**Introduction**

This project focuses on estimating the **Take-Off Weight (TOW)** of aircraft using advanced machine learning techniques. Two primary models were developed, with one initial not used in later runs:

1. ***First Model:*** *Deprecated during development*
2. **Second Model**: Utilizes a comprehensive set of flight and weather-related features.
3. **Third Model**: Uses the initial challenge set, including economic indicators. Used for all values where variables for the second model could not be obtained from the trajectory files.

Both models employ **XGBoost** for regression tasks and leverage **Bayesian Optimization** for hyperparameter tuning. The models are trained on historical flight data to predict TOW with high accuracy.

**Prerequisites**

**Libraries and Packages**

Ensure that the following R packages are installed:

* tidyverse: Data manipulation and visualization.
* caret: Data preprocessing and model training utilities.
* ParBayesianOptimization: Performing Bayesian Optimization.
* doParallel and foreach: Parallel processing capabilities.
* Matrix: Handling sparse and dense matrices efficiently.
* lubridate: Date and time manipulation.
* minioclient: Interacting with MinIO object storage.
* xgboost: GPU-enabled version for accelerated computations.

**XGBoost Installation**

A GPU-enabled version of XGBoost is required for faster computation. Install it from the provided URL, which contains the pre-built GPU version compatible with your system.

**System Requirements**

* **GPU Support**: NVIDIA GPU with compute capability compatible with XGBoost GPU version.
* **Parallel Processing**: Multiple CPU cores to leverage parallel processing during model training and optimization.

**Data Preparation**

**Working Directory**

Set your working directory to the location where the data files are stored.

**Data Loading**

**Second Model Data**

* **Dataset**: challenge\_set\_v6.csv
* **Date Conversion**: Convert the date column to numeric format to facilitate modeling.

**Third Model Data**

* **Dataset**: Subset of challenge\_set\_v6.csv with additional economic indicators like MSCI.Adj.Close and Oil.Adj.Close.
* **Date Conversion**: Similar date conversion as in the second model.

**Missing Values Handling**

* **Target Variable (tow)**: Remove any records where tow is missing, as these cannot be used for training the model.

**Feature Engineering**

**Feature Selection**

**Second Model Features**

* **Categorical Features**:
  + aircraft\_type: Type of the aircraft.
  + country\_code\_adep: Country code of the departure airport.
  + country\_code\_ades: Country code of the arrival airport.
  + airline: Airline operating the flight.
  + callsign: Callsign of the flight.
  + adep: Departure airport code.
  + ades: Arrival airport code.
* **Numerical Features**:
  + flight\_duration: Duration of the flight in minutes.
  + flown\_distance: Distance flown by the aircraft in nm.
  + ground\_speed\_at\_lift\_off: Ground speed at the moment of lift-off.
  + ground\_speed\_delta: Difference in ground speed during take-off.
  + taxiout\_time: Time spent taxiing out before take-off.
  + time\_to\_lift\_off: Time taken from start of roll to lift-off.
  + avg\_altitude: Average altitude during the flight.
  + jet\_stream\_coeff: Coefficient representing the effect of jet streams.
  + date: Numeric representation of the flight date.
  + departure\_temp: Temperature at the departure location.
  + arrival\_temp: Temperature at the arrival location.
  + u\_wind: East-West component of wind speed.
  + v\_wind: North-South component of wind speed.
  + vertical\_ascend: Vertical ascend rate.
  + vertical\_descend: Vertical descend rate.
  + humidity\_diff: Difference in humidity between departure and arrival locations.

**Third Model Features**

* **Categorical Features**:
  + Same as in the second model.
* **Numerical Features**:
  + flight\_duration
  + flown\_distance
  + taxiout\_time
  + MSCI.Adj.Close: Adjusted closing price of the MSCI index (economic indicator).
  + Oil.Adj.Close: Adjusted closing price of oil (economic indicator).
  + date

**Data Splitting**

* **Training Set**: 80% of the data.
* **Testing Set**: 20% of the data.
* **Seed**: Set a random seed (e.g., 123) for reproducibility.

**Categorical Data Processing**

* **One-Hot Encoding**: Convert categorical variables into binary indicator variables.
* **Sparse Matrix**: Use sparse matrices to efficiently handle high-dimensional data resulting from one-hot encoding.

**Numerical Data Processing**

* **Standardization**: Center and scale numerical features to have zero mean and unit variance.
* **Preprocessing Model**: Fit the preprocessing model on training data and apply it to both training and testing sets.

**Feature Matrix Construction**

* **Combine**: Merge the processed categorical and numerical matrices to form the final feature matrices (X\_train and X\_test) for training and testing.

**Model Selection and Hyperparameter Optimization**

**Choice of XGBoost Model**

**XGBoost** is selected for the following reasons:

* **Performance**: XGBoost is renowned for its efficiency and accuracy in regression tasks, often outperforming other algorithms.
* **Handling of Sparse Data**: Efficiently manages sparse input data, which is beneficial due to the one-hot encoding of categorical variables.
* **Regularization**: Provides built-in regularization parameters (lambda, alpha, gamma) to prevent overfitting.
* **Parallel Processing and GPU Support**: Supports parallel processing and GPU acceleration, significantly reducing training time for large datasets.
* **Flexibility**: Offers extensive hyperparameter options to fine-tune model performance.

**Usage of Bayesian Optimization**

**Bayesian Optimization** is used to optimize the hyperparameters of the XGBoost models.

* **Efficiency**: Bayesian Optimization is more sample-efficient than traditional methods like grid search or random search, requiring fewer iterations to find optimal hyperparameters.
* **Probabilistic Model**: Builds a probabilistic model (usually a Gaussian Process) of the objective function to make informed decisions about where to sample next.
* **Global Optimization**: Balances exploration and exploitation, efficiently searching the hyperparameter space to find the global optimum.
* **Suitability for Expensive Functions**: Ideal for optimizing functions that are expensive to evaluate, such as model training with cross-validation.

**Hyperparameter Optimization Process**

1. **Objective Function Definition**:
   * The objective function accepts hyperparameters as inputs.
   * It performs k-fold cross-validation using xgb.cv to evaluate model performance with the given hyperparameters.
   * The function returns the negative mean RMSE from cross-validation (since the optimizer maximizes the objective function).
2. **Hyperparameter Bounds**:
   * Define realistic ranges for each hyperparameter to guide the optimization process:
     + **Learning Rate (eta)**: 0.01 to 0.5.
     + **Maximum Tree Depth (max\_depth)**: 8 to 15 (must be integers).
     + **Minimum Child Weight (min\_child\_weight)**: 8 to 15.
     + **Subsample Ratio (subsample)**: 0.5 to 0.8.
     + **Column Sample by Tree (colsample\_bytree)**: 0.5 to 0.8.
     + **Regularization Parameters**:
       - gamma: 0 to 10.
       - lambda (L2 regularization): 0 to 10.
       - alpha (L1 regularization): 0 to 10.
     + **Number of Boosting Rounds (nrounds)**: 200 to 750 (must be integers).
3. **Parallel Processing Setup**:
   * Utilize available CPU cores to parallelize the optimization.
   * Create a cluster using the doParallel package and register it with foreach.
   * Export necessary variables and libraries to each worker in the cluster.
4. **Bayesian Optimization Execution**:
   * **Initialization**: Start with a predefined number of random samples (initPoints) to explore the hyperparameter space.
   * **Iterations**: Run the optimization for a specified number of iterations (iters.n).
   * **Acquisition Function**: Use an acquisition function like the Upper Confidence Bound (UCB) to decide where to sample next, balancing exploration and exploitation.
   * **Parallelization**: Execute the optimization in parallel to speed up computation.
5. **Best Hyperparameters Extraction**:
   * After optimization, extract the hyperparameters corresponding to the best (lowest) RMSE.
   * These hyperparameters are used to train the final model.

**Model Training and Evaluation**

**Data Alignment**

* **Feature Consistency**: Ensure that the training and testing datasets have identical features.
* **Handling Missing Features**: If certain features are present in the training set but absent in the testing set, add these features to the testing set with zero values to maintain alignment.

**DMatrix Creation**

* **Training DMatrix**: Create a DMatrix object for the training data, which is an optimized data structure that XGBoost uses for efficient computation.
* **Testing DMatrix**: Similarly, create a DMatrix object for the testing data.

**Final Model Training**

* **Set Parameters**: Use the best hyperparameters obtained from Bayesian Optimization.
* **Early Stopping**: Implement early stopping by monitoring the validation error and stopping training if it doesn't improve after a certain number of rounds (early\_stopping\_rounds).
* **Evaluation Metrics**: Use RMSE as the evaluation metric to measure model performance.
* **Watchlist**: Provide XGBoost with a watchlist containing the training and validation datasets to monitor performance during training.

**Feature Importance Analysis**

* **Importance Matrix**: Extract the feature importance scores from the trained model to understand the impact of each feature.
* **Visualization**: Plot the feature importance scores to visually interpret which features are most influential in predicting TOW.
* **Insights**: Use this information to potentially refine the feature set in future iterations.

**Prediction on Submission Set**

**Data Preparation**

* **Load Submission Data**: Read the submission dataset, ensuring that the date column is converted to numeric format.
* **Select Features**: Extract the same features used in the training data.
* **Preprocessing**: Apply the same one-hot encoding and standardization procedures used on the training data.
* **Feature Alignment**: Align the submission data features with the training data by adding any missing features with zero values.

**Prediction**

* **DMatrix Creation**: Convert the processed submission data into a DMatrix object.
* **Model Prediction**: Use the trained XGBoost models to predict TOW for each flight in the submission dataset.
* **Handling Multiple Models**: If predictions are made using both the second and third models, combine the results appropriately.

**Results Preparation**

* **Combine Predictions**: Merge predictions from different models if necessary, ensuring that each flight\_id has a corresponding predicted tow.
* **Remove Duplicates**: Ensure there are no duplicate flight\_id entries in the final results.
* **Prepare Submission File**: Create a DataFrame or CSV file containing the flight\_id and the predicted tow values in the required format.

**Saving Results**

* **Output File**: Save the predictions to a CSV file (e.g., team\_bold\_emu\_predictions.csv) without row names or indices.

**Submission to MinIO**

**MinIO Setup**

* **Access Credentials**: Use your provided MinIO access\_key and secret\_key for authentication.
* **Alias Configuration**:
  + Use the mc alias set command to set up an alias for the MinIO server.
  + Verify that the alias is set correctly using mc alias ls.

**File Upload**

* **Local File Path**: Specify the path to your local prediction CSV file.
* **MinIO Destination**: Define the destination bucket and object path on the MinIO server (e.g., dc24/submissions/team\_bold\_emu\_predictions.csv).
* **Upload Command**: Use the mc cp command to upload your file to the MinIO server.
* **Verification**:
  + Check the command output or return code to confirm the upload was successful.
  + If necessary, list the contents of the destination bucket to verify the presence of your file.