Predicting positions at finish for derbies via multiclass classification

Name: Seung Woo Choi

Introduction

- **What is a derby?** A derby is a type of horse race in which jockeys (players) and 3-year-old horses compete with at most 20 horses on the field for 1.25 miles. On average, 8 horses compete in a derby. The Kentucky Derby is the most widely known horse race.
- **Motivation:** Derbies are entertaining to watch, and many viewers place bets on the Kentucky Derby.
- **Context:** I work as a business consultant for a risk management firm that provides services to clients who place bets on derbies.
- **Disclaimer:** I am not a financial advisor. I am in no way providing any financial advice to anyone in any shape or form.

Problem Statement

- **Primary research question:** Can a multiclass classification model be built to predict the position at finish for real-life horse races (derbies) with 8 classes?
- **Supporting research question:** What are the most important features when it comes to predicting the position at finish for a derby?
- **Limitations:** (1) Time series analysis will not be considered, (2) Horse welfare is important but the dataset does not contain enough relevant features to explore this topic, and (3) Cloud computing was not used due to budgetary constraints.
- **Project type:** Solving a real life data mining problem
- **Paper type:** Application

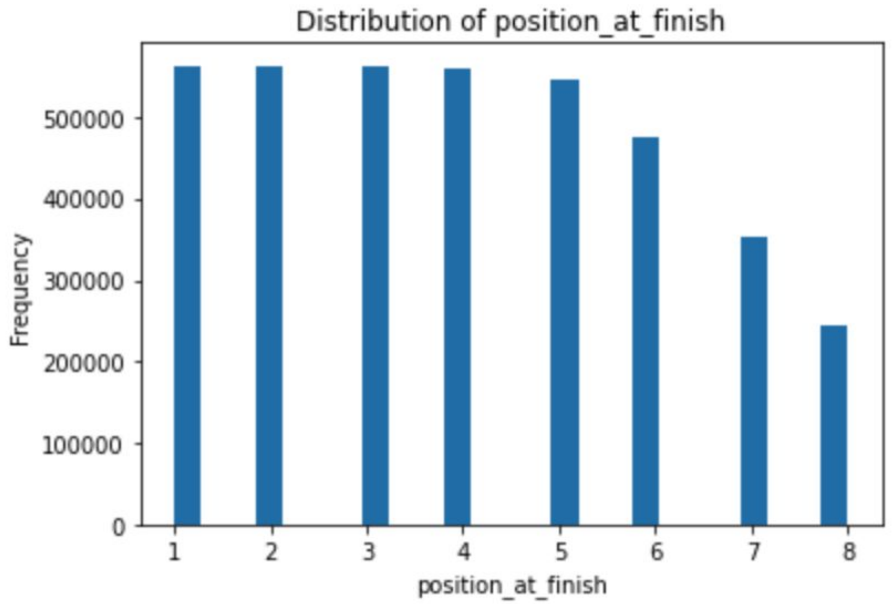
Data

- **Source:** Big Data Derby 2022 competition on Kaggle (link)
- **Provided by:** Sponsors, including NYRA, NYTHA, Equibase, The Jockey Club, The Breeder's Cup, KTA, and TOBA
- **Files:** 4 csv files - 3 csv files with horse/jockey race, racetrack race, and tracking data and 1 csv file with combined data
- **18 features:** 11 numeric features, 7 categorical features
- **Number of observations:** > 5.2M
- **Missing values:** None
- **Response variable:** position_at_finish
  - **Classes:** 1, 2, 3, 4, 5, 6, 7, 8
  - **Modifications:** Original max is 14 but filtered to 8 or less due to class imbalances and sparsity in position_at_finish > 8 (i.e. removed 7.6% of the original data)

Data (cont.)

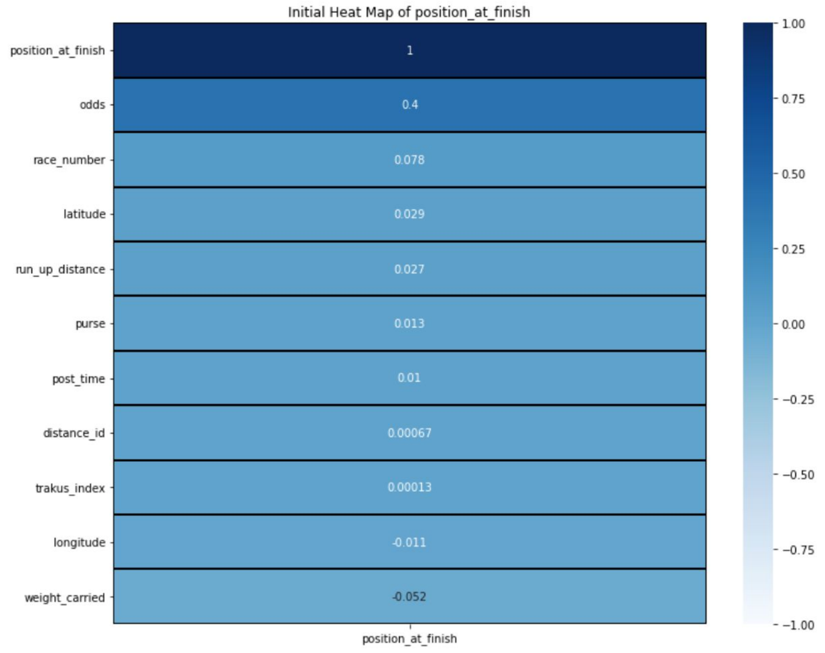
Feature	Description
odds	Odds to 1, odds of winning the race
jockey	Name of the jockey on the horse
purse	Purse of the race (in USD)
run_up_distance	Distance from the gate to the start of the race (in feet)
post_time	Time the race began (as character)

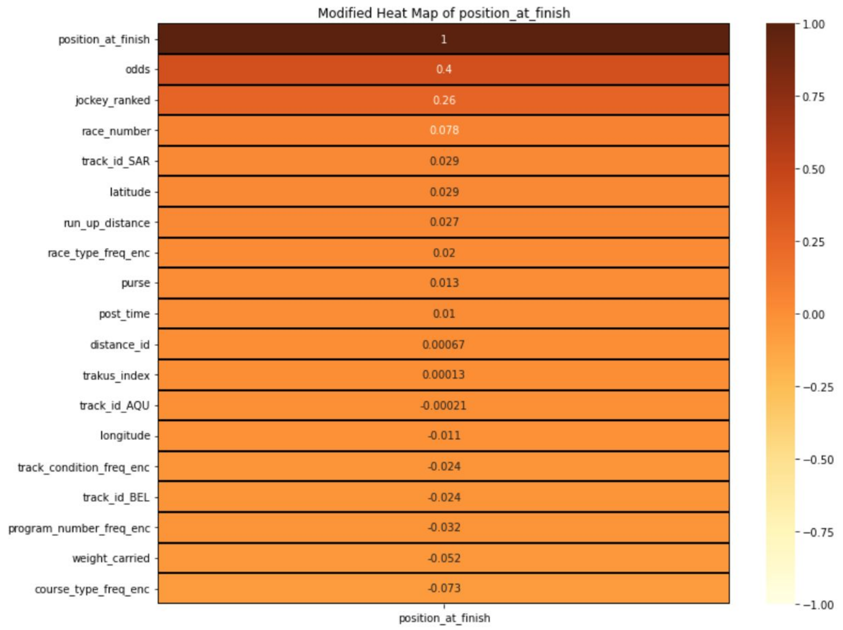
EDA on Training Data



- Random 80:20 train/test split
- Summary statistics for position_at_finish:
    - Mean: 4.04
    - Standard Deviation: 2.12
    - Range: [1, 8]
- Class proportions:
    - 1-5: 14.1% to 14.6%
    - 6: 12.3%
    - 7: 9.1%
    - 8: 6.3%

EDA on Training Data (cont.)





Feature Engineering

- Simple data type and unit conversions for **race_number**, **purse**, and **odds**
- Dropped **race_date** - time data not needed for analysis
- Dummy encoded **track_id** into 3 columns (i.e. AQU, BEL, and SAR)
- Frequency encoded and normalized **course_type** and **track_condition**
- Frequency encoded and normalized **race_type** and **program_number** with “Other” value for infrequent values
- Rank encoded **jockey** by mean position_at_finish
- 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

Model Building

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.

Evaluation Metric

Evaluation Metric	Justification
Weighted average one-vs-rest ROC AUC score	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.

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

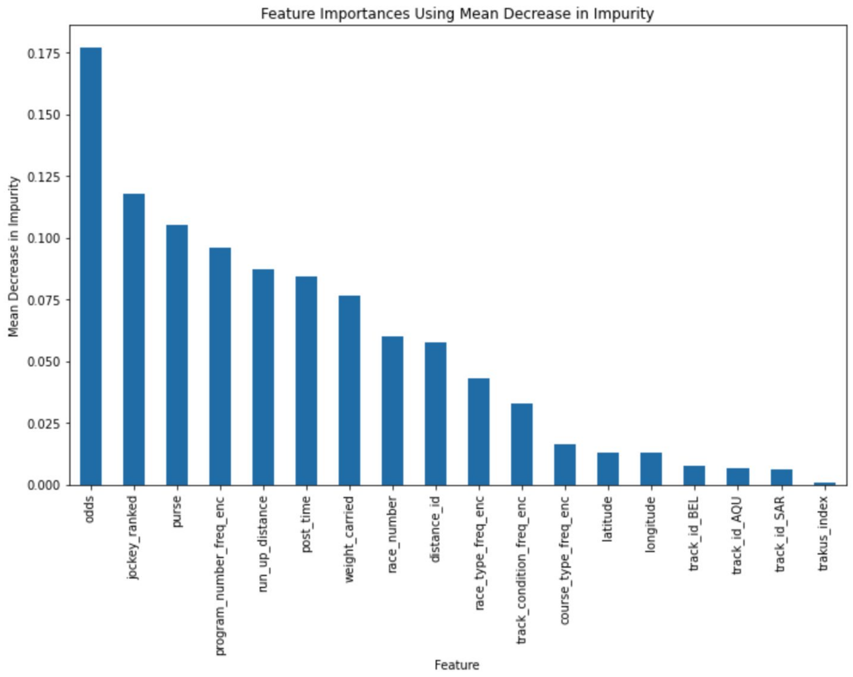
Results (cont.)

Model: random forest classifier

Metric: mean decrease in impurity (aka Gini importance) - higher equates to more importance

Important features:

1. odds
2. jockey_ranked
3. purse
4. program_number_freq_enc
5. run_up_distance



Conclusion

- **Model with best performance on evaluation metric:** Random forest classifier with 1.00
- **Model with worst performance on evaluation metric:** AdaBoost classifier with 0.64
- **Model with shortest computation time:** AdaBoost classifier with 3.77 mins
- **Model with longest computation time:** XGBoost classifier with 38.15 mins
- **Optimal model:** Random forest classifier (Score: 1.00, Computation time: 17.92 mins)
- **Worst model:** AdaBoost classifier (Score: 0.64, Computation time: 3.77 mins)
- **Most important features (based on optimal model):** (1) odds, (2) jockey_ranked, (3) purse, (4) program_number_freq_enc, and (5) run_up_distance
- **Conclusion:** The random forest 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.

Challenges

1. 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 Jupyter Notebook using Python.
2. 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 to ensure that the models could be fitted.
3. 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 instead.
4. Feature engineering required lots of nuance since 7 features were categorical variables and each had varying degrees of cardinality.

References

- https://www.kentuckyderby.com/history/the-race#:~:text=Try%20watching%20this%20video%20on,horses%20race%20against%20one%20another.
- https://www.kaggle.com/competitions/big-data-derby-2022/data
- https://stackoverflow.com/questions/24147278/how-do-i-create-test-and-train-samples-from-one-dataframe-with-pandas
- https://stackoverflow.com/questions/50558458/pandas-get-frequency-of-item-occurrences-in-a-column-as-percentage
- https://machinelearningmastery.com/multinomial-logistic-regression-with-python/
- https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- https://scikit-learn.org/stable/auto_examples/compose/plot_column_transformer_mixed_types.html
- https://towardsdatascience.com/feature-encoding-techniques-in-machine-learning-with-python-implementation-dbf933e64aa
- https://medium.com/analytics-vidhya/different-type-of-feature-engineering-encoding-techniques-for-categorical-variable-encoding-214363a016fb
- https://towardsdatascience.com/dealing-with-features-that-have-high-cardinality-1c9212d7ff1b#:~:text=What%20is%20high%20cardinality%3F,many%20of%20these%20unique%20values.
- https://pandas.pydata.org/docs/reference/api/pandas.Series.map.html
- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.rank.html
- https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
- https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
- https://scikit-learn.org/stable/modules/multiclass.html
- https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
- https://towardsdatascience.com/comprehensive-guide-on-multiclass-classification-metrics-af94cfb83fbd
- https://stackoverflow.com/questions/74414355/equivalent-for-r-dplyrs-glimpse-function-in-python-for-panda-dataframes