Capstone Report

March 20, 2019

1 Machine Learning Engineer Nanodegree

1.1 Capstone Report

Maniceet Sahay March 20th, 2019

1.1.1 Domain Background

NCAA Basketball tournament also known as March Madness brings with it the challenge of predicting the bracket and although it is unlikely to correctly predict all the matches correctly, maybe with data and machine learning we can assign some probabilities to these matches. The motivation behind picking up this as a project is the Kaggle competition that takes place every year around NCAA, the capstone also serves as an incentive to deep dive in the competition itself.

1.1.2 Problem Statement

Every year 64 teams take part in the annual NCAA Basketball competition and everyone gives in their prediction on who will go on to win the competition and due to upsets no one can predict the perfect bracket, but leveraging machine learning and historical data we can try to get as close as possible in predicting who will win.

The goal is to engineer features such as 3-pointer conversion rates and see whether they can help us in determining which team will win and give an objective view to the problem rather than going by expertise itself and have a quantitative approach to predicting the winners rather than a qualitative which most of us have.

1.1.3 Datasets and Inputs

The dataset can be obtained here

The data contains all sorts of data with the seed of teams from 1985 to 2018 and compact results where every result from 1985 to 2018 and detailed resulst which highlight results from 2003 to 2018 including all sorts of statistics such as number of 2 pointers attempted which will help us in engineering new features.

Apart from the dataset provided here, data from Kenpom were used to supplement the data already available. R code was used to scrap the data from the website.

1.1.4 Evaluation Metrics

The evaluation metric will be log loss

$Logloss = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$

where

- n is the number of games played
- $\hat{y}_i$ is the predicted probability of team 1 beating team 2
- y_i is 1 if team 1 wins, 0 if team 2 wins
- $\log()$ is the natural (base e) logarithm

A smaller log loss is better. The use of the logarithm provides extreme punishments for being both confident and wrong. In the worst possible case, a prediction that something is true when it is actually false will add an infinite amount to your error score. In order to prevent this, predictions are bounded away from the extremes by a small value.

1.1.5 Analysis

Files used We used the following files for our analysis

1. Seeds
2. NCAA Tourney Detailed Results
3. Regular Season Detailed Results
4. Teams Spellings along with ID
5. Kempom Data (Scrapped using R)

Data Wrangling The original format in which the data was present had Winning Team ID and losing team ID, to make that our data had a target variable, we duplicated the rows and gave the Winning TeamID T1 prefix in half the rows and T2 prefix in the other half, thus giving us a dataframe which had 50-50 distribution of 1s and 0s.

Then we concatenated both the regular seasons and ncaa seasons data into one dataframe.

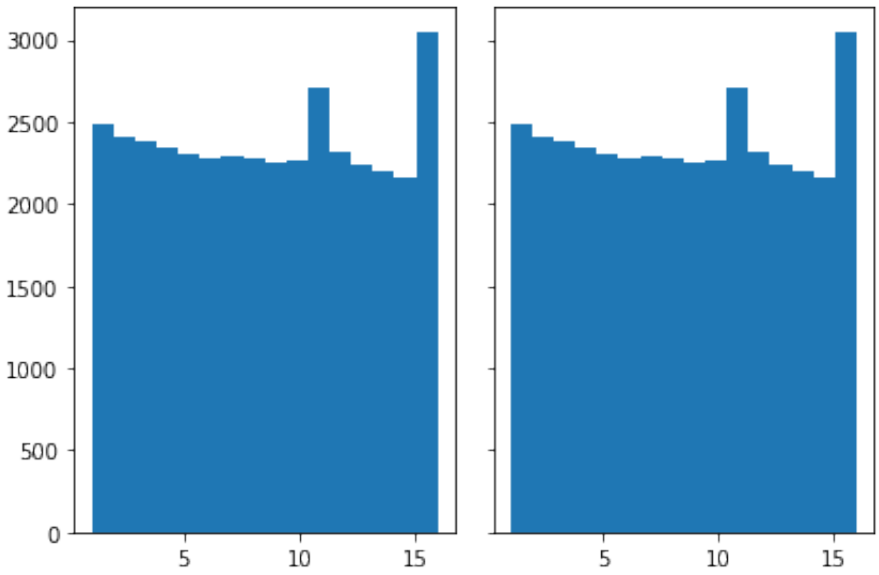
Then seeds were added to the data file, simple left join was made first on T1_TeamID based on combination of Season and T1_TeamID, then similarly on T2_TeamID.

Since Kenpom Data has Team Names, we first bring in TeamID into Kenpom Data by using Team Spellings and removing the one row for which TeamID could not be matched. Subsequently we made a left join on Season and TeamID with Kenpom Data to bring in various metrics

After bringing in the Seeds for the respective TeamIDs and metrics from Kenpom there was problem that we ended up with 139154 missing values in seeds and some missing values in Kenpom metrics.

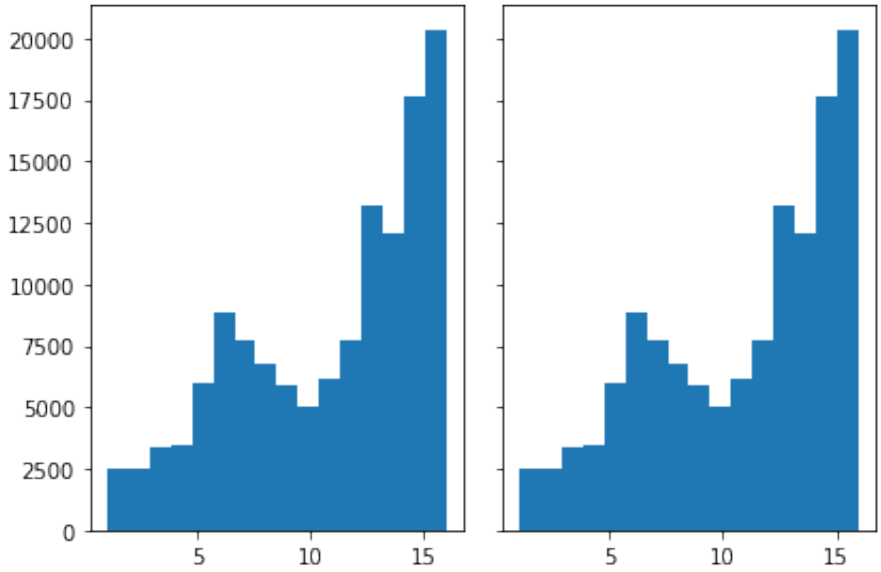
For Kenpom metrics missing values, we replace them with the mean value for the Team ID for which they were missing.

To handle missing seeds, we first saw how they were distributed



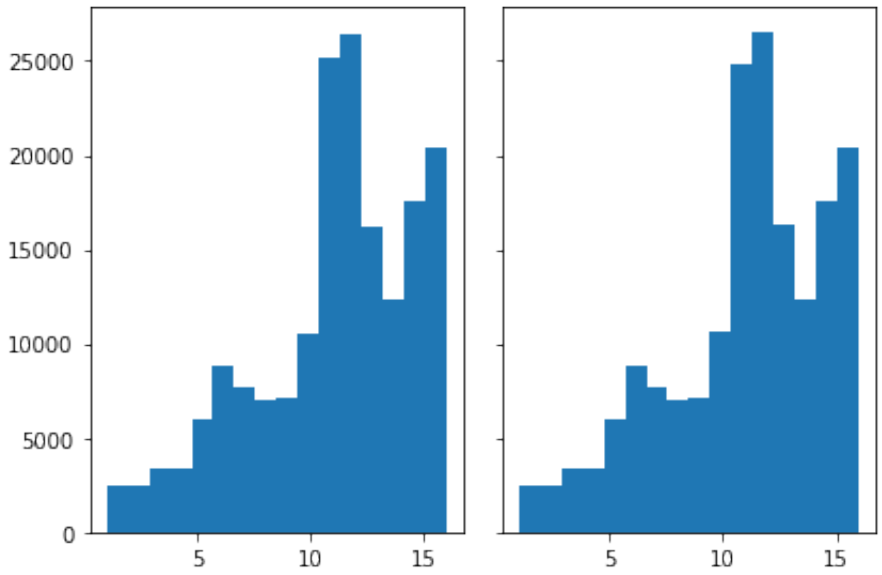
So let us see the distribution of Seeds

Based on the current distribution we can be replace the missing Seeds with mean value of Seeds for the given Team ID. After making the imputations we were still left with 47973 missing values. Lets take a look at the seed distribution



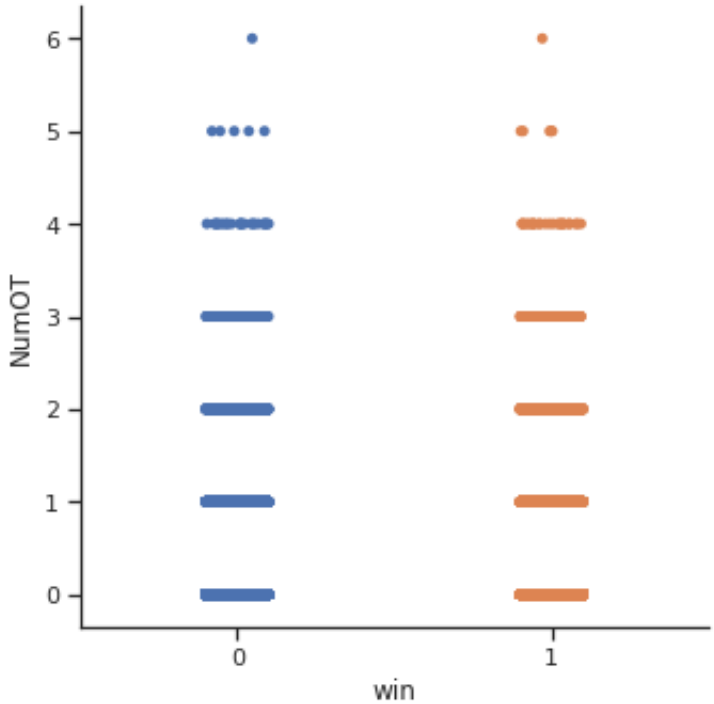
Since the distribution is slighly skewed, we have two options, either use the median of the seeds or use other features and use them to predict the Seed for the data points we don't have seed for. In this project we chose the latter approach using Random Forest to predict the Seeds.

Let us take a look at the distribution after Imputation using Random Forest

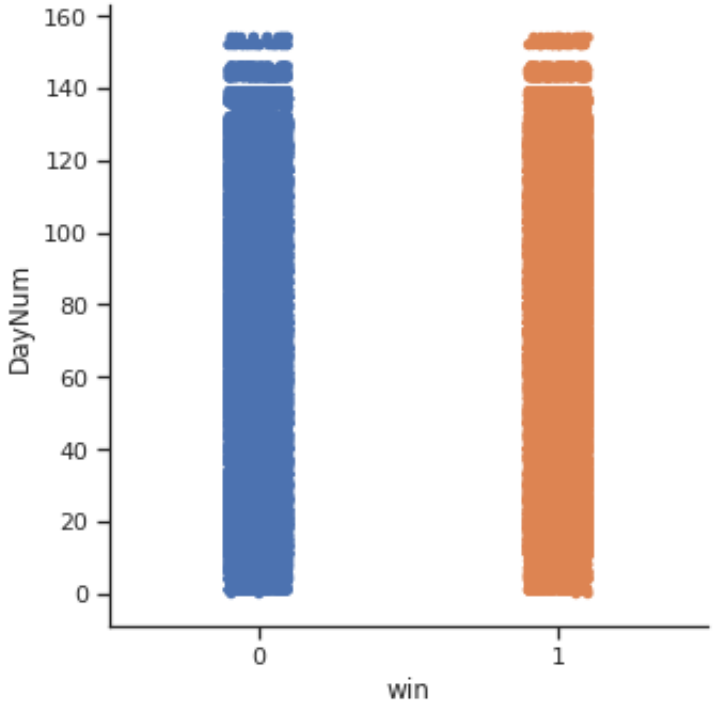


The distribution looks a little skewed, but majority of teams are in the middle which is expected.

Data Exploration Let us see some features such as NumOT, DayNum and the likes and see whether they can give us some info about the outcome of the match **Nu-**



mOT Plot **DayNum Plot**



Based on these plots we can clearly see that there is no clear connection between the outcome and how much Ovetime the match went into or the Day number in which it was played. Thus we will not be using these as features

Rest of the feature elimination will take place based on the output of the model

1.1.6 Benchmark Model

For the benchmark model, the dataset was first divided into train and validation set, with the train set containing data from seasons 2017 and earlier and validation set containing data from seasons 2018 and 2019.

Only the seeds and seed difference were used as features.

Based on this and with a logistic regression model we got a logloss of 0.6637 and auc of 0.6304, which are quite low and suggests that using these features only we are not much better of than random guessing the winners of the match.

Let us see our performance on the data set with all the features. **Note:** Location is not used as a feature as NCAA matches always take place on a neutral venue

Performance on seeds plus features We engineered multiple features based on NBA stuffer team metrics such as Assist_Ratio, Turnover_Ratio, Possession, Offensive_Efficiency, Defensive_Efficiency,Point_Differential, Efficiency_Differential, Defensive_Rebound_Perc, Offensive_Rebound_Perc,Eff_Field_Goal_Perc, Turnover_Rate, Free_Throw_Rate, Four_Factor. However, when the entire dataset was used, we got really good results such as logloss in the range of 0.03-0.05 and auc of around 0.99 which sent of alarm signals. This meant that we are sending information in the model which should not be available when testing. As we know from kaggle forums that logloss of around 0.49 is quite respectable, so lets take a look at the feature importance and think which features should we keep what we should not keep.

Weight	Feature
0.1210 ± 0.0041	T1_Score
0.1207 ± 0.0039	T2_Score
0.1003 ± 0.0036	T1_FGM
0.0995 ± 0.0031	T2_FGM
0.0524 ± 0.0025	T2_FTM
0.0520 ± 0.0017	T1_FTM
0.0355 ± 0.0014	T2_FGM3
0.0346 ± 0.0021	T1_FGM3
0.0149 ± 0.0006	T1_DR
0.0149 ± 0.0016	T2_DR
0.0011 ± 0.0007	T2_TO
0.0011 ± 0.0004	T1_OR
0.0010 ± 0.0008	T2_Possession
0.0009 ± 0.0011	T1_TO
0.0008 ± 0.0003	T1_Possession
0.0008 ± 0.0006	T2_FTA
0.0007 ± 0.0004	T2_OR
0.0006 ± 0.0006	T1_Turnover_Ratio
0.0006 ± 0.0003	T2_FGA
0.0006 ± 0.0007	T1_Turnover_Rate
… 49 more …	

Clearly features like Score and how many goals were made are influential. In the future we donot have knowledge of these features before hand, we do not know how many goals were made, how many points were scored, so we will not be keeping any features like those and only use Kenpom features along with seed and seed difference.

Looking back, we can see that NBA stuffer metrics are all backward looking and thus cannot be used in our algorithm when we are predicting the future, NCAA 2019 outcomes, thus they also won't be used.

When running a logistic regression on seeds plus metrics from kenpom Dataset, we get an average logloss of around 0.4805 across seasons 2016-2019, with each iteration being that one season is kept as validation and the model is trained on the rest. We also achieve auc's of around 0.84 in all of the seasons which is a considerable improvement over our current model.

If we use random forest as our model of choice, using 500 trees and a max_depth of 5, we get an average logloss of 0.512 which is again an improvement over our benchmark model.

If we average out the results of both logistic regression and random forest we get an average logloss of . Clearly here going with only logistic regression is the right choice.

Here is the feature importance generated by the logistic regression model.

Weight	Feature
0.0283 ± 0.0016	T2AdjustedEM
0.0259 ± 0.0047	T1AdjustedEM
0.0184 ± 0.0061	T1AdOfE
0.0180 ± 0.0037	T2AdOfE
0.0141 ± 0.0034	T1AdDeE
0.0136 ± 0.0042	T2AdDeE
0.0076 ± 0.0031	T1Luck
0.0072 ± 0.0031	T2Luck
0.0025 ± 0.0032	T2Rank
0.0020 ± 0.0016	T1Rank
0.0020 ± 0.0025	T1Conf
0.0015 ± 0.0011	T2AdDeER
0.0015 ± 0.0038	T2Conf
0.0008 ± 0.0001	T1AdSoSR
0.0007 ± 0.0004	T2AdSoS
0.0006 ± 0.0003	T2AdTempo
0.0006 ± 0.0008	T1AdOfER
0.0005 ± 0.0006	T1AdTempoR
0.0005 ± 0.0015	T2AdOfER
0.0005 ± 0.0003	T1AdTempo
… 9 more …	

The features are looking less dominant and the metric is also in line with expected results. I think we can start bracketing using these outputs

1.1.7 Conclusion

With adding features from external dataset, we saw that machine learning helped us in going from random guess to a very educated guess on which team is going to win the match. We also saw what happens when one incorporates features which can be biased such as Score which is often not available beforehand and can make the performance look too good to be true which it often it. There can be much more done such as using massey ordinals data and trying to engineer features such as elo rankings that may be beneficial to the overall outcome.

However with March Madness almost upon us, I guess we can use the outcomes of this model as a rough starting point to start filling out brackets :)

1.1.8 Acknowledgements

This capstone would not have been possible without the extremely helpful kaggle community for their kernels and discussion forums. Kempom data has been instrumental in driving the outputs without having to incorporate metrics which has hindsight bias. Finally thanks to Udacity for the learning experience during the ML NanoDegree program