Google Cloud & NCAA® ML Competition 2019-Men's

Koby Close | linkedin.com/in/koby-close/ | github.com/koby-a-close

09/05/2019

Personal Background:

I am a Biomedical Engineer seeking an opportunity as a Data/Business Analyst. This was my first Kaggle competition as I look to showcase my skills analyzing data. My professional experience has been based around collecting high quality data for product development projects, using that data for analysis of project goals, and presenting updates and findings. In my education and career I have used mostly Matlab and Excel. Since starting this career transition in July 2019 I have taken up Python, SQL, and Qlik Sense for data visualization.

Project Background and Goals:

This project is hosted by Google and the NCAA. The data provided by Google includes regular season and tournament data from 1984 on. The goal is to predict the outcomes of the 2019 Men's NCAA Tournament using historical data to build a model. Since we don't know who will win each game, every possible combination is predicted by the model and the relevant matchups are pulled during scoring. Submissions are scored using log loss. This method of scoring penalizes you heavily for being both confident and wrong.

Personal Goals:

Since this project was my first I wanted to set some goals for myself:

1) Create a model that outperformed the Google Starter Kernel. I set this goal after finishing the Google Starter Kernel (discussed more below) which had a log loss score of 0.51455 and would have placed 508th.

2) Create a model that outperformed the picks I used for bracket competition. My family does a bracket competition where winners are chosen round by round so this is a fair comparison. My final record in 2019 was 42-24 or 64% accuracy.

Skills Used/Added from this Project:

- Python Packages: pandas, numpy, sklearn, xgboost
- Model Creation: logistic regression, decision trees, gradient boosting, feature selection, cross validation, performance evaluation
- Data wrangling

Summary of Results:

Model	Log Loss Score	Projected Ranking	Accuracy
Google Starter Kernel	0.51455	508	65% (43-23)
GSK with Guessing Strategy	0.48970	304	71% (47-19)
Logistic Regression using Win %	0.61856	695	55% (36-30)
Decision Tree	0.52551	556	65% (43-23)
XGBoost Ensemble	0.50034	420	67% (44-22)

Table 1. Summary of results for each model created. Log Loss score is how the model was evaluated by Google, Projected Ranking is relative to the closed leaderboard on Kaggle, and Accuracy is the percentage of correct picks and its record.

Model #1 - Google Starter Kernel:

Strategy - I started by going through the Google Starter Kernel to understand the data, some general methods, how the submission file should be formatted, and how the model would stack up on the leaderboard.

Execution - The model used logistic regression with the feature being 'Difference in Seed'. I made no changes to the code initially and submitted it.

Result - The score was 0.51455 and would have placed 508th. Its accuracy was 65% (43-23) already beating my manual picks.

Conclusions - It already beat my predictions so that goal was met.

Model #2 - Google Starter Kernel with Guessing Adjustment:

Strategy - After submitting the starter kernel I went through the Discussion board to look for ideas on which models I should try next. The first post I found was from the 3rd Place finisher - Madtown Machine Learning Madness. They used logistic regression with random guessing on predictions with low confidence and their ability to submit two entries to hedge their random guesses. Here is their explanation for the random guessing:

"When the game is unpredictable (model prediction is less than 77%), we picked the team with lower ID to win with 64% probability. In our second submission, we predicted the other team to win with 64% probability. Assuming that team IDs have nothing to do with winning, this is just random guessing. Under log-loss scoring, this strategy works if there is an uneven split between the number of correct and incorrect predictions. The more extreme the split, the better this works. For example, 70:30 split is better than 60:40. If these games are truly unpredictable, there's about 30% probability that the split will be favorable for our strategy (binomial distribution)."

Execution - I decided to implement this strategy on the starter kernel and see if it improved that model as well. Since they used logistic regression and it worked for them it was very likely it would. To do this I added a couple of mask commands at the end of the code and wrote two new csv files for entry.

Result - The two log loss scores were 0.54449 and 0.48970 and would have placed 599th and 304th, respectively. The model that performed better than the starter kernel had an accuracy of 71% (47-19) - also a significant improvement that would have won my family competition.

Conclusions - The guessing strategy worked as expected and accomplished my first goal. However, it didn't really make a good model, it simply used the submission limits to make a statistically sound guess. I wanted to create a model that beat the kernel by itself. The record also improved significantly.

Model #3 - Logistic Regression with Winning Percentage:
Strategy - I know that the seeding committee for the NCAA is careful in its seeding but maybe I could pick a different metric to use for logistic regression that would perform better than seed difference. I decided to try Difference in Winning Percentage from the regular season.

Execution - Getting the Winning Percentage (Win %) for each team was most of the work for this model as the regression code was already built out. To get Win % I used groupby and pandas.merge to get each team's wins and losses. Win % is the percentage of games won relative to the total games. I also utilized fillna to give teams with zero losses a value. Once the Difference in Win % was calculated for each match up I built the model and wrote a csv file for entry.

Result - The log loss score was 0.61856 and would have placed 695th. It's accuracy was 55% (36-30).

Conclusions - This model did not perform well at all. I tried adding the guessing strategy to the model and the log loss scores were worse as well. I decided maybe I wasn't good at choosing features. My next model would be a decision tree so I could feed it features I suspected would be useful and allow the model to pick for me.

Model #4 - Decision Tree:
Strategy - For the decision tree I wanted to give the model a handful of different features and see if that would improve log loss performance. The features I extracted from the data for each team were Wins, Losses, Win %, Points per Game (PPG), Points Against per Game (PAPG), Wins in Close Games, Losses in Close Games, Win % in Close Games. I defined a 'Close Game' as one where the final score was less than 10 points.

Execution - Getting the features was very similar to how I got Win % for Model #3, just repeated for different values. From there I used sklearn to build a decision tree and cross validated the model using log loss as my scoring method. The initial model didn't perform particularly well on cross validation and I knew decision trees could easily overfit so the first thing I did was find a way to visualize the decision tree. I accomplished this using pydotplus and graphviz. The tree was way too complex and needed to be

optimized. To optimize the tree I wrote a function to build the tree and ran it through a loop varying max depth, minimum sample splits, and minimum sample leaves. I used the parameters that had the lowest log loss score during cross validation. To ensure repeatability I used a seed value of 7 for all shuffling of data. The parameters I used for max depth, minimum sample splits, and minimum sample leaves were 5.0, 0.1, and 0.08, respectively, and got a mean log loss score of 0.56425 during cross validation. This model was used with predict_proba to make the csv file for entry.

Results - The log loss score for the model was 0.52551 and would have placed 556th. It's accuracy was 65% (43-23).

Conclusions - The decision tree was unable to outperform Google's logistic regression model even when I did my best to optimize parameters. I would like to know more about how to evaluate if a model is overfitting or underfitting but I suspect this was a case of overfitting. Additionally, the features I provided weren't particularly telling so adding more features like offensive and defensive efficiency would increase this model's performance. While doing research on decision trees I found a lot of discussion about xgboost and decided my last attempt would be to use xgboost on this set of features.

Model #5 - Ensemble using XGBoost

Strategy - For this final model my goal was simply to try out xgboost and see if it could use the features I had already created to make a good model. I also wanted to add a performance evaluation to this code to simply see how accurate the model was on training data. Finally, I wanted to use a test/training dataset as well as cross validation to test model performance.

Execution - Implementing the model was very similar to a decision tree so the code modification was based around creating a test and training dataset to evaluate accuracy followed by a cross validation evaluation of model performance using log loss for scoring. To ensure repeatability I used a seed value of 7 for all shuffling of data and splitting into training and test sets. The model created without changing any parameters already looked better than my decision tree. The accuracy was 69.49% with a mean log loss score of 0.55668 during cross validation so I stopped there and made a csv of predictions. The parameters used in the base model for learning rate, number of estimators, max depth, and subsample were 0.1, 100, 3, and 1, respectively.

Results - The log loss score for the model was 0.50034 it would have placed 420th. It's accuracy was 67% (44-22).

Conclusions - This ensemble model achieved both of my project goals without using any guessing. The log loss score was better than the Google Starter Kernel and my record in my family competition would have been good enough for 3rd place. Similar to the decision tree model I could work to optimize parameters for this ensemble and give it more useful features to improve its performance. I plan to tweak this model until next spring and enter it into the 2020 NCAA competition to see how it does.

Conclusion:

I found this first Kaggle competition to be very insightful and a great way for me to learn some data modeling techniques. I heavily utilized StackOverflow and help files to get my code working and know this will be a valuable resource moving forward. The skill I improved the most on from this competition is wrangling the data into the form I want for model training. I am now well versed in groupby and joining commands in python using the pandas package. Additionally, I learned a lot about optimizing parameters and evaluating model performance while coding the decision tree and ensemble models.

Moving forward I would like to apply these skills to a less controlled data set to test my data wrangling more. I would also like to continue trying different models, tuning techniques, and evaluation techniques so I can be confident building models for all types of data.

My next project will be using Google' public BigQuery dataset on MLB pitchfx data from the 2016 season. I will get more practice within python as well as with SQL and visualizing data using Qlik Sense.