Predicting positions at finish for derbies via multiclass classification

Seung Woo Choi

Abstract:

In this project, my goal was to predict positions at finish for derbies utilizing multiclass classification models. Big data that contained over 5.2 million observations was used to perform the analysis, and many predictors were categorical. Feature engineering played a large role in facilitating the data mining and statistical learning process. While statistical models like multinomial logistic regression and k-nearest neighbors were considered, I built ensemble models, such as random forest, AdaBoost, and extreme gradient boosting (a.k.a. XGBoost), to make multiclass predictions on the data. The results showed that the XGBoost classifier had the best overall performance compared to the other methods. Based on the results of the XGBoost classifier, the most important features were determined to be (1) odds, (2) jockey_ranked, (3) purse, (4) run_up_distance, and (5) post_time using the Gini index.

Introduction:

Watching a derby is a common avenue for entertainment in the U.S. A derby is essentially a type of horse race in which, on average, 8 jockeys compete while riding on young horses for 1.25 miles ("The Race"). The Kentucky Derby is an example of a horse race with which many are familiar. What makes these derbies even more exciting is that many people participate in wagers for the races. According to an article published by BetFirm, the total amount of money bet on the Kentucky Derby in 2021 was about $233.0M, which is a huge sum of money (Jones). Thus, the motivation behind this project was to not only inform participants of these bets of the types of features that are important when making a decision but also caution them to practice risk management behaviors.

To set the (hypothetical) context, I work as a business consultant for a risk management firm that provides services to clients who place bets on derbies. One of my main duties is to ensure that clients of the firm practice proper risk management techniques when placing bets in derbies.

The primary research question that this project seeks to explore is: **Can a multiclass classification model be built to predict the position at finish for real-life horse races with 8 classes?** In addition to this, a supporting research question of interest is: What are the most important features when it comes to predicting the position at finish for a derby? My discussion will answer these two questions based on the analysis and results.

The following points outline the limitations of my project scope: (1) Time series analysis was not considered, (2) Horse welfare is important but the dataset does not contain enough relevant features to explore this topic, and (3) Cloud computing resources were not used due to budgetary constraints.

The most relevant data mining challenges and the problem solving strategies I employed to handle them are detailed as follows. The first challenge was that I initially tried to implement the code in R, but R ran into error messages that stated that the vector memory was exhausted when fitting the various models. Thus, I had to implement my code in Python instead. The second challenge was that even with Python, fitting the models took a long time to run. For example, certain iterations of the random forest classifier took 56 minutes while other model fits took even longer and could not converge. I explored fitting models that used pipelines, grid searches, and cross validations but ultimately resorted to

running standard model fits with engineered features when possible to ensure that the models could be fitted in a reasonable time period. The third challenge was that two models I originally sought to fit were multinomial logistic regression and k-nearest neighbors. Unfortunately, both models could not be fit, so I used 3 different types of ensemble methods that could fit the data with decent computation times and performances. Lastly, the fourth challenge was that feature engineering required lots of nuance since 7 features were categorical variables and each had varying degrees of cardinality. I applied various methods that were appropriate for each individual variable to ensure that the feature could be useful as a model input.

Through this real life data mining project, I learned about the importance of the inputs of a model (e.g. how to apply feature engineering to transform categorical variables into useful numeric inputs), how to carefully select the evaluation metric to capture the problem at hand, and how to weigh the benefits of using a programming language (e.g. R and Python) to solve a problem involving big data and multiclass classification. By applying different types of data mining and statistical learning methods, I was able to construct models that can predict the positions at finish for derbies.

In the rest of the report, I will discuss the data source, proposed methodology, analysis and results, and conclusions.

Data Source:

The data used in this report came from the Big Data Derby 2022 competition on Kaggle (link) and was provided by many sponsors, including NYRA, NYTHA, Equibase, The Jockey Club, The Breeder's Cup, KTA, and TOBA ("Big Data Derby 2022"). The data includes 4 csv files - 3 csv files with data on the horses, jockeys, and racetracks and 1 csv file with combined data. The combined data has 18 features, 11 of which are numeric features and 7 of which are categorical features. The data can be considered big data since over 5.2 million observations were included in the dataset. Fortunately, there are no missing values in the dataset. Figure 1 below displays a description of some of the features.

Figure 1. Description of some features in the data

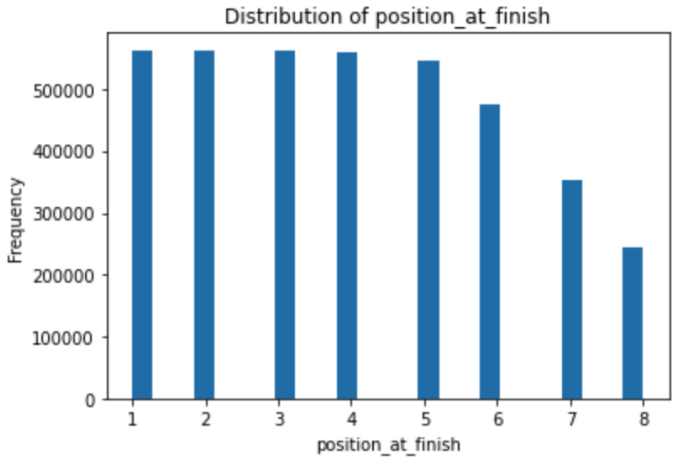
Feature	Description
odds	Odds to 1, odds of winning the race
jockey	Name of the jockey on the horse
purse	Purse of the race (in USD)
run_up_distance	Distance from the gate to the start of the race (in feet)
post_time	Time the race began (as character)

The response variable is position_at_finish, which contains 8 values (i.e. 1, 2, 3, 4, 5, 6, 7, and 8) that represent 8 classes. The original range was 1 to 14, but the data was filtered to include classes less than or equal to 8 due to the fact that 8 is the average number of horses and jockeys on the field and there

were imbalances and sparsity in the response variable for high values (i.e. 9 to 14). This modification resulted in a removal of 7.6% of the original data.

Figure 2 illustrates the distribution of the 8 classes for the response variable. It is clear from the plot that the classes have slight imbalances, with classes 1-5 consisting of higher than average (i.e. the expected proportion of each class is 12.5%) proportions and classes 7-8 consisting of lower than average proportions. Class 6 has a proportion close to the expected proportion.

Figure 2. Histogram of the position_at_finish



Methodology:

The methodology used in this project was informed by the insights from my exploratory data analysis (EDA) and focuses on two aspects - feature engineering and model building. The feature engineering component required understanding the nuance of each categorical feature and was applied separately on the training and testing data, whereas the model building components were relatively simple once viable models were determined.

A random 80:20 train/test split was performed on the full data. The summary statistics show that the mean position_at_finish was 4.04, and the standard deviation was 2.12. In terms of the class proportions, classes 1 to 5 made up 14.1 to 14.6% of the data per class, class 6 made up 12.3% of the data, class 7 made up 9.1% of the data, and class 8 made up 6.3% of the data.

In terms of feature engineering, categorical features were modified to provide better numerical inputs for the statistical models. Simple data type and unit conversions were applied to race_number, purse, and odds, and the race_date column was dropped since time data was not needed for the analysis. The categorical variable, track_id, was dummy encoded into 3 columns (i.e. AQU, BEL, and SAR) due to its low cardinality. Since course_type and track_condition were categorical features with low cardinality (i.e. they still had more values than track_id where it didn't make sense to dummy encode them), they were frequency encoded and normalized as is. The categorical variables, race_type and program_number, had numerous infrequent values, so they were both frequency encoded and normalized with an "Other"

value for infrequent values. Lastly, I rank-encoded the variable, jockey, by mean position_at_finish, which means that each jockey's mean position_at_finish was computed and then the jockeys were ranked in order with the lowest rank being the best. For each feature engineered variable, the transformation was applied separately on the training and testing data to prevent leakage into the testing set.

Interaction terms and polynomial features were considered but not implemented given the limits of the laptop's specifications, the large size of the data, and the potential impact to run times and time complexity of the models used when the number of features increases.

Figure 3 shows the initial heat map of position_at_finish before feature engineering was implemented. Notice that there are 10 features (excluding position_at_finish) in the original heat map and only odds had a decent correlation coefficient of 0.4.

Figure 3. Initial heat map of position_at_finish (before feature engineering)

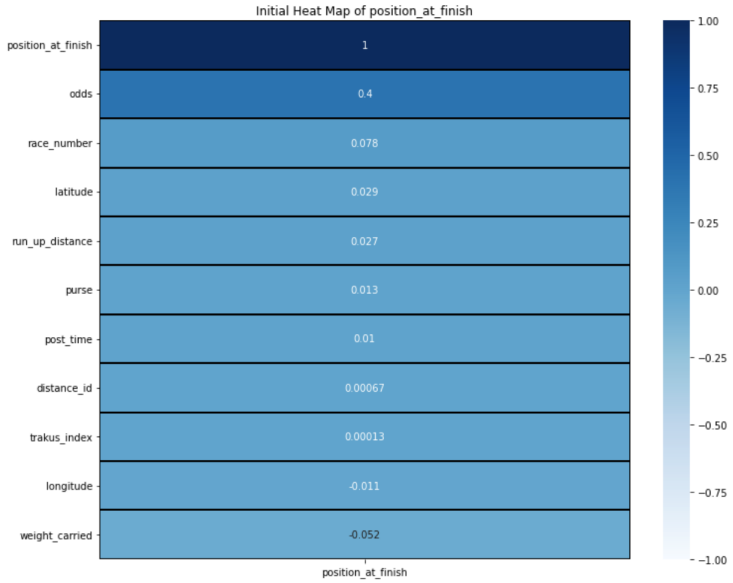
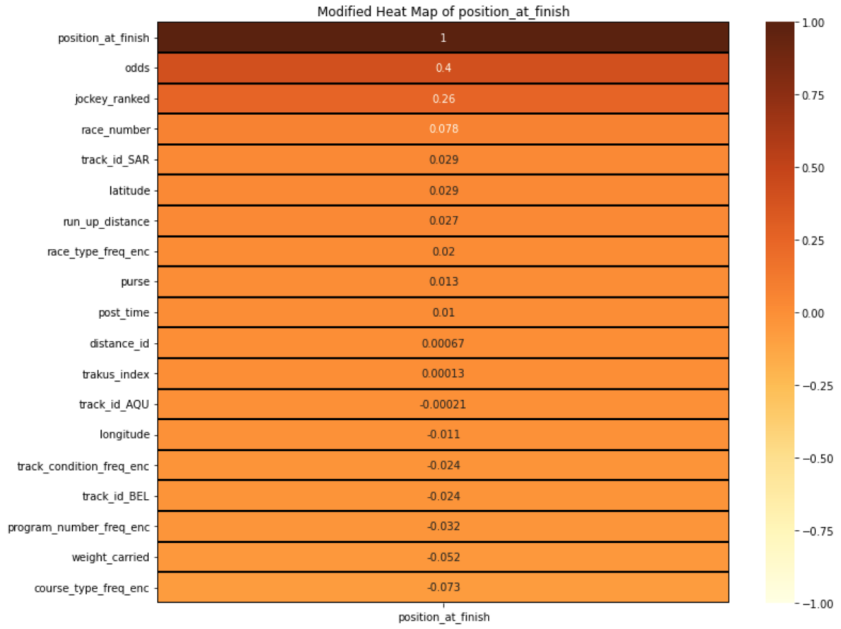


Figure 4 shows the modified heat map of position_at_finish. There are now 18 variables (excluding position_at_finish), and jockey_ranked also has a decent correlation coefficient of 0.26.

Figure 4. Modified heat map of position_at_finish (after feature engineering)



The models were built mostly with standard parameters, but some parameters were tuned to enhance model performance and shorten computation time. Table 1 below describes the model builds for the best version of each model.

Table 1. Model descriptions

Model	Description
AdaBoost Classifier	An ensemble method that utilizes the gradient descent algorithm and weak learners to build a strong classifier. 9 features were included. Tuned the model parameter for estimator. Performed cross validation to validate results.
Random Forest Classifier	An ensemble method that combines the bagging algorithm with random feature selection. 11 features were included. Modified the model parameters for n_jobs, n_estimators, min_samples_leaf, and oob_score. Performed cross validation to validate results.
XGBoost Classifier	An ensemble method that applies extreme gradient boosting to enhance speed and accuracy. All features were included. Modified the response variable to fit model input and the model parameter tree_method. Performed cross validation to validate results.

For the AdaBoost classifier, I adjusted the features to improve the computation time and tuned the parameter for the estimator to improve the model's performance on the evaluation metric. The features - odds, jockey_ranked, weight_carried, program_number_freq_enc, race_number, distance_id, race_type_freq_enc, latitude, course_type_freq_enc - were specified in the model to decrease the

computation time (i.e. relative to including all the features in the model). I also tuned the estimator parameter by testing 3 different max_depth values to determine which value resulted in the best model fit. I then performed cross validation on the entire data using the optimal max_depth value to validate the results.

For the random forest classifier, I adjusted the features and model parameters to improve the model. The predictors - odds, jockey_ranked, purse, program_number_freq_enc, run_up_distance, post_time, weight_carried, race_number, distance_id, race_type_freq_enc, and track_condition_freq_enc - were included in the model. The model parameters were set as: random_state=7406, n_jobs=-1, n_estimators=50, min_samples_leaf=20, and oob_score=True. Since the model already performed at a high level, the computation time was the only output that was improved by setting specific values for these parameters. Cross validation was also performed to validate that the evaluation metric observed was actually observed.

For the XGBoost classifier, the main changes that were made include modifying the response variable to fit the required model input and setting the parameter tree_method to “hist”. The latter had an enormous impact on the computation time and even resulted in a slight improvement in the evaluation metric. Cross validation was then performed to validate the observed results.

Analysis and Results:

The model performances were evaluated using the weighted average one-vs-rest ROC AUC score. The ROC AUC scores and F1 scores can be great evaluation metrics for multiclass classification problems. Since the class proportions for the response variable did not have a high imbalance, the ROC AUC score was a better fit for the analysis. While averaging methods such as micro, macro, and weighted were all considered, the weighted average technique was selected to incorporate class imbalances/frequencies in the calculations of the evaluation metric. Lastly, one-vs-rest (OvR) was chosen over one-vs-one since the latter tends to have computational challenges.

In addition to the weighted average one-vs-rest ROC AUC score, I utilized the computation time to assess the practicality of each model. The computation time measures how long it takes for the model to be fitted. This is an important measure given that the dataset contains about 5M observations and some multiclass classification models either cannot be fitted or take large amounts of time where it no longer becomes practical to use the model. In some cases, using the default parameters resulted in computation times close to 1 hour. Modifying various parameters with the intent to decrease computation time allowed the final iteration of each model to be computed quickly.

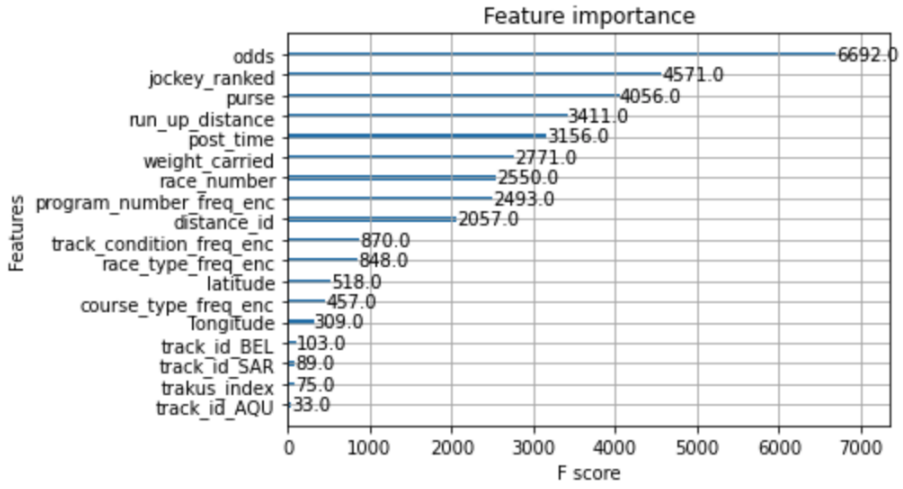
Table 2. Model results

Model	Computation Time (seconds)	Weighted-average OvR ROC AUC Score
AdaBoost Classifier	304.525	0.64
Random Forest Classifier	152.160	≅1.00
XGBoost Classifier	133.997	0.97

The results from executing the proposed data mining methods are displayed above in Table 2. Table 2 shows that the AdaBoost classifier had the longest computation time and the lowest weighted-average OvR ROC AUC score. The XGBoost classifier had the shortest computation time and the second highest evaluation score. The random forest classifier had a moderate computation time but the highest evaluation score.

The optimal model was determined to be the XGBoost classifier. This is because the model had the shortest computation time, a weighted-average OvR ROC AUC score close to that of the random forest classifier, and a single modified parameter that may make the model more generalizable for new, unseen data. The feature importances were identified using the XGBoost classifier and the Gini index, which essentially indicates that higher scores equate to more importance. Figure 5 displays the most important features, which were odds, jockey_ranked, purse, run_up_distance, and post_time. The random forest classifier also showed that the odds, purse, jockey_ranked, run_up_distance, and post_time were the most important features.

Figure 5. Feature importances using the XGBoost classifier and Gini index



The computer softwares used to implement my data mining methods can be found in the **Bibliography and Credits** section.

Conclusions:

Overall, the optimal model is the XGBoost classifier with a score of 0.97 on the evaluation metric and a computation time of 133.997 seconds. Despite having a better weighted-average OvR ROC AUC score (i.e. approximately 1.00), the random forest classifier was not selected as the best model because the computation time was longer (i.e. 152.160 seconds), tuning the parameters to find the right ones also requires time and resources, and the model only resulted in a marginally higher evaluation score. The worst-performing model was the AdaBoost classifier with a score of 0.64 on the evaluation metric and a computation time of 304.525 seconds. The short computation time, high weighted-average OvR ROC AUC score, and the practicality of having few tunable parameters makes the XGBoost classifier the best

of the three models. Based on the optimal model, the most important features were identified as: (1) odds, (2) jockey_ranked, (3) purse, (4) run_up_distance, and (5) post_time.

From the analysis and results, the XGBoost classifier performs very well in terms of the weighted average one-vs-rest ROC AUC score and has a relatively short computation time. I would, however, advise the model to be used with caution, especially in the aforementioned business context, since there is always the risk of losing your entire wager, the model may be overfitting given the large dataset, and features with high cardinality can result in misleading feature importances.

Future work could include analyzing the data to find insights that can improve horse welfare. As mentioned in the **Introduction** section, this was not possible in my project because the dataset did not contain enough pertinent variables to investigate this topic. The findings of such research could be utilized to improve the health and condition of the horses that participate in derbies.

Lessons I have learned:

This project presented many challenges that helped me to learn valuable lessons. The first and immediate challenge was working with big data on a local machine. While I could have used a cloud computing resource like AWS, I decided not to so that I can keep costs at a minimum. I started off working in R, but the code used to load the data, create visualizations, and model the data were all problematic since R took a long time to execute the code and could not handle the large dataset when it came to modeling. To overcome this hurdle, I had to take a step back and try implementing the code in Python instead. Python was much better at handling the dataset and was instrumental in unblocking me. Executing the code for steps like data collection, data cleaning, EDA, and pre-processing were fairly straightforward and doable. The modeling step, however, was a huge struggle since many of the initial models I attempted to build, such as multinomial logistic regression and k-nearest neighbors, would run for a long time but either never finish executing in a reasonable amount of time or finish with an error message about failing to converge. The random forest classifier was the first model that I successfully implemented, but it took about 56 minutes or so to fit the model. Seeing that an ensemble model worked well for this multiclass classification problem, I tried fitting other ensemble models. The AdaBoost classifier was fast but had a decent performance at best, while the XGBoost classifier took about 38 minutes to run but performed well on the evaluation metric. I then researched parameters that could reduce the computation times and managed to bring the computation times down to an average of 196.894 seconds (i.e. 3.28 minutes), which was a huge success since the performance of the models did not decrease as the computation time went down. Lastly, I learned about various methods to apply feature engineering to categorical variables. Each categorical variable had a different degree of cardinality, so there was a lot of nuance to approaching each variable. I applied feature engineering to each categorical feature in the training and testing datasets separately to minimize the risk of data leakage between the two sets of data.

References:

Ali, Moez. “Handling Machine Learning Categorical Data with Python Tutorial”. *Data Camp*, Feb. 2023, https://www.datacamp.com/tutorial/categorical-data.

“Big Data Derby 2022”. *Kaggle*, n.d. https://www.kaggle.com/competitions/big-data-derby-2022/data.

Brownlee, Jason. “Feature Importance and Feature Selection With XGBoost in Python”. *Machine Learning Mastery*, 27 Aug. 2020, https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/.

Brownlee, Jason. “How to One Hot Encode Sequence Data in Python”. *Machine Learning Mastery*, 14 Aug. 2019, https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/.

Brownlee, Jason. “Multinomial Logistic Regression With Python”. *Machine Learning Mastery*, 1 Jan. 2021, https://machinelearningmastery.com/multinomial-logistic-regression-with-python/.

Chaudhury, Srijani. “Tuning of Adaboost with Computational Complexity”. *Medium*, 24 Aug. 2020, https://medium.com/@chaudhurysrijani/tuning-of-adaboost-with-computational-complexity-8727d01a9d20.

“Check if element exists in list in Python”. *Geeks for Geeks*, 22 Feb. 2023, https://www.geeksforgeeks.org/check-if-element-exists-in-list-in-python/.

“Equivalent for R / dplyr's glimpse() function in Python for Panda dataframes?”. *Stack Overflow*, n.d., https://stackoverflow.com/questions/74414355/equivalent-for-r-dplyrs-glimpse-function-in-python-for-panda-dataframes.

“Get frequency of item occurrences in a column as percentage [duplicate]”. *Stack Overflow*, n.d., https://stackoverflow.com/questions/50558458/pandas-get-frequency-of-item-occurrences-in-a-column-as-percentage.

“How do I create test and train samples from one dataframe with pandas?”. *Stack Overflow*, n.d., https://stackoverflow.com/questions/24147278/how-do-i-create-test-and-train-samples-from-one-dataframe-with-pandas.

“How to add labels to a boxplot figure (pylab)”. *Stack Overflow*, n.d., https://stackoverflow.com/questions/31842892/how-to-add-labels-to-a-boxplot-figure-pylab.

“How to Create a Correlation Matrix using Pandas?” *Geeks for Geeks*, 8 Oct. 2021., https://www.geeksforgeeks.org/how-to-create-a-correlation-matrix-using-pandas/

“How to get column names in Pandas dataframe”. *Geeks for Geeks*, 11 Jan. 2023, https://www.geeksforgeeks.org/how-to-get-column-names-in-pandas-dataframe/.

Jones, Jack. “How Much is Bet on the Kentucky Derby”. *Bet Firm*, 6 May 2022, https://www.betfirm.com/how-much-is-bet-on-the-kentucky-derby/.

“Matplotlib, multiple scatter subplots with shared colour bar”. *Stack Overflow*, n.d., https://stackoverflow.com/questions/44970881/matplotlib-multiple-scatter-subplots-with-shared-colour-bar.

Miles, Jessica. “Getting the Most out of scikit-learn Pipelines”. *Towards Data Science*, 29 Jul. 2021, https://towardsdatascience.com/getting-the-most-out-of-scikit-learn-pipelines-c2afc4410f1a.

Mohan, Nishant. “Target Encoding for Multi-Class Classification”. *Towards Data Science*, 21 Aug. 2020, https://towardsdatascience.com/target-encoding-for-multi-class-classification-c9a7bcb1a53.

“Pandas Convert Column to Int in DataFrame”. *Spark By Examples*, 26 Jan. 2023, https://sparkbyexamples.com/pandas/pandas-convert-column-to-int/.

“Pandas DataFrame: rank() function”. *W3resource*, 19 Aug. 2022, https://www.w3resource.com/pandas/dataframe/dataframe-rank.php.

“Pandas Filter Rows by Conditions”. *Spark By Examples*, 18 Jan. 2022, https://sparkbyexamples.com/pandas/pandas-filter-rows-by-conditions/#:~:text=2.-,Filter%20Rows%20by%20Condition,new%20DataFrame%20with%20selected%20rows.&text=You%20can%20also%20write%20the%20above%20statement%20with%20a%20variable.

“Pandas Groupby and Computing Mean”. *Geeks for Geeks*, 26 Nov. 2020, https://www.geeksforgeeks.org/pandas-groupby-and-computing-mean/.

Purge, Gursev. “Performance Comparison of Multi-Class Classification Algorithms”. *Medium*, 28 Dec. 2020, https://gursev-pirge.medium.com/performance-comparison-of-multi-class-classification-algorithms-606e8ba4e0ee.

“Python Color Constants Module”. *Webucator*, n.d., https://www.webucator.com/article/python-color-constants-module/.

R. Sruthi. “Understand Random Forest Algorithms With Examples (Updated 2023)”. *Analytics Vidhya*, 24 Mar. 2023, https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/.

Saini, Anshul. “Master the AdaBoost Algorithm: Guide to Implementing & Understanding AdaBoost”. *Analytics Vidhya*, 3 Mar. 2023, https://www.analyticsvidhya.com/blog/2021/09/adaboost-algorithm-a-complete-guide-for-beginners/.

Sangani, Raj. “Dealing with features that have high cardinality”. *Towards Data Science*, 4 Aug. 2021, https://towardsdatascience.com/dealing-with-features-that-have-high-cardinality-1c9212d7ff1b#:~:text=What%20is%20high%20cardinality%3F,many%20of%20these%20unique%20values.

T., Bex. “Comprehensive Guide to Multiclass Classification Metrics”. *Towards Data Science*, 9 Jun. 2021, https://towardsdatascience.com/comprehensive-guide-on-multiclass-classification-metrics-af94cfb83fbd.

“The Race”. *Kentucky Derby*, n.d., https://www.kentuckyderby.com/history/the-race#:~:text=Try%20watching%20this%20video%20on,horses%20race%20against%20one%20another.

Tripathi, Himanshu. “Different Type of Feature Engineering Encoding Techniques for Categorical Variable Encoding”. *Medium*, 20 Sept. 2019, https://medium.com/analytics-vidhya/different-type-of-feature-engineering-encoding-techniques-for-categorical-variable-encoding-214363a016fb.

“using best params from gridsearchcv”. *Stack Overflow*, n.d., https://stackoverflow.com/questions/41475539/using-best-params-from-gridsearchcv.

Vandeput, Nicolas. “Do You Use XGBoost? There is a 200x Faster Way”. *Towards Data Science*, 29 Jul. 2021, https://towardsdatascience.com/do-you-use-xgboost-heres-how-to-make-it-200x-faster-16cb6039a16e.

Wade, Corey. “Getting Started with XGBoost in scikit-learn”. *Towards Data Science*, 10 Nov. 2020, https://towardsdatascience.com/getting-started-with-xgboost-in-scikit-learn-f69f5f470a97.

Wong, Kay Jan. “Feature Encoding Techniques in Machine Learning with Python Implementation”. *Towards Data Science*, 10 Jan., https://towardsdatascience.com/feature-encoding-techniques-in-machine-learning-with-python-implementation-dbf933e64aa.

“Working with Missing Data in Pandas”. *Geeks for Geeks*, 9 Feb. 2023, https://www.geeksforgeeks.org/working-with-missing-data-in-pandas/.

“5 ways to apply an IF condition in Pandas DataFrame”. *Data to Fish*, 25 Jun. 2022, https://datatofish.com/if-condition-in-pandas-dataframe/.

Technology: Python, Jupyter Notebook
Python libraries: Numpy, pandas, scikit-learn, matplotlib, seaborn, category_encoders, xgboost, time

References (Python libraries):

“Column Transformer with Mixed Types”. *Scikit Learn*, n.d., https://scikit-learn.org/stable/auto_examples/compose/plot_column_transformer_mixed_types.html.

“Cross-validation: evaluating estimator performance”. *Scikit Learn*, n.d., https://scikit-learn.org/stable/modules/cross_validation.html.

“Decision Trees”. *Scikit Learn*, n.d., https://scikit-learn.org/stable/modules/tree.html#tree-mathematical-formulation.

“Feature importances with a forest of trees”. *Scikit Learn*, n.d., https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html.

“How to calculate summary statistics”. *Pandas*, n.d., https://pandas.pydata.org/docs/getting_started/intro_tutorials/06_calculate_statistics.html.

“Linear Models”. *Scikit Learn*, n.d., https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression.

“Merge, join, concatenate and compare”. *Pandas,* n.d., https://pandas.pydata.org/docs/user_guide/merging.html.

“Metrics and scoring: quantifying the quality of predictions”. *Scikit Learn*, n.d., https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter.

“Multiclass and multioutput algorithms”. *Scikit Learn*, n.d., https://scikit-learn.org/stable/modules/multiclass.html.

“Multiclass Receiver Operating Characteristic (ROC)”. *Scikit Learn*, n.d., https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html.

“numpy.unique”. *Numpy*, n.d., https://numpy.org/doc/stable/reference/generated/numpy.unique.html.

“pandas.DataFrame.rank”. *Pandas*, n.d., https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.rank.html.

“pandas.DataFrame.rename”. *Pandas*, n.d., https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.rename.html.

“pandas.factorize”. *Pandas*, n.d., https://pandas.pydata.org/docs/reference/api/pandas.factorize.html.

“pandas.get_dummies”. *Pandas*, n.d., https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html.

"pandas.read_csv". *Pandas*, n.d., https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html.

"pandas.Series.map". *Pandas,* n.d., https://pandas.pydata.org/docs/reference/api/pandas.Series.map.html.

"Preprocessing data". *Scikit Learn*, n.d., https://scikit-learn.org/stable/modules/preprocessing.html.

"sklearn.compose.ColumnTransformer". *Scikit Learn,* n.d., https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html.

"sklearn.ensemble.AdaBoostClassifier". *Scikit Learn*, n.d., https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html.

"sklearn.ensemble.RandomForestClassifier". *Scikit Learn*, n.d., https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

"sklearn.metrics.accuracy_score". *Scikit Learn*, n.d., https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html.

"sklearn.metrics.classification_report". *Scikit Learn*, n.d., https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html.

"sklearn.model_selection.GridSearchCV". *Scikit Learn*, n.d., https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

"sklearn.preprocessing.OneHotEncoder". *Scikit Learn*, n.d., https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html.

"sklearn.preprocessing.StandardScaler". *Scikit Learn*, n.d., https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

"sklearn.tree.DecisionTreeClassifier". *Scikit Learn*, n.d., https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

"sklearn.svm.LinearSVC". *Scikit Learn*, n.d., https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC.

"XGBoost Parameters". *XGBoost*, n.d., https://xgboost.readthedocs.io/en/stable/parameter.html.