Improving Arrival Throughput Predictions for NASA Airports

Boston University

Chengkai Yang and Jingbo Wang

December 13th, 2024

1. Introduction

The objective of this project is to develop a predictive modeling approach that accurately forecasts the number of arriving flights (throughput) for NASA airports for the next three hours, with a temporal resolution of 15 minutes. The primary evaluation metric, as specified in the task, is based on the Root Mean Squared Error (RMSE), transformed via an exponential function:

$$\text{score} = \exp(-\text{RMSE}/10)$$

A perfect prediction (RMSE = 0) corresponds to a score of 1.0, and worse predictions result in a score closer to 0.0.

This modeling challenge involves forecasting multiple future intervals simultaneously (multi-target regression) and leverages historical flight data, weather forecasts, and airport operational factors. The ultimate goal is to produce robust, real-time predictions that generalize well to unseen data. Due to storage and computing units limitations, we only used FUSER as the primary dataset for training and testing models.

Competition Link: [The NASA Airport Throughput Prediction Challenge | bitgrit]

2. Methods

2.1 Data Preprocessing and Feature Engineering

FUSER Data Cleaning and Preprocessing:

The primary dataset, known as FUSER data, integrates multiple sources: flight-level operational data, weather forecasts (LAMP), runway configurations (D-ATIS), and TFM/TBFM predictions. Each data type is stored in separate CSV files, organized by date and airport. Before modeling, the data cleaning and preprocessing is done by combining Multiple Data Sources by Airport and Date_time:

For each (airport, date_time) combination, we loaded the corresponding runways, ETD, TBFM, TFM_track, configs (D-ATIS), MFS, and LAMP datasets. We ensured that all these disparate datasets were merged into a single DataFrame that contained:

1. A reference time (`ref_time`) derived from `configs` data (e.g., `start_time` from

D-ATIS messages).

2. Interval-based future arrival/departure counts (`interval_n_Arrival`, `interval_n_ETD`, `interval_n_TFM`, `interval_n_TBFM`), each representing a 15-minute window from the reference time.
3. Weather conditions aligned to these intervals from LAMP forecasts.
4. Additional runway configuration features (e.g., number of arrival and departure runways available).

Reference Time Alignment:
In this approach, the temporal alignment of data relies on the already standardized intervals defined in the `submission_format.csv` file rather than explicitly flooring reference times to 15-minute increments in the code. The `submission_format.csv` encodes future prediction times as discrete intervals (e.g., `interval_1` for the first 15-minute window, `interval_2` for the next 15 minutes, etc.). These intervals are inherently aligned to uniform 15-minute blocks, allowing all data sources—flight, weather, and runway usage—to be mapped consistently to these standardized time windows without an additional flooring step.

Interval Assignment and Data Cleaning:

- **Flight Data:** With a consistent reference schedule provided by the submission format, each flight's arrival or departure time is matched to the nearest predefined 15-minute interval. If a flight's timestamp falls within the window defined by `interval_1` (e.g., 00:30–00:45) or `interval_2` (00:45–01:00), it is associated accordingly. Records lacking essential information (such as a valid timestamp or airport_id) are dropped or imputed to maintain data quality.
- **Weather Data:** Weather forecasts, originally sampled at varying times, are similarly aligned to these pre-established intervals. If the original data is not perfectly synchronous with the intervals, we rely on the uniform spacing defined by the submission format to identify which 15-minute window the forecast should fall into. Missing or anomalous weather values are replaced with NaN and then handled through interpolation or imputation. Categorical variables, such as cloud conditions, are converted into numeric representations to facilitate model training.
- **Runway Usage:** Information about the number of active runways is integrated into the same 15-minute structure. Even if runway data were recorded at slightly irregular times, the standardized intervals provide a framework to assign each runway configuration state to the correct window. Where data gaps occur, runway usage is either interpolated or assigned a default value so that every interval remains consistent and informative.

By leveraging the interval definitions in the submission format—rather than performing an explicit flooring operation—this method ensures that all data (flights, weather, and runways) is

mapped onto a regular temporal grid. This uniform time structure simplifies training and prediction, enabling the model to learn patterns from and predict future arrivals at a stable, well-defined set of intervals.

Feature Engineering Rationale:
To capture temporal patterns and improve predictive power, we engineered features that encode short-term historical trends and time-dependent cycles:

1. **Time-Based Features:**
   We extracted `hour`, `day_of_week`, and `month` from `ref_time`. Because these are cyclical (e.g., hour 23 is adjacent to hour 0), we applied sine/cosine transforms (`hour_sin`, `hour_cos`, `dow_sin`, `dow_cos`) to preserve cyclical structure. This allowed the model to better understand daily and weekly patterns in arrivals.
2. **Lag Features:**
   Since arrivals at the next intervals are often influenced by recent past activity, we introduced lag features of `interval_1_Arrival` (e.g., `lag_1_arr`, `lag_2_arr`, `lag_3_arr`, `lag_4_arr`). These represent arrival counts from previous 15-minute intervals (up to one hour back), providing the model with short-term historical context.
3. **Rolling Statistics:**
   To capture local trends and volatility, we computed a 1-hour rolling mean and standard deviation of arrivals (`roll_1h_mean_arr`, `roll_1h_std_arr`). By smoothing recent history, these features help the model discern stable trends versus transient spikes or dips.

Incremental Experiments with Complexity:
Initially, we experimented with more extensive feature engineering:

- Longer lag windows (2 hours or more) and multiple rolling intervals (30 min, 2 hours).
- Adding lag and rolling features for weather and runway usage.

However, these complex additions often led to overfitting and did not improve validation RMSE. We concluded that simpler, more targeted lag and rolling features for arrivals provided a better balance of predictive performance and generalization.

Assumptions:

- The provided datasets are sufficiently representative of operational patterns and contain no major regime shifts not captured in training.
- Weather forecasts are reasonably accurate at short horizons.
- The chosen 15-minute intervals and up to 1-hour lag/rolling windows capture the

essential short-term dynamics needed for prediction.

2.2 Model Selection and Tuning

We tested several models before selecting CatBoost as the final model:

1. Random Forest Regression
    - Strengths: Robust to missing data and easy interpretability.
    - Weaknesses: Struggled with multi-target regression and produced suboptimal validation RMSE.
2. XGBoost
    - Explored with hyperparameter tuning (learning rate, max_depth, colsample_bytree).
    - Results: Competitive performance but sensitive to data preprocessing. Required extensive manual tuning for stability.
3. LightGBM
    - Implemented advanced hyperparameter tuning via RandomizedSearchCV.
    - Introduced feature fraction, bagging fraction, and early stopping for optimization.
    - Results: Strong RMSE but overfitting occurred in ensembles and complex lags.

Final Choice of Model:
We evaluated several gradient boosting frameworks (XGBoost, LightGBM) and random forests but ultimately selected CatBoostRegressor. Reasons include:

- Multi-Target Regression Support: CatBoost can naturally handle multiple target intervals simultaneously, simplifying pipeline complexity.
- Robustness and Categorical Handling: CatBoost's native handling of categorical variables and balanced defaults reduced the need for extensive custom preprocessing.
- Stable Training Dynamics: CatBoost often produces strong results with minimal tuning and is less sensitive to parameter instability compared to some other gradient boosting methods.

Baseline Hyperparameters:
Our initial CatBoost configuration included:

- iterations=2000: Enough rounds for thorough learning without excessively long training times.
- learning_rate=0.03: A relatively moderate rate, allowing the model to converge steadily.
- depth=7: Providing enough capacity to capture complex relationships without extreme overfitting.
- od_wait=200: Early stopping after 200 iterations of no improvement on the validation set, preventing unnecessary overfitting and saving computation time.

- loss_function='MultiRMSE': Aligning with the multi-target regression objective.

Refinements and Rationale:

1. Learning Rate:
   We experimented with lowering the learning rate from 0.03 to 0.02 to allow more gradual improvements. A slightly smaller learning rate helped the model fine-tune its predictions and sometimes led to a modestly better validation RMSE. It also offered more stability during the late stages of training.
2. l2_leaf_reg (Regularization):
   Increasing `l2_leaf_reg` from the default (3) to 5 introduced stronger L2 regularization. This was done after observing that the model showed signs of overfitting with more complex features. With simpler features, higher regularization helped ensure the model did not overfit minor fluctuations in the training set.
3. Iterations and Early Stopping:
   We tested allowing more iterations (e.g., 3000+), but found diminishing returns. The model generally reached a plateau around 2000 iterations with the given learning rate and `od_wait`. Adjusting `od_wait` to smaller values (like 100) was also tried, but sometimes caused premature stopping. Ultimately, leaving `od_wait=200` allowed the model time to discover late improvements without excessive training time.
4. Subsampling and Feature Sampling:
   We considered introducing subsampling (`subsample`) and feature sampling (`rsm`), alongside `bootstrap_type='Bernoulli'`, to reduce overfitting. However, in practice, these did not outperform the simpler configuration without subsampling. We reverted to the simpler approach after confirming no improvement in validation scores.

Summary of Tuning Process:
Our tuning strategy was incremental and guided by validation RMSE:

- Start from a straightforward setup (depth=7, lr=0.03, iterations=2000).
- Lower learning_rate to 0.02 and increase l2_leaf_reg to 5 to reduce overfitting.
- Test more iterations and different `od_wait` values, revert if no improvement.
- Attempt subsampling and additional complexity only if initial tweaks do not improve results.

This step-by-step approach allowed us to isolate which modifications truly enhanced performance and ensured that improvements were stable.

2.3 Obstacles and Solutions

Overfitting with Complex Features:
Attempting to include extensive lag/rolling features and weather-based complexity often led to the model memorizing training patterns rather than learning general principles. The validation RMSE would stagnate or worsen.

Solution:
We reverted to a simpler feature set focusing on recent arrivals and a single rolling window. This stripped-down approach reduced noise and delivered the best validation performance.

Difficulty in Parameter Sensitivity:
Small changes in `learning_rate` or `l2_leaf_reg` did not always yield immediate, large performance gains. Instead, improvements were subtle, requiring careful observation of validation RMSE and incremental, patient adjustments.

Solution:
We embraced a slow, iterative tuning process. By changing one parameter at a time and monitoring results, we gradually found a sweet spot in hyperparameters. Early stopping (`od_wait`) helped confirm when no further improvement was likely, preventing wasted computation.

Complex Integration of Multiple Data Sources:
Combining flight, weather, and runway data from separate files required careful alignment, interpolation, and error handling.

Solution:
We established a consistent time reference and used robust methods (fillna, interpolation) to handle missing values. By ensuring a clean and coherent dataset before modeling, we minimized data-related issues.

2.4. Results

Model	Submission Score	Notes
Random Forest	**0.493**	**Baseline performance.**
CatBoost	**0.504**	**Best-performing model**

LightGBM	**0.510**	**Effective but prone to overfitting.**
CatBoost Tuned	0.549	**Best-performing model tuned with rolling and lagging features.**

4. Project GitHub Link

[Improving Arrival Throughput Predictions for NASA Airports]

5. Final Score

- **Final Highest Public Score:** 0.549
- **Final Public Rank:** 36

6. Conclusion

Through systematic data cleaning, careful alignment of various data sources, and judicious feature engineering, we prepared a robust dataset that captured key predictive signals for future arrival throughput. Aligning `ref_time` to standardized 15-minute marks and ensuring all flight, weather, and runway data were consistently mapped to interval windows was crucial. The final model — a CatBoostRegressor trained on short-term lag features, rolling statistics, and essential weather and runway features — delivered stable results. Despite attempts at more complex feature engineering, simpler approaches ultimately generalized better.

This process underscores the importance of thorough data preparation, including consistent time alignment and careful handling of missing values, as well as the value of iterative experimentation with features and hyperparameters. The final predictions, combined with time-based evaluation and the exponential scoring metric, support well-informed operational decisions for NASA airports.