	0.6952	0.6989	0.6679
Classification Tree	0.6229	0.6342	0.5671	0.5775

Table 6: Features selected using gain ratio.

Feature Reduction Summary

Random forest provided the best results when used in conjunction with kNN and limiting the number of features used in the training to five. The classification accuracy increased to from 0.63 to 0.7322, adding more than 10% of additional accuracy.

5.3 Euclidean Distance vs. Manhattan Distance

The results on Table 9 show the differences between using Euclidean Distance and Manhattan Distance on k-nearest

neighbor training on a randomly ordered instances and limiting the number of features used in training to five.

# Neighbors	Euclidean	Manhattan
60	0.7174	0.6984
80	0.7209	0.7098
100	0.7322	0.7211
120	0.7171	0.7286
130	0.7095	0.7362
140	0.7020	0.7246

Table 7: Nearest Neighbor Distance Algorithms and K values

Using a Manhattan distance and the nearest 130 neighbors increased our previous best result from 0.7322 to 0.7362.

6 Conclusion

The results of our experiments show that random ordering used in conjunction with k-nearest neighbor, Manhattan distance, and random forest as the feature reduction algorithm provide the best classification accuracy. Our best classification accuracy, using each of these algorithms, was 0.7362. The following sections describe possible causes about why these features provided the best results among all experiments that were tested.

6.1 Style Theory

In basketball, it is believed that certain teams match-up better against certain teams. For example, one team may be extremely quick while another team is extremely tall. If the first team can alter the game so that it is played at a fast pace, then the game style is in their favor. Essentially, specialties on a basketball team translate to success against opponents with certain characteristics.

6.2 K-nearest Neighbor & Style Theory

K-nearest neighbor provided the best results in comparison to the other training models most likely due to this theory. The data compares itself to the instances that provide similar data. In other words, quicker teams that play taller teams will compare themselves to other match-ups that involved quicker teams playing taller teams. The consistencies of the match-ups and their associated outcomes will mean more to an instance that shares those characteristics.

6.3 Combination of Features Theory

Some features, when combined, prove to be more beneficial to the success of a team than others. For example, a team that has a low number of steals per game, shoots a large number of field goals, and shoots a high field goal percentage will have more success than a team with a combination of large number of steals, high field goal percentage, and low number of shots taken each game. Learning models that take into account the combination of features when classifying instances will perform better than those that do not.

6.4 Random Forest and Combination of Features Theory

Using random forest takes into account how features interact one with another. This allowed us to use those features that provided the best results when used in collaboration with other features. Thus, we were better able to compensate for the importance of a combination of features and use that as part of the training process.

6.5 Perfect Bracket Probabilities

Even with large amounts of statistical information, there are some parts of a basketball game you simply cannot predict. Injuries cannot be forecast, and the mental state of players playing the game cannot be observed. However, these things can drastically affect the outcome of a basketball match. Perfect brackets will therefore continue to be impossible to calculate, no matter how much data is provided to the learning models.

7 Future Work

We believe that our classification accuracy can improve to 85% if the proper features are used in training. Vital statistical information we were unable to gather include the following:

- Individual statistics for players on each team
- Venue of the match including the distance
- Importance of a match-up
- Win/loss record for the last five, ten, and fifteen games
- Average number of players who play consistently

All of this data is expected to play a role in how teams perform. However, this type of data was out of our scope to accumulate. Organizations with more access to this type of data will be able to study whether these features have a profound effect on the success of a basketball team.

This research would also benefit greatly with more data. Since only the statistics for the last four tournaments were accessible in a consistent format, our training models risked the danger of overfit. We would recommend that much more data be gathered to help avoid overfit. However, since basketball continually changes throughout the years, we recommend not use data from more than ten years back.

We would also recommend experimenting with using several feature reduction techniques together as opposed to using the reduction techniques one by one.