# CS 209A: Final Project[1]

Predicting the 2022 FIFA World Cup

*Omar Abdel Haq, Elie Eshoa, Ibrahim Suat Evren, Mari Kikuta, May Soshi*

December 11, 2022

[1]See the following link for the full submission portfolio:
https://drive.google.com/drive/folders/1l6CiTvyarUActSI2UJo1SXLdI_D_-MzY?usp=share_link

## Introduction

The World Cup is the most viewed sporting event in the world, with billions of people watching or streaming the games every day. This year, Qatar is spending $229 billion to host the World Cup. According to Forbes Magazine, a total of 3 million World Cup tickets have been sold. In 2026, the World Cup is predicted to be even bigger with Canada, Mexico, and USA as host countries. To say the least, the World Cup is a big deal. With this year being the 22nd quadrennial Men's FIFA World Cup, we thought it would only be fitting to try to predict the winners of the World Cup based on player, team, and historical data. The World Cup thus far has been rather unpredictable with many fan-popular teams losing and many underperforming teams advancing to the next round. Amongst this uncertainty, can our model predict the World Cup with certainty? Read ahead to find out.

## Base Implementation – Naive Model

The first step of our project was to establish a naive model to predict the World Cup winners. Our naive model predicts the winner and the three runner-ups of the World Cup by simulating the tournament and selecting the winner of each match based on each team's probability of winning. Each team's probability of winning is calculated by dividing the number of wins by the number of matches that team has participated in. If the win probabilities for two countries are equal then we select the winner randomly. After running our naive model, we found that Brazil would be the predicted World Cup winner, with Portugal, Argentina, and Denmark as the 2nd, 3rd, and 4th place runners-up [Figure A].

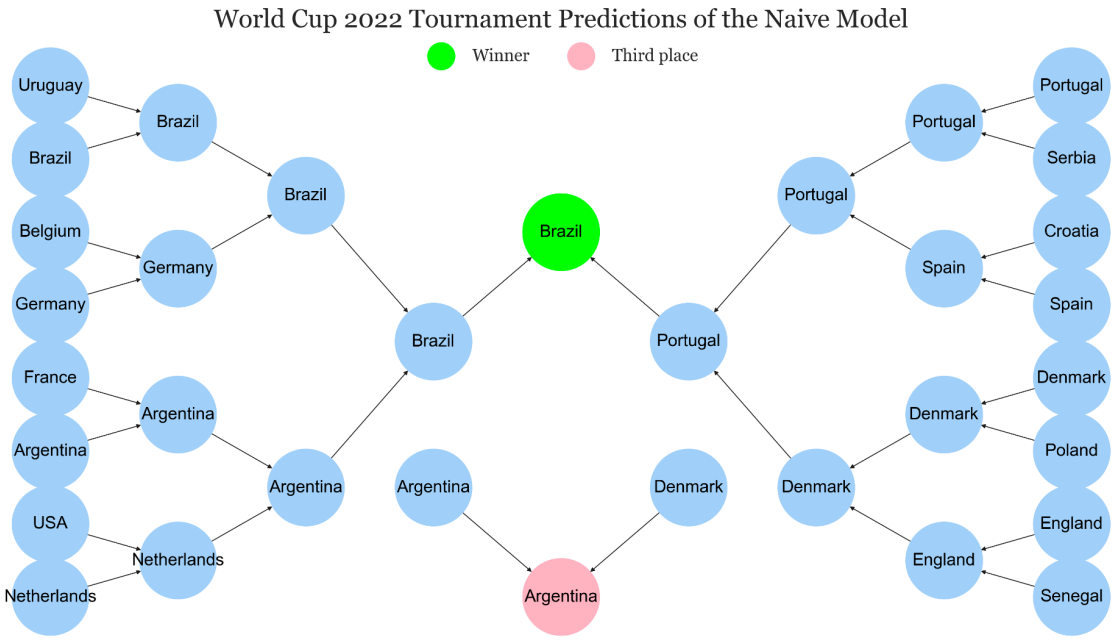

**Figure A. The naive model's predictions. The edges point to the winner of the match.**

## Description of the Data

Since our model would go on to use the team-level statistics going head to head in a match, the data we needed had to include player information for each team, player-specific statistics as measured by the FIFA video game, and the matches played (and their results). Due to limited information available online on player statistics prior to the 2010 World Cup, our model was trained only on the matches and results of the 2010, 2014, and 2018 World Cups. For each of the four World Cups we had interest in (the three we used to train our models and the 2022 tournament we are interested in predicting), we scraped the official list of teams participating, and the names of players in each team, with each team consisting of around 26 players. Having scraped and organized the datasets for each tournament's teams and their players' names, we needed to find player-level statistics for each team.

We used player-level statistics data from the FIFA video game, available on Kaggle, to find the player level statistics for the players on each team. Namely, for each player on each team, we looked for the most similar player's name in the FIFA dataset (if one existed); we used the difflib library in Python to find the most similar player names. Then, we augmented the player's row to include relevant information about the player's abilities: overall rating, dribbling ability, freekick accuracy, agility, ball control, interceptions, and penalties. This step, after repeating for all players, introduced some missingness into our data: some players were either not in the FIFA dataset, or their names could be spelled in very different ways we couldn't account for. We ended up restricting our player analysis on the overall rankings, as we assumed they were based on a summation of all the other statistics anyway.

To deal with missingness, we decided to use mean imputation for each team. For each team, we calculated its mean player statistics (ignoring players with missing information), and then substituted in these mean values as the assumed statistics for the other players in that team. By the end, we had created a dataset which, for each tournament, included the teams participating, all players in each team, as well as their abilities on the field.

## Design Matrix

After scraping our data, we created our design matrix by taking the top goalkeeper, top 3 forwards players, top 4 mid-field players, and top 3 defense players of each team and averaging their individual scores to create a goalkeeper (GK), forward (FW), mid-field (MF), and defense (DF) score. Our reasoning behind this was to create a score for each position on the team so that we can see how much each group of players contributes to the success / win-rate of the team. We also created a "backup" score which includes all

the remaining players that oftentime substitute in for the main 11 players on the field. We then used the team score dataset and merged it with the matches dataset to create our final design matrix. For simplicity, we disregarded penalties when deciding the outcome of a match. A team was declared as a winner of a match only if they had the higher score in the match, otherwise we considered it as a tie. For our response variable, we created a 'winner' indicator variable which indicates a 1 if the two teams tie, 2 if the home team wins, and 0 if the away team wins. For each match (row), we added an additional match to the matrix that reversed the order of the home and away teams to account for the fact that being home or away does not affect the outcome of the match. We wanted to ensure there was no bias toward the home teams when it came to classification and prediction. For instance, for the South Africa-Mexico match, we added a second row called Mexico-South Africa which includes the same data but treats Mexico as the home team as opposed to the away team. Since the home and away teams are switched in the second row, each of the variable columns are also switched in the dataset. Doing this balances out the bias toward the home team since we use 2 to indicate the home team as a winner and 0 to indicate the away team as a winner. This final design matrix was used to train our models to predict the outcome of a match. After our models were trained, we used the initial group matches from the 2022 World Cup to predict a winner.

| Matches | GK Home | GK Away | ... | MF Home | MF Away | Year | Winner |
|---|---|---|---|---|---|---|---|
| *South Africa - Mexico* | 72.0 | 71.0 | | 69.67 | 68.38 | 2010 | 1 |
| *Mexico - South Africa* | 71.0 | 72.0 | | 68.38 | 69.67 | 2010 | 1 |
| ⋮ | | | ⋮ | | | | ⋮ |
| *England - Belgium* | 83.0 | 89.0 | | 82.25 | 85.5 | 2018 | 0 |

**Table 1: Design Matrix**

### Models

To train our data, we created five different models: simple logistic regression, a threshold model, decision tree, decision tree with bagging, and random forest. For each of these models, we created a train and test set of size 0.75 and 0.25, respectively.

For the **logistic regression model**, the train and test accuracy were 0.551 and 0.529, respectively. As we can see, both the train and test sets are performing poorly in this

model. A potential reason for this performance could be due to the fact that we are dealing with an ordinal response variable – that is, winning is better than drawing which is better than losing – and as a result, the logistic model was unable to learn from the train set well and then correspondingly generalize to the test set.

To combat this, we implemented a model from scratch inherited in *sklearn* by using the object-oriented nature of Python, which we call the ***threshold model***. The threshold model takes into account the ordinal nature of the response variable. For that, we first implemented **ordinal logistic regression** [1] to decide on the probability of each outcome given a match. Our ordinal logistic regression model creates a dummy variable (y_12) for all matches that were either a home team loss or a tie and another dummy variable (y_01) for all matches that were either a tie or a home team win. By grouping the two indicators together, we were able to reduce our ordinal response variable to a binary response variable. Just for demonstration's sake, here is what the transformation from the 'Winner' variable to 'y_01' and 'y_12' looked like:

| Winner | y_12 | y_01 |
|---|---|---|
| 2 | 1 | 1 |
| 1 | 0 | 1 |
| 2 | 1 | 1 |
| ⋮ | ⋮ | ⋮ |
| 0 | 0 | 0 |
| 1 | 0 | 1 |

**Table 2: Transformation from 'Winner' response variable to dummy variables y_01 and y_12**

We can see in this table that the 'y_12' variable treats losses and ties as one indicator and win as another indicator whereas the 'y_01' variable treats wins and ties as one indicator and losses as another indicator.

The main assumption of the ordinal logistic regression is the *proportional odds assumption* [2]. In the context of our problem, this corresponds to every covariate having the same effect on both of the variables y_12 and y_01. We see that this assumption indeed holds due to the symmetric nature of our decision problem.

We converted the probabilities obtained via ordinal logistic regression into deterministic predictions by using thresholds. Our model took the parameter *threshold* as an input. In particular, given a match, it predicted that

- Away wins if $Pr[y_{01} = 0] - Pr[y_{12} = 1] > threshold$,
- Home wins if $Pr[y_{12} = 1] - Pr[y_{01} = 0] > threshold$,
- Tie otherwise.

We decided by cross validation in the train set that the best threshold is 0.031. Although this model resulted in a slight improvement in the train set, it gave the same exact accuracy in the test set as the simple logistic regression. We hypothise that this is because of the fact that our data set is very small.

In addition to this, we created a decision tree to classify our data. Through cross-validation, we found that the best depth for our decision tree was 18. The train and test accuracy for the decision tree at depth 18 was 1.0 and 0.5577, respectively. This is a slight improvement from the previous two models, both of which had a test accuracy of 0.529.

To further improve the decision tree model, we used bagging to aggregate our predictions and reduce overfitting. The bagging decision tree model had a training accuracy of 0.862 and a test accuracy of 0.577. This is an improvement from the previous decision tree implementation since it overfits less and generalizes better to the test set.

The final model we created was a random forest. The random forest model had a train accuracy of 0.994 and a test accuracy of 0.596. Evidently, from all five models, random forest performs the best.

| Model | Train Score | Test Score |
|---|---|---|
| Logistic Regression | 0.551 | 0.529 |
| Threshold (*0.031*) | 0.554 | 0.529 |
| Decision Tree (*d = 18*) | 1.000 | 0.558 |
| Bagging Decision Tree (*d = 18*) | 0.862 | 0.577 |
| Random Forest (*d = 8*) | 0.994 | 0.596 |
| XGBoost with SMOTE | 1.000 | 0.567 |

**Table 3: Summary of models train and test accuracies**

**209 Components**

1. ***Implementing the Threshold Model from scratch***

This component was already explained in the models section.

2. ***XGBoost and Synthetic Data Generation with SMOTE***

Since our dataset is small, our main concern would be to try and create more data points for our model to learn on, or to somehow counteract the scarcity of the data using creative methods. One such method would be the Synthetic Minority Over-sampling Technique (SMOTE), which is a technique that generates synthetic data. The idea of SMOTE is to generate new data for the minority class by randomly selecting a point from the minority class and computing the k-nearest neighbors for this point. The synthetic points are then added between any two nearest data points joined by a straight line. In order to do this, the algorithm calculates the distance between two data points in the feature space, multiplies the distance by a random number between 0 and 1 and places the new data point at this new distance from one of the data points used for distance calculation.

Although SMOTE is usually used for minority classes, we still hugely benefit from it to generate synthetic data, since our dataset is small. Now that we have an internal method that can somehow mimic the existence of a large dataset, we can use XGBoost, which is known to work well on relatively bigger datasets and is a very famous method for gradient boosting. Indeed, when a SMOTE transformation was applied to extend our small dataset, our XGBoost model performed better on the synthetically generated data. The jump in the testing accuracy can be seen in our notebook.

Another method that we considered, but have not analyzed due to its randomness, was adding noise to the data to get a bigger dataset. It was interesting to see how the model performs with a bigger dataset, which is the concatenation of the old dataset with the new dataset that includes some noise. This method will not be used, but is worth mentioning as another way to increase the dataset size.

**Feature Importance**

From the previous section, we determined that the random forest with depth-8 and 250 trees achieved the highest accuracy score with 0.994 for the training set and 0.596 for the validation set. We also found that the bagging decision tree with depth-18 had the second best accuracy with 0.86 for the training set and 0.577 for the validation set. The simple decision tree from which the bagging model was created had the third highest accuracy score with 1.0 for the training set and 0.558 for the test set. Performing feature importance for both the decision tree and the random forest mode using mean decrease in impurity, we see that the ‘backup Away’ (substitute players in the away team) and ‘MF Home’ (midfielders on the home team) variables had the most impact on the decision tree classification whereas ‘FW Away’ (forwarders on the away team) and ‘FW Home’ (forwarders on the home team) had the most impact on the random forest classification.

The decision tree model indicates that the backup players in the away team and the mid-field players in the home team played a significant role in swaying the outcome of the matches. In contrast, the random forest model indicates that the forward players in both the home and away teams are affecting the outcome of the match. The permutation importance score [Figure C] also shows that forward players are contributing to the outcome of the match for random forest whereas midfield players for the home team and forward players for the away teams are influencing the outcome of the match for the decision tree. While MDI may be quicker to calculate the feature importance, it is biased towards predictors with high cardinality that have more unique values in a column compared to other variables. Thus, the permutation-importance method may be more reliable, even though it is computationally more expensive. .

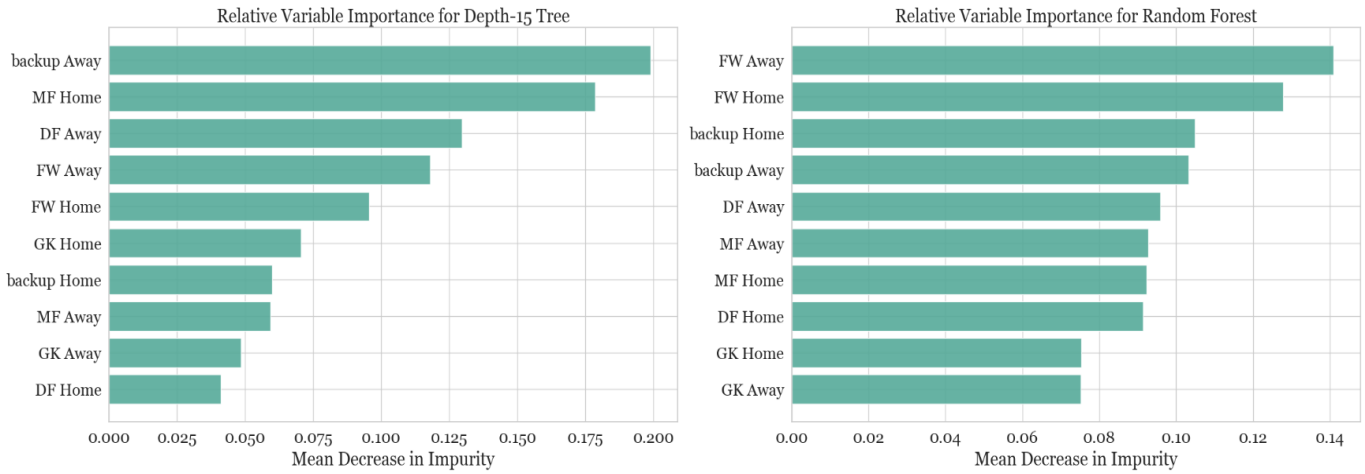


**Figure B: Mean Decrease in Impurity Feature Importance for Decision Tree (Left) & Random Forest (Right)**

| Weight | Feature |
|---|---|
| 0.1577 ± 0.0645 | MF Home |
| 0.1269 ± 0.0601 | FW Away |
| 0.0365 ± 0.0453 | FW Home |
| 0.0346 ± 0.0300 | MF Away |
| 0.0279 ± 0.0264 | GK Away |
| 0.0279 ± 0.0250 | backup Home |
| 0.0202 ± 0.0407 | backup Away |
| 0.0202 ± 0.0303 | DF Home |
| 0.0192 ± 0.0530 | DF Away |
| 0.0019 ± 0.0437 | GK Home |

| Weight | Feature |
|---|---|
| 0.0712 ± 0.0324 | FW Home |
| 0.0500 ± 0.0595 | FW Away |
| 0.0385 ± 0.0285 | DF Home |
| 0.0365 ± 0.0363 | MF Home |
| 0.0327 ± 0.0261 | backup Home |
| 0.0231 ± 0.0405 | GK Away |
| 0.0115 ± 0.0331 | backup Away |
| 0.0067 ± 0.0273 | GK Home |
| 0.0000 ± 0.0258 | MF Away |
| -0.0010 ± 0.0303 | DF Away |

**Figure C: Permutation Importance for Decision Tree (Left) and Random Forest (Right)**

## Results and Interpretations

We use the chosen random forest model to simulate the 2022 World Cup. In the group stage, we decide the result of each match based on the maximum probability out of the three classes 0, 1, and 2, where 0 indicates the probability of the away team winning, 1 is the probability of a draw, and 2 is the probability of the home team winning. We calculate the points of each of the four teams in each group (1 for a draw and 3 for a win), and then take the two highest scoring teams to the next round from each group — the round of 16 teams. From this round onwards, for each match, we compare the probabilities of the home and away team winning, and advance the one with the higher probability, since draws are not accepted. We follow the same logic by predicting the results of the matches and propagating the results to the next round, until we have a winner. Figure D below shows the simulation results.

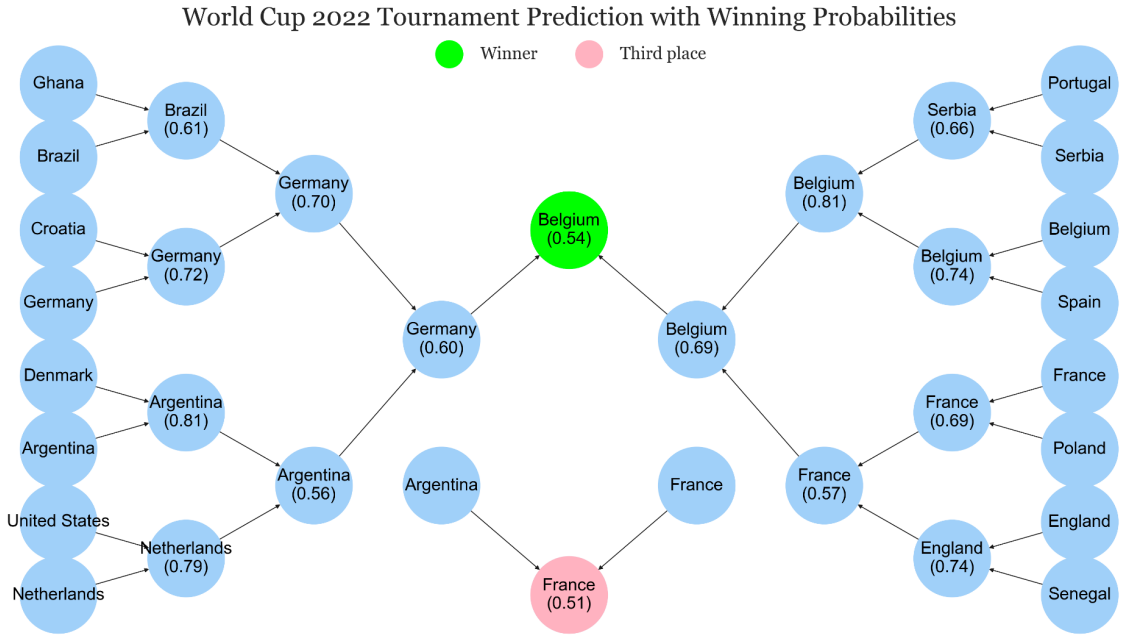


**Figure D. The final model's predictions. The edges point to the winner of the match, and the probability of winning is placed under the team name in between the parentheses.**

As we see the World Cup unfold in real life, half of our semifinals predictions (Argentina and France) are correct, however we see differences in other places in the predicted tournament simulation. This is expected because of two reasons. First, the model is not perfect and the accuracies in table 3 summarize that. Secondly, the decisions are made by taking the highest probability out of the three classes (away win, draw, and home win), but in real life, if any of these probabilities for the three classes were to crystalize (be sampled, assuming life events are random variables with certain distributions and likelihoods) differently, the tree would look drastically different.

## Strengths and Limitations

The main strength of our project was our ability to use historical, team, and player data to feature engineer variables to feed into our model. Our initial model paradigm was to create a score for each of the different positions on the team and an overall strength score of the team. We created the position scores by averaging the top player score for each team and then averaging the position scores with specific weights to create a strength level for the team. Although we did not end up using the overall team strength score, our feature engineering of data allowed us to create predictor variables for our model.

The main limitation that we faced was the scarcity of our data. Since we based our design matrix on particular features of each team, such as the defense, offense, and midfield strength, we could not fetch team-specific scores for old historical matches. For instance, for the 2010, 2014, 2018 World Cups, we were able to aggregate the strengths of the top players, for each of the defense, offense, and midfield components, and combine them into column values for each team. However, after searching the internet for more data, we were unable to find the players for historical matches, neither were we able to find specific scores for each team's defense, offense, and midfield components. This left us unable to populate the data for World Cups that happened before 2010, and hence our dataset was relatively small.

## Future Work

The problem with our dataset was that there were only 416 entries in the dataframe, which led to overfitting on the training data – as we have seen, training scores for decision tree with depth 15 and random forest with depth 8 were nearly 1.00. To combat this problem, we could potentially choose simple models, remove outliers, and utilize confidence intervals instead of relying on point estimates. Choosing simple models may entail presumption of distribution of the predictors as well as their relationship with the outcome, which generally results in higher bias and lower variance. A model with tendency to underfit might generalize better to validation set in our small dataset as it is inherently more likely to be overfitted.

Alternatively, we could augment the data through scraping from different sources. While combining different datasets may cause many missing data, having much more historical data for FIFA may improve estimating the indicators of each team and better predict the outcome of matches. Moreover, regularization such as Lasso and Ridge help to contribute to feature selection and produce smaller coefficient estimates, respectively [3].

## References

[1] “Ordinal Logistic Regression | R Data Analysis Examples.” https://stats.oarc.ucla.edu/r/dae/ordinal-logistic-regression/ (accessed Dec. 11, 2022).

[2] “Key Assumptions of Ordinal Regression,” Jul. 25, 2011. https://www.restore.ac.uk/srme/www/fac/soc/wie/research-new/srme/modules/mod5/3/index.html (accessed Dec. 11, 2022).

[3] ”Breaking the curse of small datasets in Machine Learning: Part 1” https://towardsdatascience.com/breaking-the-curse-of-small-datasets-in-machine-learning-part-1-36f28b0c044d (accessed Dec. 11, 2022).

## Link

Submission Folder :
https://drive.google.com/drive/folders/1l6CiTvyarUActSI2UJo1SXLdI_D_-MzY?usp=share_link

Code:
https://drive.google.com/file/d/1EDf6C_Anl2sq7V3oz_INSj2tt1GWK_cj/view?usp=share_link
https://drive.google.com/file/d/1nElLqqIAzrc-En4p4e_28XvgnP9Jf9dq/view?usp=share_link

Video:
https://drive.google.com/file/d/1SK1-0GkPRG-Uw1kZQZZlBanTDveZSZob/view?usp=share_link