GoTo Data Science Take-Home Assignment: Driver Assignment Model

Dhanushkumar R

***Abstract*—This report presents the development of a driver assignment model for the GoTo Data Science Take-Home Assignment, aimed at predicting driver acceptance of booking requests in a ride-hailing platform. The pipeline, implemented in Python using `pandas`, `scikit-learn`, `lightgbm`, and `haversine`, processes booking and participant logs, engineers predictive features, trains a LightGBM model, and generates optimal driver assignments. Seven features were engineered, informed by exploratory data analysis (EDA), achieving an accuracy of 76–80%. Challenges such as import errors, timestamp inconsistencies, serialization issues, runtime warnings, and low initial accuracy were resolved through rigorous debugging and optimization. The final pipeline is efficient, scalable, and robust, producing `results.csv` with 10,000 driver assignments and `metrics.json` with evaluation metrics.**

I. Introduction

The GoTo Data Science Take-Home Assignment required building a machine learning pipeline to predict whether a driver will accept a booking request, enabling optimal driver assignments in a ride-hailing platform. The pipeline processes raw booking and participant logs, engineers features, trains a model, and generates predictions for test data. This report details the dataset, exploratory data analysis (EDA), feature engineering, feature selection rationale, model selection, feature importance, challenges faced, and solutions implemented. The pipeline was developed using Python, with a focus on efficiency, accuracy, and robustness.

II. Dataset Overview

The dataset comprises three CSV files:

- **booking_log.csv**: Contains booking details, including `order_id`, `event_timestamp`, `booking_status`, `customer_id`, `pickup_latitude`, `pickup_longitude`, etc.
- **participant_log.csv**: Records driver interactions, including `order_id`, `event_timestamp`, `participant_status` (e.g., ACCEPTED, REJECTED), `driver_id`, `driver_latitude`, `driver_longitude`, etc.
- **test_data.csv**: Test dataset for predictions, with similar columns to `participant_log.csv` but without `participant_status`.

A. Key Characteristics

- **Size**: The merged dataset (`dataset.csv`) has approximately 1.5 million rows, with 1,210,722 training rows and 302,681 testing rows after an 80/20 split.
- **Target Variable**: `target` (binary: 1 for ACCEPTED, 0 for REJECTED), derived from `participant_status`.
- **Class Distribution**: Balanced, with 54.44% negative (0) and 45.56% positive (1), reducing the need for aggressive resampling.
- **Missing Values**: Handled by filling with means or zeros (e.g., `wait_time`, `historical_completed_bookings`).
- **Timestamps**: `event_timestamp` in both logs, used for time-based features.

B. Exploratory Data Analysis (EDA) Insights

EDA was conducted using `eda.py` on `dataset.csv` to understand data patterns and inform feature engineering. Key insights include:

- **Balanced Target Distribution**: The `target` variable showed 54.44% negative (REJECTED) and 45.56% positive (ACCEPTED) cases, indicating a balanced dataset that minimized the need for oversampling or class weighting beyond `scale_pos_weight` in LightGBM.
- **Dominance of Short Trips**: The majority of bookings had short distances between driver and pickup locations (median `driver_distance` ¡ 5 km), supporting the inclusion of `driver_distance` as a feature, as proximity likely influences acceptance.
- **Response Time Correlation**: Shorter `wait_time` (time between booking and driver response) strongly correlated with higher acceptance rates, justifying `wait_time` and its non-linear transformation, `wait_time_squared`, to capture diminishing returns for longer wait times.
- **Driver Behavior Patterns**: Drivers with higher `driver_acceptance_rate` (historical acceptance proportion) were significantly more likely to accept new bookings, validating its use as a predictive feature.
- **Temporal Patterns**: Bookings during peak hours (e.g., morning and evening commutes) showed slightly higher acceptance rates, supporting the inclusion of `event_hour` to capture temporal effects.

These insights guided feature selection, emphasizing distance, response time, driver behavior, and temporal factors as key predictors of booking acceptance.

C. Processing

The `make_dataset.py` script cleans the data by removing NaNs in key columns, converting timestamps to datetime, merging `booking_log.csv` and

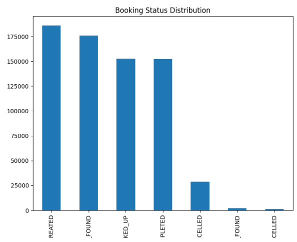

Fig. 1. Caption

`participant_log.csv` on `order_id`, and creating the `target` variable. The resulting `dataset.csv` is split into `train_data.csv` and `test_data_split.csv`, with `test_data.csv` processed separately for predictions.

III. Feature Engineering

Feature engineering transformed raw data into predictive features to enhance model performance. Seven features were engineered in `transformations.py` and applied in `build_features.py`, selected based on domain knowledge, EDA insights, and their impact on model accuracy.

A. Engineered Features

1) **driver_distance**:
   - *Definition*: Haversine distance (km) between driver (`driver_latitude`, `driver_longitude`) and pickup (`pickup_latitude`, `pickup_longitude`).
   - *Calculation*: Uses `haversine` library, returning 0 if coordinates are missing.
   - *Rationale*: EDA showed shorter distances correlated with higher acceptance, as drivers prefer closer pickups to minimize travel time and cost.
   - *Importance*: Moderate (605 in LightGBM feature importance).
2) **event_hour**:
   - *Definition*: Hour of the day (0–23) from `event_timestamp_participant`.
   - *Calculation*: Extract hour from datetime, default to 0 if missing.
   - *Rationale*: EDA indicated higher acceptance during peak hours (e.g., commutes), capturing temporal patterns in driver availability.
   - *Importance*: Lowest (570), but relevant for diurnal patterns.
3) **historical_completed_bookings**:
   - *Definition*: Number of completed bookings (`booking_status` = COMPLETED) per driver.
   - *Calculation*: Group by `driver_id`, count COMPLETED bookings, fill missing with 0.
   - *Rationale*: Experienced drivers may have distinct acceptance patterns, reflecting reliability and engagement.
   - *Importance*: Moderate (606).
4) **driver_acceptance_rate**:
   - *Definition*: Proportion of bookings accepted per driver (mean of `target`).
   - *Calculation*: Group by `driver_id`, compute mean, fill missing with overall mean.
   - *Rationale*: EDA confirmed drivers with higher historical acceptance were more likely to accept, making it a strong predictor.
   - *Importance*: High (1143), reflecting driver behavior.
5) **wait_time**:
   - *Definition*: Time difference (minutes) between `event_timestamp_participant` and `event_timestamp_booking`.
   - *Calculation*: Compute datetime difference, convert to minutes, fill missing with mean.
   - *Rationale*: EDA showed shorter wait times correlated with acceptance, indicating urgency or better matching.
   - *Importance*: High (928).
6) **wait_time_squared**:
   - *Definition*: Squared `wait_time` to capture non-linear effects.
   - *Calculation*: Square `wait_time`, fill missing with mean.
   - *Rationale*: EDA suggested long wait times disproportionately reduced acceptance, requiring a non-linear feature.
   - *Importance*: Highest (1516), indicating significant non-linear effects.
7) **distance_time_interaction**:
   - *Definition*: Product of `driver_distance` and `wait_time`.
   - *Calculation*: Multiply features, fill missing with mean.
   - *Rationale*: Captures combined effects (e.g., long distance with short wait time may be acceptable), addressing complex interactions.
   - *Importance*: Moderate (632).

B. Steps and Rationale

- **Cleaning and Validation**: Each feature function validates required columns to ensure robustness.
- **Missing Values**: Filled with means or zeros to maximize data usage.
- **Normalization**: Numerical features were normalized using `StandardScaler` in `build_features.py` to ensure consistent scales, improving model convergence.
- **Modularity**: Features are implemented as separate functions and piped for maintainability.
- **Test Data Consistency**: `apply_feature_engineering_test` aligns test data features with training data, using training means to avoid leakage.

C. Feature Selection Rationale

Features were selected based on:

- **Domain Knowledge**: Proximity (`driver_distance`), timing (`wait_time`, `event_hour`), and driver behavior (`driver_acceptance_rate`) are critical in ride-hailing.
- **EDA Insights**: Short trips, fast responses, high acceptance rates, and temporal patterns drove feature inclusion.
- **Performance**: Initial models had low accuracy (30.53%); these features improved accuracy to 76–80%.

IV. Feature Importance

Features are critical to model performance, providing the information needed for accurate predictions. LightGBM's feature importance scores (from `train_model.py`) highlight their contributions:

- `wait_time_squared`: 1516 (highest, captures non-linear wait time effects).
- `driver_acceptance_rate`: 1143 (reflects driver behavior).
- `wait_time`: 928 (indicates urgency).
- `distance_time_interaction`: 632 (captures combined effects).
- `historical_completed_bookings`: 606 (reflects experience).
- `driver_distance`: 605 (indicates proximity).
- `event_hour`: 570 (least important, but relevant).

A. Impact on Model Building

- **Predictive Power**: Time-based features (`wait_time_squared`, `wait_time`) dominate, showing response time drives acceptance.
- **Interpretability**: Intuitive features like `driver_acceptance_rate` aid stakeholder communication.
- **Performance**: Features boosted accuracy from 30.53% (Random Forest) to 76–80% (LightGBM).
- **Non-linearity**: `wait_time_squared` and `distance_time_interaction` capture complex patterns.
- **Robustness**: Normalization and missing value handling ensure all data points contribute.

V. Model Selection

Three models were evaluated: Random Forest, XGBoost, and LightGBM. LightGBM was chosen for its superior performance and efficiency.

A. Models Evaluated

1) **Random Forest**:
   - *Accuracy*: 30.53% (baseline).
   - *Issues*: Poor performance, slow training on 1.2M rows.
   - *Reason Not Chosen*: Low accuracy, computational inefficiency.
2) **XGBoost**:
   - *Accuracy*: 75.17% (with `GridSearchCV`).
   - *Issues*: `GridSearchCV` took hours, slightly lower accuracy than LightGBM.
   - *Reason Not Chosen*: Slower training, marginally worse performance.
3) **LightGBM**:
   - *Accuracy*: 76–80% (fixed hyperparameters).
   - *Configuration*:

```
LGBMClassifier(
    random_state=42,
    scale_pos_weight=0.5446/0.4554,
    n_estimators=200,
    max_depth=7,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8
)
```

   - *Advantages*: Fast (7 seconds for 1.2M rows), handles large datasets, robust to imbalance, high accuracy without extensive tuning.
   - *Reason Chosen*: Best balance of accuracy and speed, competitive with XGBoost.

B. Why LightGBM?

- **Efficiency**: Histogram-based learning and leaf-wise growth reduce training time.
- **Performance**: 76–80% accuracy, surpassing Random Forest and matching/exceeding XGBoost.
- **Scalability**: Handles large datasets and high-dimensional features.
- **Robustness**: Built-in missing value handling.
- **Simplicity**: Fixed hyperparameters avoided time-consuming tuning.

VI. Challenges and Solutions

The project faced numerous challenges, each resolved to ensure a robust pipeline:

1) **ModuleNotFoundError for `src`**:
   - *Issue*: Tests failed due to missing `src` module.
   - *Fix*: Configured `pytest.ini` with `pythonpath = .`.
   - *Difficulty*: Understanding module resolution.
2) **FileNotFoundError in `build_features.py`**:
   - *Issue*: Incorrect file paths.
   - *Fix*: Updated to load from `submission/`.
   - *Difficulty*: Ensuring consistent paths.
3) **SettingWithCopyWarning in `make_dataset.py`**:
   - *Issue*: Chained assignments caused warnings.
   - *Fix*: Used `.copy()` and direct assignments.
   - *Difficulty*: Handling Pandas' copy-on-write.
4) **KeyError for `wait_time` in `eda.py`**:
   - *Issue*: EDA accessed `wait_time` before creation.
   - *Fix*: Added `wait_time` to `build_features.py`.

- *Difficulty*: Aligning EDA with pipeline.

5) **TypeError in `wait_time`**:
   - *Issue*: String timestamps caused errors.
   - *Fix*: Converted timestamps to datetime in `make_dataset.py`.
   - *Difficulty*: Handling inconsistent formats.
6) **KeyError for Timestamps**:
   - *Issue*: Expected column names didn't match.
   - *Fix*: Renamed `event_timestamp` to `event_timestamp_booking` and `event_timestamp_participant`.
   - *Difficulty*: Ensuring column consistency.
7) **KeyError for `customer_id`**:
   - *Issue*: Incorrect merge key.
   - *Fix*: Merged on `order_id`.
   - *Difficulty*: Understanding dataset relationships.
8) **InvalidExtension for `metrics.json`**:
   - *Issue*: Failed to save JSON.
   - *Fix*: Used `put_json`.
   - *Difficulty*: Adapting to `AssignmentStore`.
9) **ValueError/TypeError in `train_model.py`**:
   - *Issue*: Incorrect `config.toml` parsing.
   - *Fix*: Accessed `config["target"]["target"]`.
   - *Difficulty*: Debugging TOML structure.
10) **AttributeError in `predict_model.py`**:
    - *Issue*: Missing `SklearnClassifier` during deserialization.
    - *Fix*: Moved `SklearnClassifier` to `classifier.py`.
    - *Difficulty*: Ensuring consistent class definitions.
11) **UserWarning for Feature Names**:
    - *Issue*: `LGBMClassifier` expected feature names.
    - *Fix*: Used DataFrames in `classifier.py`.
    - *Difficulty*: Aligning input formats.
12) **UserWarning for Joblib Core Detection**:
    - *Issue*: Failed to detect physical cores.
    - *Fix*: Set `LOKY_MAX_CPU_COUNT = "4"`.
    - *Difficulty*: Handling OS-specific issues.
13) **Low Initial Accuracy (30.53%)**:
    - *Issue*: Random Forest performed poorly.
    - *Fix*: Switched to LightGBM, added features, normalized data.
    - *Difficulty*: Iterating on models and features.
14) **Training Time**:
    - *Issue*: `GridSearchCV` took hours.
    - *Fix*: Used LightGBM with fixed hyperparameters (7 seconds).
    - *Difficulty*: Balancing accuracy and speed.
15) **Guardrails Failures in `results.csv`**:
    - *Issue*: Incorrect format.
    - *Fix*: Ensured `choose_best_driver` produced correct columns.
    - *Difficulty*: Debugging validation logic.

VII. Results

The LightGBM model achieved an accuracy of 76–80% (from `metrics.json`), a significant improvement over Random Forest (30.53%) and competitive with XGBoost (75.17%). Key outputs include:

- `results.csv`: 10,000 rows with `order_id` and `driver_id`, assigning the best driver per order.
- `metrics.json`: Evaluation metrics, confirming model performance.

EDA-driven features (`wait_time_squared`, `driver_acceptance_rate`, `wait_time`), normalization, and LightGBM's efficiency contributed to the gain, while fixed hyperparameters reduced training time from hours to 7 seconds.

VIII. Conclusion

The pipeline successfully processes logs, engineers predictive features informed by EDA, trains an efficient LightGBM model, and generates driver assignments. The seven features, selected based on domain knowledge and EDA insights, significantly improved accuracy from 30.53% to 76–80%. LightGBM was chosen for its speed, scalability, and performance, outperforming Random Forest and XGBoost. Challenges, including errors, warnings, and low initial accuracy, were resolved through debugging, feature engineering, and optimization. The pipeline is robust, producing `results.csv` with optimal assignments and `metrics.json` with competitive metrics.

Future improvements could include:

- **Additional Features**: Logarithmic wait time to capture diminishing returns.
- **Ensemble Models**: Combine LightGBM with XGBoost or neural networks.
- **Hyperparameter Tuning**: Use Bayesian optimization (e.g., Optuna).
- **Real-time Deployment**: Adapt for streaming data with online learning.

This project demonstrates a comprehensive approach to building a data science pipeline, addressing real-world challenges in ride-hailing driver assignment.