Team Oracle Write Up

By: Matthew Love, Suraj Rajendran, Prathic Sundararajan

Who are you (mini-bio) and what do you do professionally? If you are on a team, please complete this block for each member of the team.

Suraj Rajendran is a dedicated PhD student at Weill Cornell, specializing in computational biology. He has a keen interest in the application of artificial intelligence in various fields of healthcare, including genomics and trial emulation. As an aspiring researcher, Suraj is committed to exploring the potential of AI-driven solutions to revolutionize the healthcare industry.

Matthew Love is an experienced software engineer at Qualcomm, where he focuses on embedded systems. His professional interests lie in edge deployment, and he is always looking to apply his expertise in developing cutting-edge technologies to optimize system performance. Matthew's background in software engineering, combined with his passion for innovation, makes him a valuable contributor to the team.

Prathic Sundararajan is a proficient software engineer at Becton Dickinson, with a broad range of interests and experience working at the intersection of technology and various other domains, such as space, defense, and healthcare. His unique perspective and interdisciplinary knowledge enable him to effectively address complex challenges and design innovative solutions to real-world problems.

What motivated you to compete in this challenge?

Our motivation to compete in this challenge stems from a deep-rooted interest in aviation, ATC communications, and related technologies. The opportunity to work with real aviation data and apply our analytical skills to address complex air traffic management issues has been a driving force for our participation. Moreover, we recognize the potential impact of accurately predicting pushback times on overall air traffic efficiency and resource allocation, further fueling our enthusiasm for the challenge.

Additionally, the prospect of collaborating with brilliant minds at NASA during Phase II of the competition is incredibly exciting. We believe that working alongside experts in the field will not only be enjoyable but also provide valuable insights and a chance to learn from their experience. This unique opportunity to contribute to a project of such magnitude and relevance, combined with our personal interests in aviation and passion for problem-solving, motivated us to participate in this challenge.

High level summary of your approach: what did you do and why?

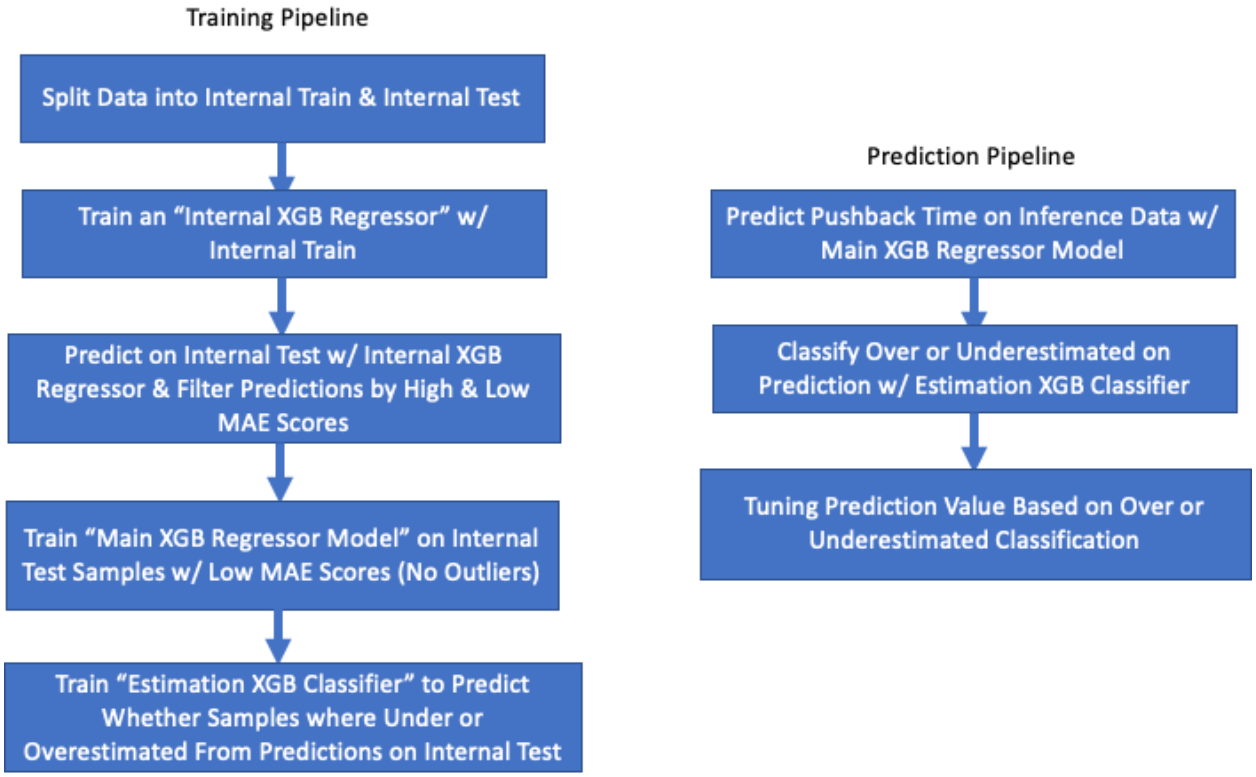
Our approach to predicting pushback times was primarily driven by the choice of tree-based models, as research indicates their efficacy in handling tabular data. To begin with, we examined important features across all datasets and assessed their contribution to trained models. Ultimately, the most significant features were derived from the estimated time of departure (most recent timepoint, average across the last 30 hours, and standard deviation across the last 30 hours), airline code, prediction time, and taxi time. We calculated taxi time using standtimes and arrivals, but this feature did not yield the expected improvements in our model.

To accommodate the unique characteristics of each airport, we trained separate XGBoost models for every airport. These models were trained on data points that were not considered outliers. To identify

outliers, we employed an internal XGBRegressor to predict minutes until pushback on a subset of data and calculated the error for each sample within the remaining data. We considered samples with a 'large error' (absolute error greater than 20 or 30, depending on the airport) as outliers. Non-outlier samples were used to train another XGBRegressor, which we refer to as the 'main model.'

During the evaluation, we noticed that our model tended to overestimate or underestimate certain samples significantly. To address this issue, we trained an XGB classifier, the 'estimation classifier,' to predict whether a sample's prediction would be overestimated or underestimated. This model had a modest performance, with an AUC of 0.64.

Our complete pipeline for new samples consists of the following steps: extract necessary features for the sample, input it into the 'main model' corresponding to the originating airport, and predict whether the minutes_until_pushback is overestimated or underestimated using our 'estimation classifier.' Based on the classification, we either subtract or add the median of all over/underestimated values, which were determined and saved during the training of the 'estimation classifier.' This approach aims to refine our predictions and minimize the impact of over- or underestimation on the model's overall performance.



How did knowing that you might need to federate this model change your approach to this challenge problem?

Awareness of the potential need to federate our model influenced our approach to this challenge. The features that provided the most significant improvements, specifically the estimated time of departure

(ETD) based features, do not require federation according to the competition rules. Similarly, our taxi time feature, derived from stand times, does not necessitate federation. The only feature in our model requiring federation is the airline code.

In light of this, we propose a two-stage approach to address the federation requirement. First, we suggest retraining airport-specific models, as we have done in Phase 1, using only features that do not need to be federated. These models will be provided to all airlines. Next, we plan to train a deep network, such as an Artificial Neural Network (ANN), leveraging features that require federated training. This federated model can be further personalized for each airline using advanced techniques like multi-task learning or transfer learning.

As a result, each airline will receive two predictions for a sample: one from the XGBoost pipeline we developed in Phase 1 and another from the federated deep model. We plan to experiment with several methods to utilize both outputs, ensuring a comprehensive and accurate prediction while addressing the federation constraints posed by the challenge. This approach allows us to balance the benefits of our current feature set while accommodating the need for a federated model.

Do you have any useful charts, graphs, or visualizations from the process?

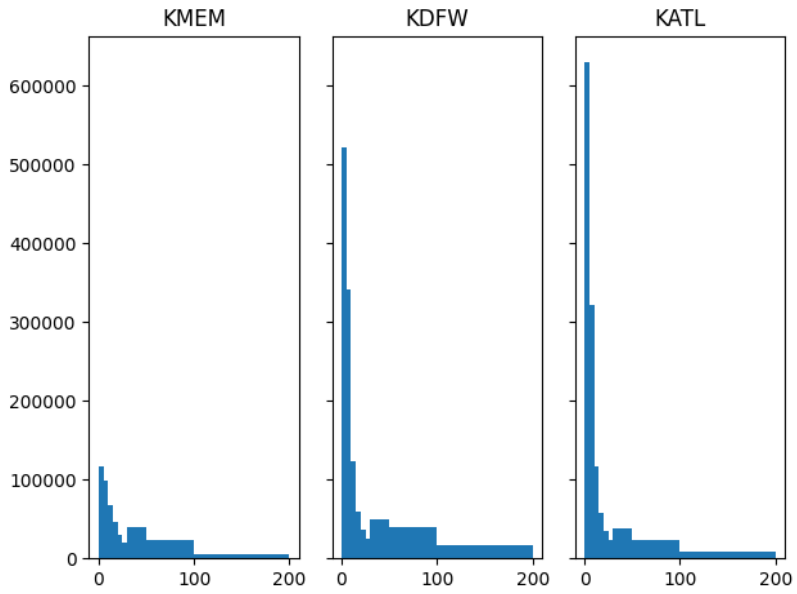


Figure 1. Histogram of mean absolute errors (X-axis) of internal regressor, with frequency of samples on y-axis

In our analysis, we have created histograms of the mean absolute error (MAE) for different airports. These histograms allow us to visualize the distribution of the MAE for each airport, revealing how well our model performs in each case. We found that some airports, such as KMEM, have a higher density of instances with large MAEs. This observation led us to the conclusion that we should filter out samples during training that may result in abnormally large MAEs in order to preserve the model's performance on more typical samples. The histograms serve as a useful tool for identifying airports where our model struggles, which in turn helps us understand the model's limitations and guides our efforts to improve its

overall performance. More specifically, this led to us attempting that internal boosting strategy which we found helped our performance."

Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

Impactful Part #1: ETD Feature

```
def extract_etd_curr_airport(curr_load_dir, sav_dir, curr_airport, bool_submission_prep):
    """
    Extracts estimated time of departure (ETD) information for a given airport and
    saves the results in a CSV file.

    Args:
        curr_load_dir (str): The directory containing the data files to be loaded.
        sav_dir (str): The directory where the resulting CSV file will be saved.
        curr_airport (str): The airport code for which the ETD information is to be extracted.
        bool_submission_prep (bool): A flag indicating if the function is used for submission preparation.

    Returns:
        None
    """
```

This code section is a function called extract_etd_curr_airport that extracts estimated time of departure (ETD) information for a given airport and saves the results in a CSV file. When we conducted feature importance, these features that were ETD based were very useful. This could be because the systems already used for calculating ETD are more or less accurate in most cases. This function is one of the most impactful parts of the code for the following reasons:

Data Extraction and Filtering:

1. The function efficiently filters and extracts relevant ETD data for a specific airport. It does so by loading data from the given directory, and then filtering it based on the airport code. This focused approach ensures that only the necessary data is processed, reducing computational time and memory usage.

Feature Engineering:

2. This function performs feature engineering on the data, specifically for unique timestamps. By sorting the data based on 'gufi' (unique flight identifier) and 'timestamp', it allows for easier access and manipulation of the data. Additionally, it drops duplicates and calculates the minutes until departure from the timepoint and timestamp. These engineered features provide valuable information for the model, enabling it to make more accurate predictions.

Scalability and Progress Tracking:

3. The function is designed to handle large datasets and provides progress tracking through percentage completion reports. This allows users to monitor the function's progress, making it easier to handle large-scale data processing tasks.

Impactful Part #2: Internal Regressor Model

```
print(f'Only {sum(bool_mask)} out of {len(df_internal_test)} were above the threshold of {mae_thresh} and were removed from training.')
df_internal_test.loc[bool_mask,'ml_model_used'] = 1
df_internal_test_lower = df_internal_test[df_internal_test.ml_model_used == 0].copy()
df_internal_test_higher = df_internal_test[df_internal_test.ml_model_used == 1].copy()

X_lower = df_internal_test_lower[list(set(feature_cols))]
y_lower = df_internal_test_lower[label_col]

regressor_lower = xgb.XGBRegressor()
print(f'Trained on {len(X_lower)} many datapoints.')
regressor_lower.fit(X_lower, y_lower)
##########

model_file_name = f"{model_save_dir}{airport}_{model_type}_NOOUTLIER_{comment}.pkl"
pickle.dump(regressor_lower, open(model_file_name, "wb"))
```

This code section performs several important tasks related to model training and evaluation, making it another impactful part of the code. The final regressor is trained on non-outlier data as determined by our internal regressor, allowing increased fidelity for the model. The main tasks performed are:

Data Splitting:

1. The code uses GroupShuffleSplit to split the data into training and testing sets, ensuring that the 'gufi' (unique flight identifier) groups are not mixed between the two sets. This ensures that the model is evaluated on unseen data, providing a more accurate estimate of its performance.

Internal Regressor Model Training and Evaluation:

2. An XGBoost Regressor (internal_regressor) is trained on the training data and used to make predictions on the test data. The Mean Absolute Error (MAE) and Mean Squared Error (MSE) are calculated to evaluate the model's performance. This evaluation is crucial for understanding the model's ability to generalize and make accurate predictions.

Data Filtering Based on MAE Threshold:

3. The code filters the test data based on a threshold for MAE, separating the lower and higher error instances. This filtering helps create a lower model (regressor_lower) that is trained specifically on the filtered data, helping to improve the model's overall performance.

Impactful Part #3: Estimation Classifier

```
median_underestimation = estimate_classifier_params['median_underestimation']
median_overestimation = estimate_classifier_params['median_overestimation']
X_test_lower = df_predict.copy(deep=True)
X_test_lower['pred_minutes_until_pushback'] = y_pred_lower
y_prob_estimate = estimate_classifier.predict_proba(X_test_lower[all_trained_features + ['pred_minutes_until_pushback']])[:, 1]
X_test_lower = X_test_lower.reset_index(drop=True)
X_test_lower['final_pred_minutes_until_pushback'] = np.where(y_prob_estimate > 0.5, X_test_lower['pred_minutes_until_pushback'] + med
X_test_lower['final_pred_minutes_until_pushback'] = np.where(y_prob_estimate < 0.5, X_test_lower['final_pred_minutes_until_pushback']
y_pred = X_test_lower['final_pred_minutes_until_pushback'].values
```

This code section focuses on training an over and under estimation classifier, which is another impactful part of the code. The estimator can be used to better fine tune final minutes until pushback estimations. The usefulness of it is limited because of the estimator's accuracy but it still helps a certain amount. The main tasks performed are:

Classifier Label Generation:

1. The code calculates the prediction errors by comparing the lower model's predictions (regressor_lower) with the actual minutes until pushback. It then generates binary labels for overestimation (0) and underestimation (1) of the model's predictions. This information helps to identify and quantify the model's estimation tendencies.

Estimation Classifier Training:

2. An XGBoost Classifier (estimate_classifier) is trained on the data, with features including the original feature columns and the predicted minutes until pushback. The classifier learns to predict whether a given instance will be an overestimation or underestimation based on these features.

Overestimation and Underestimation Quantification:

3. The code calculates the probabilities for each instance to be an overestimation or underestimation using the estimation classifier. It then filters the instances based on the classifier's upper and lower probability thresholds, finding the median overestimation and underestimation values. During inference time, we add or subtract these median values to better estimate the pushback values. These values help quantify the model's tendencies and provide insight into potential areas for improvement.

Please provide the machine specs and time you used to run your model.
CPU (model): Apple M1 Max
GPU (model or N/A): N/A
Memory (GB): 64 GBs
OS: MacOS
Train duration: 70 mins
Inference duration: ~20 mins/2 million timepoints

Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?
When using our model, there are a few aspects to be mindful of to ensure smooth operation and accurate predictions. One such aspect is that some parts of our feature extraction process rely on pqdm, a package that enables multithreading. While this can offer performance improvements, multithreading can occasionally lead to issues or instability. To mitigate this risk, we have also provided a base version of our feature extraction code that does not utilize multithreading, which has a lower likelihood of encountering failures.

As you work with our model, we recommend monitoring memory usage and ensuring that your system can handle the computational requirements. Additionally, it is crucial to keep an eye on any potential numerical stability issues that may arise during the model's operation. By being vigilant of these concerns and using the base version of our feature extraction code when necessary, you can achieve optimal results from our model while minimizing the likelihood of encountering issues.

Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?
During the initial stages of our project, we utilized additional resources to gain a deeper understanding of the problem domain and the data provided. To supplement our analysis, we explored online flight trackers to familiarize ourselves with real-world flight operations and air traffic management. This helped us gain insights into the underlying patterns and relationships in the data that might not be readily apparent from the dataset alone.

Furthermore, we conducted a literature search to review state-of-the-art methods in predicting taxi time and minutes to pushback. This research allowed us to identify relevant techniques, best practices, and potential pitfalls in similar predictive modeling tasks. By incorporating these findings into our work, we were able to make informed decisions about our modeling approach and feature selection, ultimately leading to a more robust and accurate solution.

Although these resources were not directly included in our code submission, they played a vital role in shaping our understanding of the problem and informing our approach to developing an effective model for predicting pushback times.

How did you evaluate performance of the model other than the provided metric, if at all?
Apart from the provided metric, we conducted an experiment to evaluate our model's performance under different data conditions. We intentionally reduced the amount of training data to only 30% of the original dataset to observe the impact on our model's performance. This experiment aimed to assess the model's robustness, stability, and generalizability when confronted with limited data availability.

By analyzing the performance of our model under such constraints, we were able to gain insights into the model's potential weaknesses and strengths. This additional evaluation technique helped us better understand our model's behavior and potential limitations, informing possible improvements and adjustments to enhance its performance and adaptability in various data scenarios.

What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?
Throughout the development process, we explored several approaches that did not make it into the final workflow due to various reasons. Here is a quick overview of these experiments:

1. Additional features: We computed several features based on the literature, such as traffic (based on arrivals and departures at airports), weather conditions, and plane initialization. However, these features did not lead to a significant improvement in the model's performance and were subsequently excluded from the final feature set.
2. Recurrent Neural Network (RNN) with LSTM layers: We experimented with an RNN incorporating LSTM layers to leverage the temporal aspects of the data. While this model showed promising results in the open arena, its performance did not carry over to the pre-screened submission, indicating potential generalization issues.

3. Convolutional Neural Network with LSTM (CNN-LSTM): Another approach we tested was a hybrid model combining CNN and LSTM layers. Despite its potential to capture both spatial and temporal patterns, the CNN-LSTM model proved to be computationally expensive and did not provide significant performance improvements compared to other models.

These experiments, although not included in the final workflow, contributed to our understanding of the problem and allowed us to refine our approach by identifying the most effective techniques and features for predicting pushback times.