Predicting the Oscars

Jason Chan
Brown University, Data Science Initiative
Github Repository

1. Introduction

Every year, millions of viewers tune in to the Academy Awards, popularly known as the Oscars, to watch the world's most prestigious film awards ceremony. Hundreds of movies fight for the top spot of earning the Oscar for Best Picture award, but only one film wins. In this machine learning project, I used a data set containing a variety of information on movies to train a model with the goal of predicting next year's Oscar winner for Best Picture.

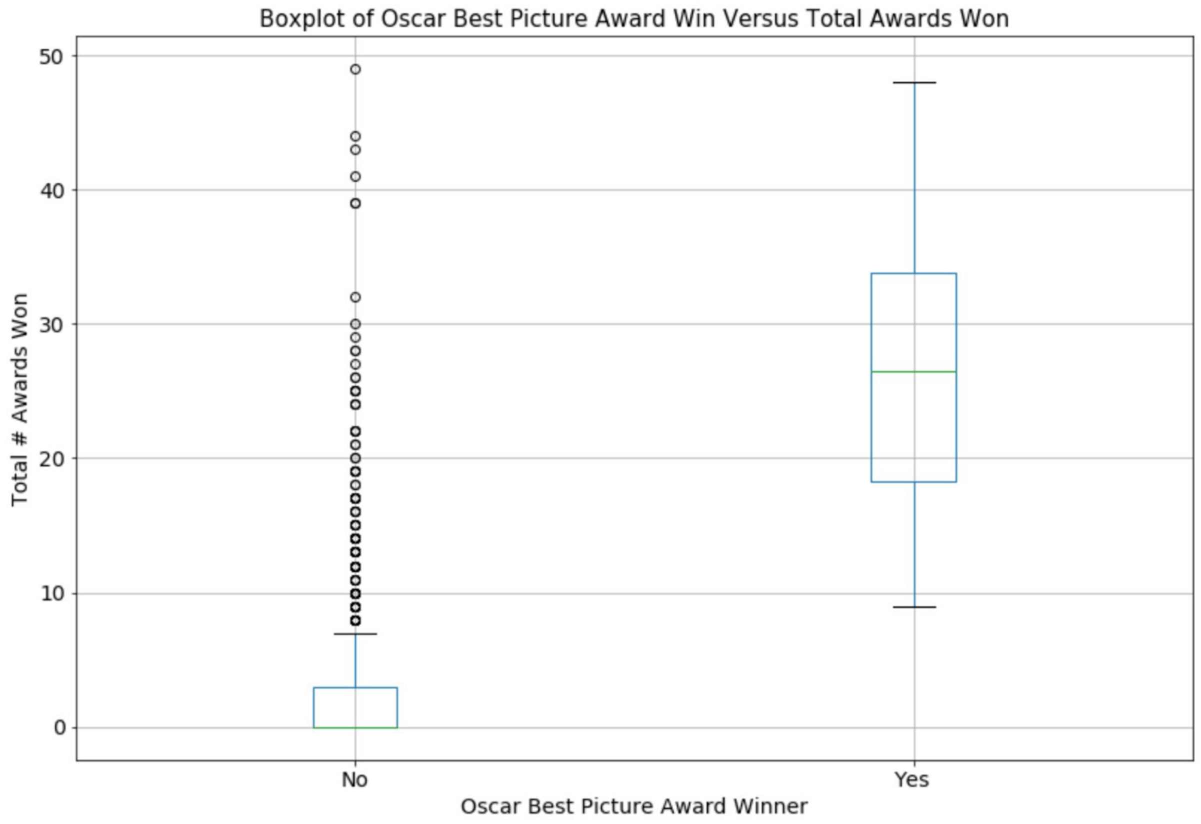
This data set contains a variety of movie attributes. These fields contain information such as duration, genre, gross, user and critic reviews, etc, as well as a variety of other film awards, such BAFTA, the Golden Globes, Critics Choice awards, etc.

Given that the Oscars is *the* premier movie accolade, and winning an Oscar for Best Picture is considered the best possible recognition in the movie business, people spend a significant time discussing and debating about the results of the Oscars. Being able to predict the winner of the Oscars would be an impressive feat.

2. Exploratory Data Analysis

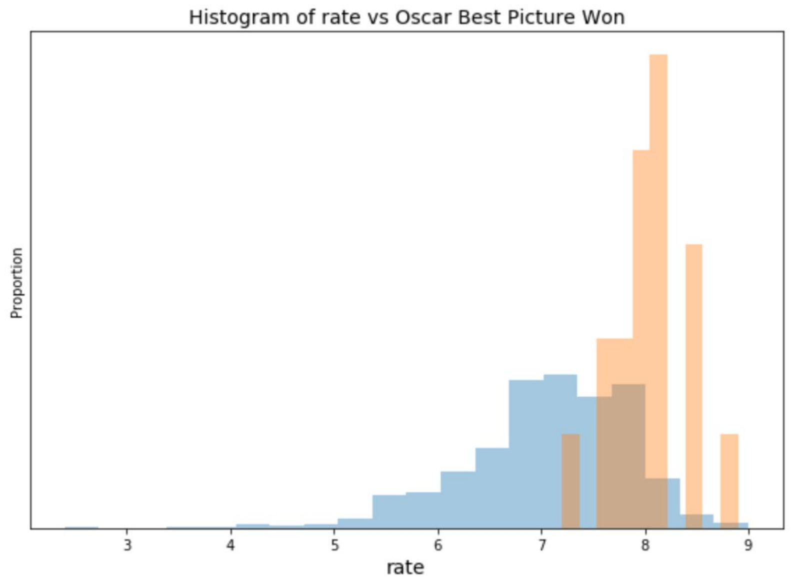
Before we began, we performed preliminary exploratory data analysis, to see relationships between the target variable - winning the Oscar Best Picture, and a variety of other features. Below are several plots that investigate features that appeared to be intuitively important.

Figure 2.1 Boxplot of Oscar Best Picture Winner/Loser Groups vs Total Awards Won



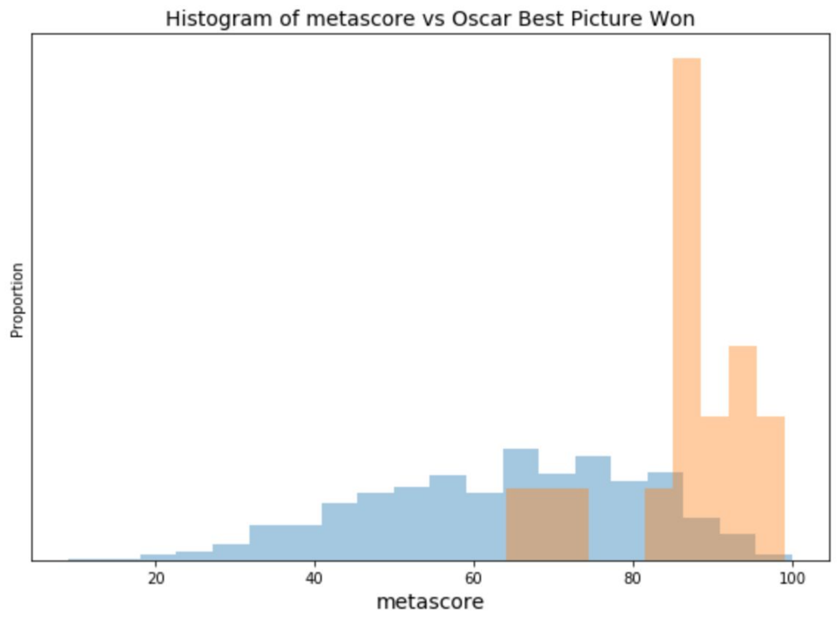
In this figure, we see that there is a stark difference in the distribution of the total number of awards won when comparing films that won the Oscar for Best Picture versus those that did not. A similar plot not shown considering the distribution of the total number of award nominations shows a similar story: Oscar Best Picture award winners consistently received significantly more award nominations and wins than non-winners.

Figure 2.2 Histogram of IMDB rating versus Oscar Best Picture won



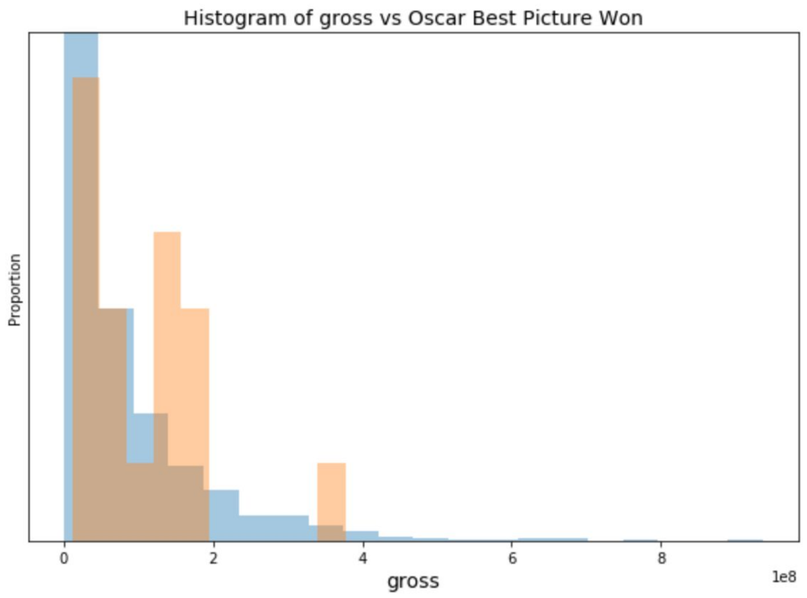
In this plot, we see two histograms imposed on the same plot of the distribution of IMDB ratings. The orange data represents the distribution of winners, and the blue set represents the distribution of non-winners. Again, we see a much higher mean in IMDB rating of winners, with a much smaller standard deviation, while the non-winners distribution has a lower mean and much wider variance.

Figure 2.3 Histogram of metascore (from metacritic) vs Oscar Best Picture Won



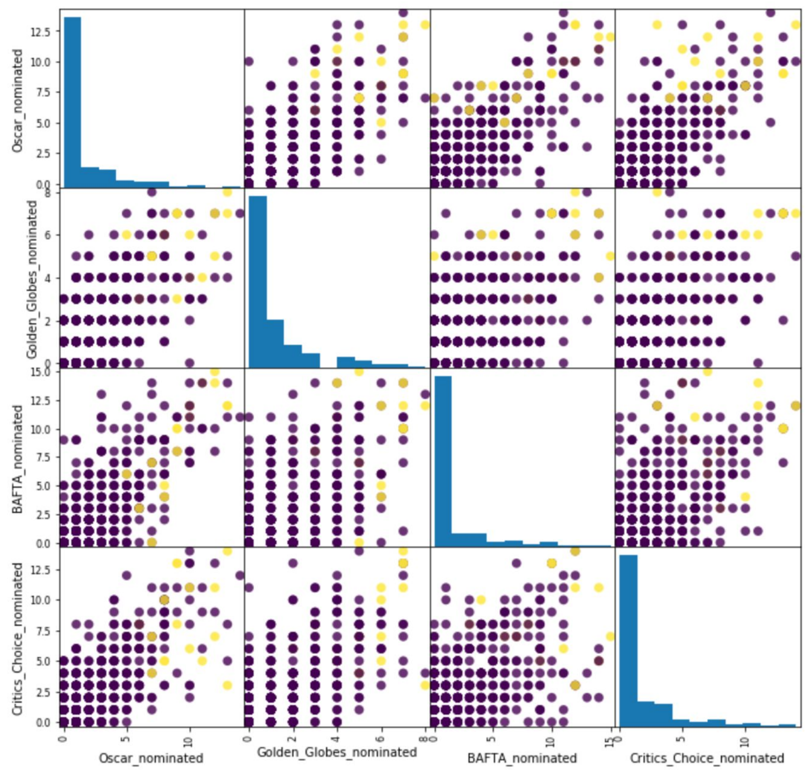
We see a similar story as in Figure 2.2, with again the orange histogram representing the Oscar Best Picture winners, and see a similar trend of a higher mean in metacritic scores.

Figure 2.4 Histogram of gross vs Oscar Best Picture Won



Surprisingly, it seems that there is not a clear visual difference between the histograms of gross of Oscar Best Picture winners versus non-winners

Figure 2.5 Scatter matrix of important film award nominations



This scatter matrix of important film award nominations includes off-diagonal graphs showing correlation scatter-plots where yellow-points represent winners and purple-points represent non-winners. The yellow points appear to generally sit in the upper right-hand side of the plots, indicating a linear correlation. The plots on the diagonals are histograms of the data.

3. Methods

3.1 Preprocessing

Before preprocessing, the shape of the data was (1235, 69). Numerical features such as gross, number of user reviews, IMDB ratings, were preprocessed using a standard scaler. Categorically binary features (Yes/No) were converted to (1/0).

Most importantly, there were a variety of categorical features such as genre, film award nominations/won which had data in the form of strings such as: “ Action|Adventure|Sci-Fi” or “Best Song|Best Composer|Best Director|Best Picture”, etc. A custom transformer was created to split all the data in these features into one-hot encoded matrices that included every possible value in that feature. This was performed before hyperparameter tuning and model training/comparison, due to the complex nature of the preprocessing. Thus, it may be considered a source of data leakage. Missing data was dealt with using multivariate imputation.

Table 3.1.1 Features with missing values and missingness percentage.

Feature	**Fraction of Values Missing**
metascore	0.023482
gross	0.034008
user_reviews	0.011336
critic_reviews	0.008097
popularity	0.109312
Total Missing Points	0.12874493927125505

3.2 Machine Learning Pipeline

In order to determine the best ML models and tune hyperparameters, I utilized traditional train test splitting to produce a test set and a cross-validation set to be used with stratified 4-fold split. GridSearchCV wasa used to tune hyperparameters and determine the best estimators and corresponding test scores. This process was repeated 6 times for each type of model separately. The processor within the pipelines contained standard scalers and multivariate imputers (unless specified as non-imputed for XGBoost).

I trained and tested three types of models, with hyperparameters:

- Logistic Regression
  - `C: np.logspace(-2,4,10))`
- Random Forest
  - `max_depth: [2,5,10,50,100]`
  - `min_sample_split: [2,7,12,17,22]`
- XGBoost with imputation
  - `colsample_bytree: [0.75, 1.0]`

    - `max_depth: [4,6,8]`
    - `min_child_weight: [2,5,10]`
- XGBoost without imputation
    - `colsample_bytree: [0.5,0.75,1.0]`
    - `min_child_weight: [2,5,10]`
    - `max_depth: [2,4,6,8]`
    - `subsample: [0.5, 0.75, 1.0]`
    - `learning_rate: [0.01, 0.05,0.1]`

Due to the highly imbalanced dataset (1217 non-winners, 18 winners), we cannot use accuracy to evaluate our results. Instead, we used the following metrics that conveyed stronger information about success of imbalanced classification:

- Precision, percentage of positive classifications that were correct
- Recall, percentage of positive point that were correctly classified
- Average Precision, area under curve of the Precision-Recall curve
- F1, weighted harmonic mean of precision and recall

The uncertainties from splitting and non-deterministic methods were measured by performing multiple iterations of each gridsearch pipeline, averaging metrics over iterations and finding the standard deviation of metrics.

4. Results

4.1 ML Algorithm Scores

Table 4.1.1 Logistic Regression Results, best hyperparameters: `C = 1.0`

Metric	**Mean**	**Standard Deviation**
Average Precision	0.35	0.2
Precision	0.64	0.18
Recall	0.5	0.2
F1	0.54	0.2

Table 4.1.2 Random Forest Results, best hyperparameters: `max_depth = 5.0, min_samples_split = 2.0`

Metric	**Mean**	**Standard Deviation**
Average Precision	0.1	0.12
Precision	0.33	0.47
Recall	0.08	0.12
F1	0.13	0.19

Table 4.1.3 XGBoost *with* Imputation, best hyperparameters: `colsample_bytree = 0.75, max_depth = 4, min_child_weight = 2`

Metric	**Mean**	**Standard Deviation**
Average Precision	0.3	0.25
Precision	0.67	0.37
Recall	0.33	0.24
F1	0.43	0.27

Table 4.1.4 XGBoost *without* Imputation, best hyperparameters: `colsample_bytree = 1.0, max_depth = 2, min_child_weight = 2, learning_rate = 0.01, subsample = 1.0`

Metric	**Mean**	**Standard Deviation**
Average Precision	0.39	0.11
Precision	0.69	0.16
Recall	0.58	0.19
F1	0.59	0.12

XGBoost without imputation provided the result with the highest score and lowest standard deviation of 0.39 ± 0.11. It was also the fastest model to run, since multivariate imputation was not needed which was computationally heavy.

Typically, when comparing models, we also compare them to the baseline model, which predicts the most frequent class for all points. However, in the highly imbalanced classification scenario, it is not useful to discuss the baseline model. Given the imbalance of our data set, a baseline model would have resulted in: $TP = 0,\ FP = 0,\ TN - 1217,\ FN = 18$. In our case, since we are considering precision, recall and f1-score, what we'd see is that:

- Precision = $TP/(TP + FP) =$ `DIVIDE BY ZERO ERROR`
- Recall = $TP/(TP + FP) = 0$
- F1-Score = $2\,P * R/(P + R) =$ `N/A` because precision cannot be calculated
- Average precision, due to its dependence on precision, cannot be calculated.

Therefore, the only meaningful metric to consider compared to the baseline would be recall, but that was not the primary evaluation metric that we considered.

4.2 Feature Importances

In order to investigate feature importances, we utilized XGBoost's internal feature importance generation methods - in particular, `xgboost.plot_importances().`

XGBoost stores three different types of importances: weight, coverage, and gain. Weight measures the relative number of times a particular feature occurs in the trees of the model. Coverage measures the relative number of observations related to the feature. Gain measures the relative contribution of each feature to the model by calculating each feature's contribution for each tree in the model. A higher value of this metric as compared between features indicates greater importance in generating the prediction.

In our case, gain is the most relevant feature to interpret the importance of our model, as it represents the increase in accuracy brought by a feature to the branches it is on. Total gain measures the gain effects over the whole ensemble method.

As we can see from the plots below, in general it appears that the most important features were the number of Critics Choice Awards won, followed by number of BAFTA nominations, followed by number of IMDb votes. In the context of prediction the Oscar Best Picture winners, what these results mean is that the increase in the number of Critics Choice awards won, or BAFTA nominations, or IMDb votes, results in the most significant increases in probability of winning the Oscars Best Picture. In other words, these most important features listed should give us strong indicators for a film that may be successful in winning the award - the higher the values of the features, the better. This fits our intuition perfectly, as most successful movies typically win more awards, have higher ratings, etc.

Figure 4.2.1 XGBoost Feature Importance Plot: Weight Importance

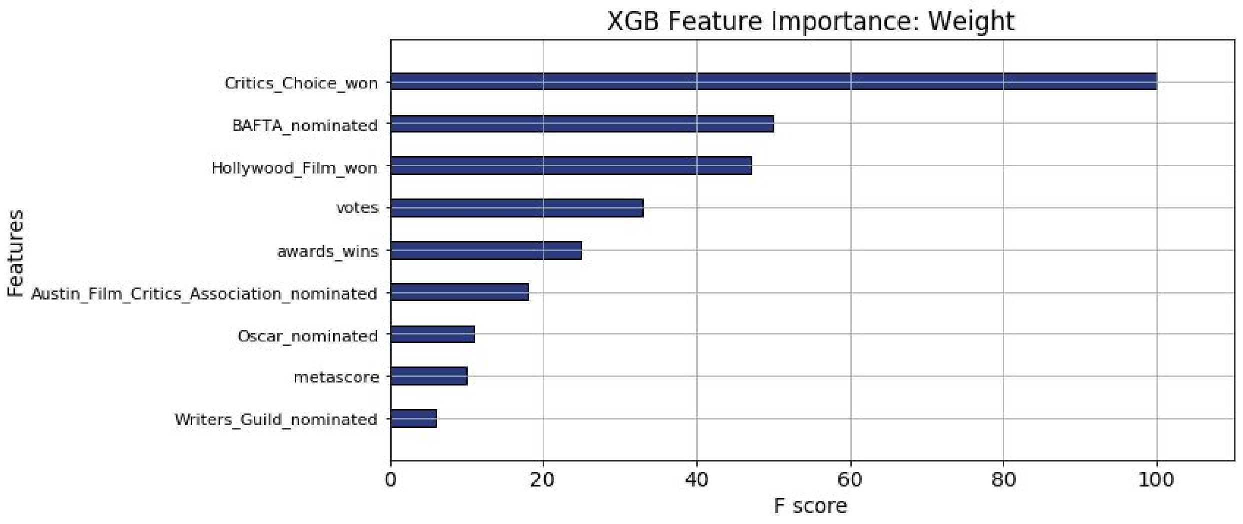


Figure 4.2.2 XGBoost Feature Importance Plot: Coverage Importance

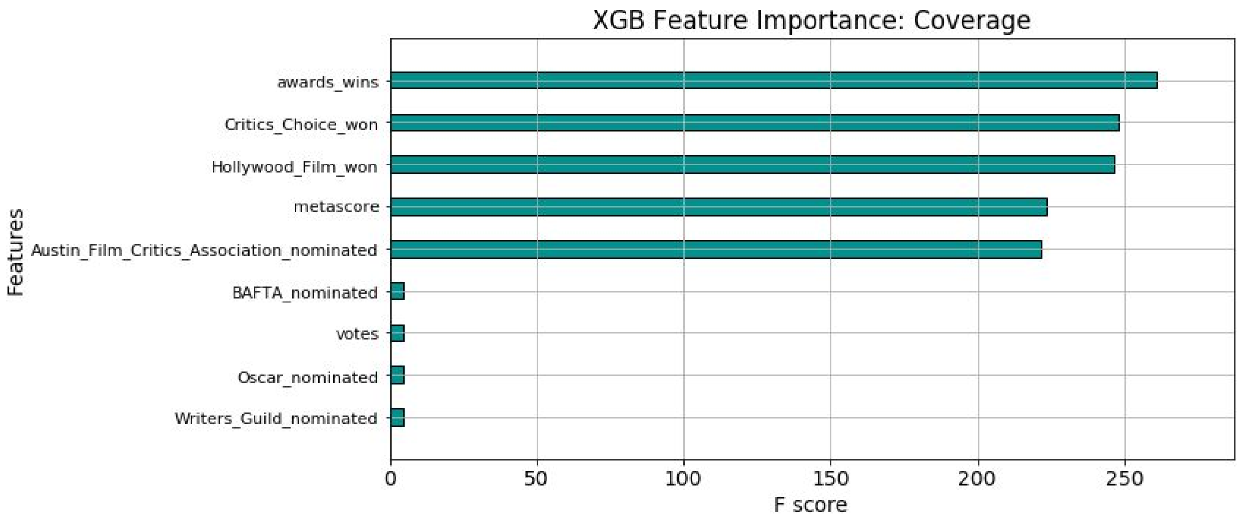


Figure 4.2.3 XGBoost Feature Importance Plot: Gain Importance

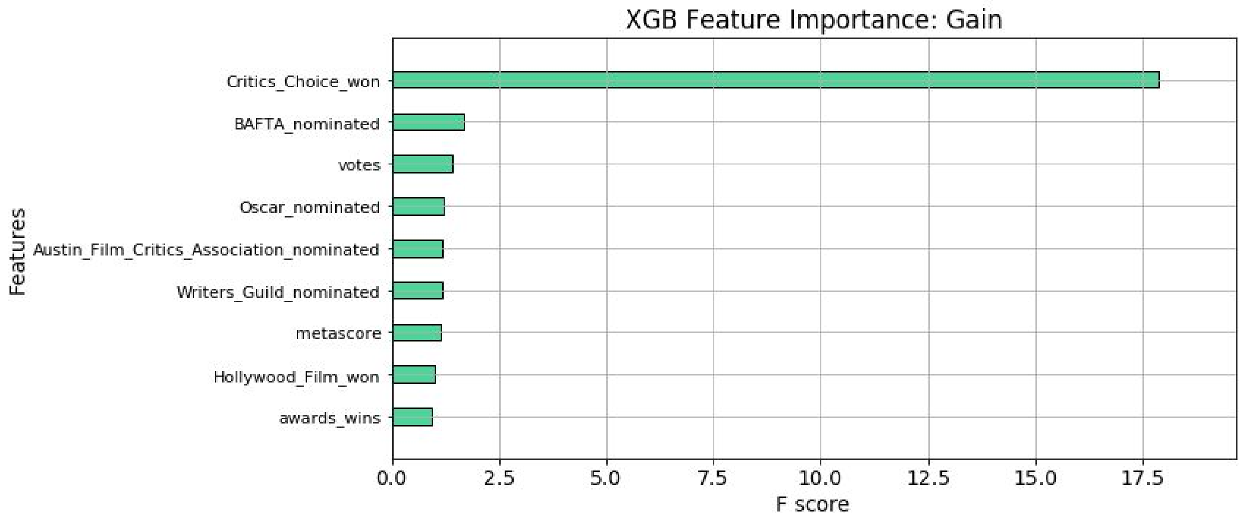
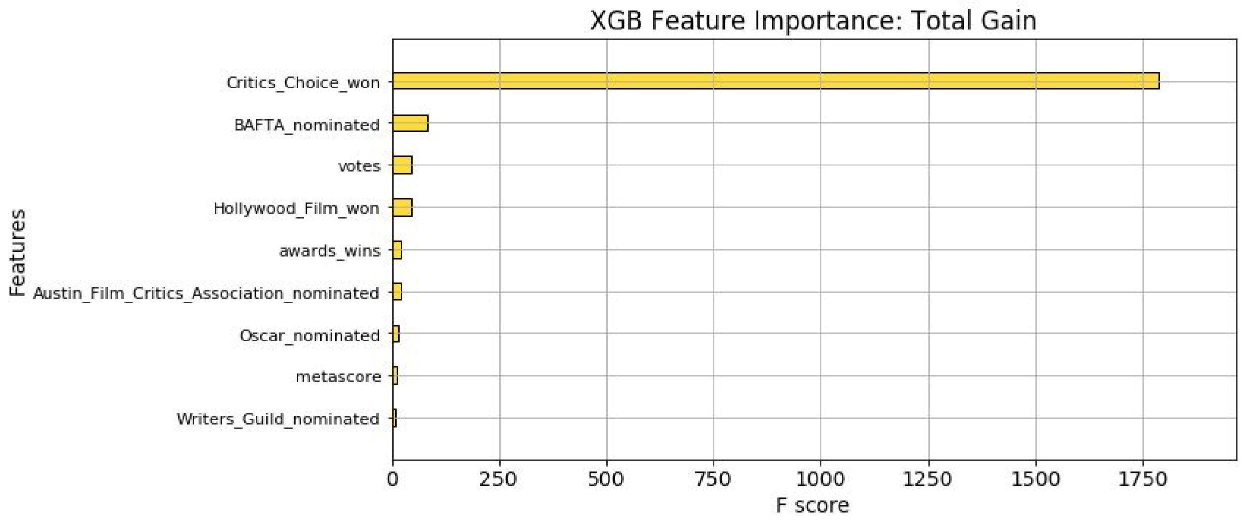


Figure 4.2.4 XGBoost Feature Importance Plot: Total Gain Importance



4.4 Real Life Testing on 2019 Data Results

In order to test the results of my data, I tested my final XGBoost model on previously unseen data for films considered in the 2019 Oscars. Of the films nominated for best picture, the outputted Best Picture win probabilities led to this prediction ranking:

Table 4.4.1 Ranking of outputted prediction probabilities for 2019 Best Picture nominations.

Movie	**Win Score**
Roma	45.9
The Favourite	25.1
Black Panther	21.1
A Star Is Born	18.9
Green Book	18.5
Vice	18.5
Bohemian Rhapsody	18.5
BlacKkKlansman	18.5

My model predicted that of the nominated films, the one most likely to win was **Roma**. In fact, the **real life winner** of the 2019 Oscar for Best Picture went to **Green Book**.

However, what's interesting is that if we consider the widespread discussion in 2018 on the 2019 Best Picture predictions, the clear favorite and established prediction among experts and amateur film and media circles for the winner of the award *was indeed Roma*. A quick google search showed that the following prestigious institutions all published predictions that selected Roma to be the winner of the award: 1.NY Times, 2.Screen Rant, 3.Collider, 4.CNET, 5.Fortune, 6.Rolling Stone, 7.Vanity Fair, 8.Gold Derby, 9.Metacritic, 10.Good Morning America

Further investigation shows that Roma's failure to win was likely due to it being a foreign-language film (no foreign-language film has won Best Picture), and because it was produced by Netflix (a streaming service considered to be "buying" it's way into an Oscar). Because of this context, I believe that my model provided a confident prediction.

5. Outlook

There were several issues encountered in this process that may be revisited. The large number of dimensions (over 1000) may be detrimental to ML algorithms. We can try dimensionality reduction (e.g. PCA, TSNE, SelectKBest with F-test/mutual information) to reduce dimensions and possibly improve the model and make the output more interpretable. We can also try tuning up L1/elastic-net parameters, or using a reduced feature model.

We only used stratified sampling, but we could try other resampling techniques. Given our small number of training points, we may want to oversample the minority class, such as using SMOTE, or combine a variety of resampling techniques.

Limited resource constraints (time, computation power) hindered the brute force method of GridSearchCV. With an upgrade using more servers, distributed computing, stronger processing power, hyperparameter tuning can be improved.

Most importantly, I would like to extend this model to classify other Oscar Awards such as Best Director/Actor/Actress, etc. as well as collecting the full set of data from 2019 films to predict the 2020 Oscars results!

7. References

1. “Check out This Dataset ‘Movies 2000-2018.’” BigML.com - Machine Learning Made Easy, https://bigml.com/user/academy_awards/gallery/dataset/5c6886e1eba31d73070017f5.
2. “How Do You Attack a Machine Learning Problem with a Large Number of Features?” Dr. Sebastian Raschka, 16 Nov. 2019, https://sebastianraschka.com/faq/docs/large-num-features.html.
3. Phunter. “Xgboost with GridSearchCV.” Kaggle, Kaggle, 28 Jan. 2016, https://www.kaggle.com/phunter/xgboost-with-gridsearchcv.
4. tilii7. “Hyperparameter Grid Search with XGBoost.” Kaggle, Kaggle, 13 Oct. 2017, https://www.kaggle.com/tilii7/hyperparameter-grid-search-with-xgboost.
5. Jain, Aarshay. “Complete Guide to Parameter Tuning in XGBoost (with Codes in Python).” Analytics Vidhya, 13 Sept. 2019, https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/.
6. Cambridge Spark. “Hyperparameter Tuning in XGBoost.” Medium, Cambridge Spark, 23 Mar. 2018, https://blog.cambridgespark.com/hyperparameter-tuning-in-xgboost-4ff9100a3b2f.
7. Brownlee, Jason. “Feature Importance and Feature Selection With XGBoost in Python.” Machine Learning Mastery, 21 Aug. 2019, https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/.
8. “(Tutorial) Learn to Use XGBoost in Python.” DataCamp Community, https://www.datacamp.com/community/tutorials/xgboost-in-python.
9. Abu-Rmileh, Amjad. “Be Careful When Interpreting Your Features Importance in XGBoost!” *Medium*, Towards Data Science, 20 Feb. 2019, https://towardsdatascience.com/be-careful-when-interpreting-your-features-importance-in-xgboost-6e16132588e7.