Making Sense of the Mayhem: Machine Learning and March Madness

Alex Tran and Adam Ginzberg

Stanford University
atran3@stanford.edu ginzberg@stanford.edu

I. Introduction

The goal of our research was to be able to accurately predict the outcome of every matchup in a March Madness bracket. This is an extraordinarily difficult problem because of the high amount of variance in college basketball and the sheer number of games that are played in the tournament. In the history of March Madness, no one has ever created a perfect bracket. Last year, Warren Buffett agreed to give $1 billion to anyone that submitted a perfect bracket to the Yahoo Bracket Challenge and $20,000 to the top 20 best performing brackets. It only took 25 games for the last perfect bracket in the challenge to be eliminated. We wanted to see how a well formulated machine learning model and algorithm would perform on this problem compared with the success of human-generated brackets. Our model hoped to produce the highest performing bracket as defined by the Yahoo Bracket Challenge.

II. Data

We scraped ESPN for the tournament game data for the last four years of March Madness. We then scraped over 50 features relating to each team. For each year we had 63 training examples, the total number of matchups in the tournament. We randomly decided which team in a matchup to list as Team 1, and then we classified the specific matchup with label y if Team 1 won or lost (1 or 0 respectively). This structuring gives us 252 training examples in total with approximately half having output label 1 and half having output label 0.

III. Model

We modeled our problem as a classification problem. We utilized supervised learning to train on a set of previous tournament games and then aimed to classify future games as either a win or a loss for the indicated team. We characterized each game using a number of features relating to each team. The first main set of features that we used were seasonal game statistics such as field goals, rebounds, assists, etc. This information established a base profile for a team, and provided a reasonable estimate as to how a team performed on a game to game basis. The second set of features we used were more complex metrics including strength of schedule ranking, margin of victory, win % in last 12 games, etc. These statistics when used in combination with our seasonal game statistics proved very useful because they were able to account for the fact that college basketball teams often have vastly different schedules and play disjoint sets of teams. See the appendix for a comprehensive list of the features that were available to us.

We implemented a feature extractor that served several functions for our model. First, our feature extractor returned a dense map of the feature values because a feature vector which is too large proves to be unwieldy for an algorithm such as logistic regression. Then, our feature extractor normalized all of the seasonal game statistics by dividing by the number of games played so that every team's statistics could be interpreted in the same context. Because of the limited size of our dataset, we then chose to use the difference of team feature val-

ues (where appropriate) in order to reduce the size of our feature vector. We then rescaled all of the features to roughly fall into the range [0, 1] so that each feature would have equal footing and so that an algorithm such as logistic regression would converge more quickly. Finally, we derived several new features from our original raw feature values. For example, in order to account for the quality of a given team's schedule when weighting its record, we created a feature which multiplied the team's strength of schedule ranking by its win percentage taking into account potential interaction between these two features.

Until now, we have only discussed the formulation of our training examples in describing our model. However, another critical aspect of our model was our scoring function. Recall that our goal was to optimize the score our bracket would receive when submitted to the Yahoo Bracket Challenge. Thus, when creating a bracket, any decision that one makes in the earlier rounds must persist throughout the tournament. Thus incorrectly predicting a single upset in the first round can completely ruin a bracket because the incorrectly predicted team will appear in future matchups while the actual advancing team will not. In addition because our objective was to optimize the score achieved by the Yahoo Bracket Challenge, we evaluated the success of our bracket by summing over the scores associated with every correct classification according to this function: $\text{Score} = 10 \cdot 2^{round-1}$. This scoring function gives exponentially higher weight to correctly predicting matchups in the later rounds of the tournament. We chose to maximize total score rather than minimize test error.

We now briefly describe the baseline and oracle that we used in comparison with our model. Our baseline classified the team with the higher seed as the victor (breaking ties arbitrarily). This baseline is a very natural choice because it follows the expectation of the selection committee with respect to each team subject to random noise due to predetermined seeding rules. The baseline certainly does not perform poorly, it typically averaged among the 70^{th} percentile of human brackets; however it remains a naive approach to bracket construction. For our oracle we chose the highest performing human bracket for a given year. This bracket is the best out of millions of entries and reflects an upper bound on the score we hope to obtain.

IV. Algorithms

The two main algorithms that we used were logistic regression and an SVM.

We briefly mention how we approached training and testing over the data. We used k-fold cross validation. This method segmented our data into 3 tournaments to be used as training data and 1 tournament to be used as testing data. We trained our algorithms on every possible combination of the 3 tournaments, and then tested each on the corresponding tournament that was left out. We then output the average of our training error, test error, and points scored over all 4 folds.

For logistic regression, we trained the model on our training data using stochastic gradient descent. We chose to run logistic regression on our data because it is a relatively simple algorithm that is quite effective. Logistic regression creates a linear decision boundary. This property is both an advantage and disadvantage of the algorithm. Because of its simplicity logistic regression is unlikely to overfit the data; however, at the same time the algorithm has a relatively low ceiling in terms of performance at least in the base dimension of our feature space. This is a problem of high bias. We used annealing to reduce our learning rate after every update to our parameters. The update rule for logistic regression:

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

For Support Vector Machine, we utilized the library LIBSVM to run the algorithm on our data. We worked in an infinite dimensional feature space because we used the Gaussian kernel. This is the primary advantage of SVM: working in an infinite dimensional feature space and using the kernel trick allowed

us to capture the interaction between features. This is particularly valuable for our dataset. One disadvantage of SVM is that it is susceptible to overfitting because in working in an infinite dimensional feature space it is extremely likely that the extracted data will become linearly separable and hence the model will learn hidden attributes of our data rather than of the problem more generally. To fine tune our use of SVM, we ran grid search and cross validation to find the optimal cost C and γ parameters which are fixed when we optimize the primal problem:

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y^{(i)}(\mathbf{w}^T\phi(x^{(i)}) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0. \end{aligned}$$

with Gaussian kernel:

$$K(x^{(i)}, x^{(j)}) = e^{(-\gamma\|x^{(i)} - x^{(j)}\|^2)}$$

V. Results

Our results are in Table 1.

For logistic regression, we used annealing to reduce the learning rate after every iteration according to the equation:

$$\alpha = \frac{1}{\sqrt{\#\text{ of updates made so far}}}$$

See Figure 1 for a graph of the learning rate.

We were quite pleased with the results of logistic regression. We made significant progress on the baseline, outperforming it for every fold. In addition, for folds 2 and 3 we were able to correctly predict the winner of the entire tournament, achieving impressively high scores of 1580 and 1160. These folds corresponded to the years 2012 and 2013 respectively which had far fewer upset teams reaching the later rounds. Folds 1 and 4 marginally outperformed the baseline but the results were not too impressive, in those two years the tournament results seemed to completely defy reason. For example, last year the two teams in the tournament final were 7 and 8 seeds, an extremely unlikely event.

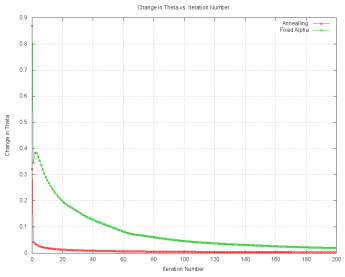
The results of folds 1 and 4 inspired us to run a slightly modified experiment. For each matchup we assign a probability to the event that a given team wins using our sigmoid function, typically in the range [0.5, 0.85]. Originally, we classified that team as going to win if the probability was greater that 0.5. For our modified version we recognized that if we believe Team 1 is going to win 51% of the time we also believe it is going to lose 49% of the time. Thus, we decided to simulate 1000 brackets using these probabilities, and see how well the best of these brackets performed. I.e. we flipped a coin weighted according to our generated probabilities at each matchup and then advanced the appropriate team. This result proved to be quite impressive. In the 2014 run, our 1000 bracket simulation produced a bracket that would have performed better than every single human bracket last year except for 3 out of over 11 million generated.

For SVM, we used grid search to tune the parameters C and γ in our algorithm. See Figure 1 for a graph of these parameters.

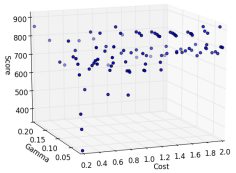
We were also very pleased with our results for SVM. It did not perform as well as logistic regression in terms of average score, an artifact most likely due to an inadequate amount of data. However, SVM still correctly predicted the eventual tournament winners for folds 2 and 3. This suggests that when the tournament results are more predictable, our current features are sufficient for learning a reasonably good model. We anticipate that if we were able to acquire more years of data our SVM would continue to improve. We posit that using the infinite dimensional Gaussian kernel on this small of a dataset is why our SVM is underperforming logistic regression.

Table 1: *Errors and Scores for Different Algorithms and Feature Vectors*

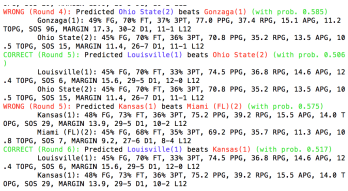
Model/Algorithm	Avg. Training Error	Avg. Test Error	Avg. Score
Baseline (choose highest seed)	0.32	0.32	652
Logistic Regression	0.30	0.30	1017
SVM ($C = 2$, $\gamma = 0.125$)	0.32	0.33	922
Best of 1000 Simulation	0.30	0.22	1680
Oracle, i.e. best human bracket	n/a	n/a	1650



(a) *Convergence of theta using annealing versus fixed alpha*



(b) *Average score using various cost and gamma values for SVM*



(c) *Sample verbose output*

Figure 1: *Graphs for Annealing and Grid Search and Error Analysis Output*

VI. Analysis

Once we had extracted our features and coded up working algorithms and a scoring function, we began work in analyzing our results to see which examples we were misclassifying and why.

To discover what kinds of features were relevant we compared the success of various subsets of features using Forward and Backward Search. We would add or subtract a feature, run our algorithm to see if our results improved, if so take action, and repeat. We discovered that of the game statistics nearly all of them were useless except for FG%, FT%, and 3PT%, which were marginally helpful. Thus, we discarded statistics like assists, rebounds, steals, etc. This approach of Forward and Backward Search only took us so far because while we had an intuition for what kinds of features might be useful, our intuitions were often wrong and a feature that might be helpful in one subset turned out to not necessarily be helpful in another.

Our next approach led us to look specifically at which matchups we were getting wrong and just how wrong we were (i.e. how high of probability did we think a certain team had of beating another that actually did not). To do this, we added a verbose option in our output. See Figure 1.

The verbose output allowed us to pinpoint exactly which matchups we were getting wrong, which round that matchup occurred in, and what probability we thought the team we chose had of winning. Because of the nature of our scoring function, the first matchups that we looked at were the ones in the later rounds. Reversing whether we correctly classified the winner of the tournament had a huge effect on our score. We then looked at matchups which we were very unsure of i.e. those who we were predicting a certain team to win with probability 0.51 for example. After identifying these such matchups, we then looked at the features corresponding to each team. Our thought process was "Team A has a good SOS,

low TOs, high FG%, etc" and "Team B has a bad SOS, bad scoring margin, but really good win % in their last 12 games." If Team B wins the matchup, then maybe we should be emphasizing win % in last 12 games more. We went along like that iterating our feature extractor until we were performing as well as we could without overfitting.

VII. Discussion

Our best results with logistic regression averaged a score of 1017, which would be in the 99th percentile of any year. The oracle for our problem, i.e. the best performing human bracket each year, averaged a score of 1600. Further, the score for a perfect bracket is 1920. Thus, while our average score of 1017 is impressive, we are still incorrectly labeling a large number of matchups, and we are only about halfway to the score of a perfect bracket. The two major factors for our inability to make further progress are the availability of data and the inherent difficulty of predicting a perfect bracket.

As we mentioned earlier, ESPN only has detailed data available for the last four years. This is not much data at all and limits the amount of learning we can achieve with our classifiers. We were not able to truly utilize the power of SVM since we would end up simply overfitting the small set of training examples. Further, since college basketball is an amateur sport, quality data is generally less available than the data for professional sports.

Beyond the data, this problem is quite simply very hard. Basketball is an extremely high variance sport, and even though you may be able to determine which team is generally better, there are absolutely no guarantees about what will happen on one given night. This is especially impactful because March Madness uses single game elimination. Further, the progression of matchups in a bracket makes errors extremely harmful. For example, incorrectly predicting that the overall champion will lose in the first round means that you will lose out on the points for every matchup that the champion is involved in.

In the end, while we didn't get close to our goal of creating the perfect bracket, we have to be happy with our results as there are simply too many challenges to this problem. It would be interesting to see how these algorithms would perform a decade down the road when there are more tournaments and data available on ESPN.

VIII. Future Work

While we have made reasonable progress, there is still much future work to do. At this juncture, we feel that time would best be spent collecting data for more tournament years and more features. From there, it would be very interesting to see what kind of additional features may prove valuable for this problem. Perhaps qualitative features about the players on a specific team or inter-temporal features regarding a coach and a specific program may be useful. A next step may be to try improving the algorithm perhaps via a neural network. At the end of the day, this specific problem is so complex and suffers from such high variance, it seems unlikely that any amount of data or learning could ever generate a perfect bracket.