Hotel Booking Probability Prediction: An End-to-End Production-Ready Machine Learning Solution

Problem Definition and Strategic Approach:

The core challenge was to predict the probability of a hotel booking based on user search parameters. This prediction aimed to optimize hotel listing order, thereby enhancing user experience and maximizing conversion rates. The approach is rooted in a robust, data-driven methodology, prioritizing both predictive accuracy and actionable insights.

Solution Summary

I approached the hotel booking probability prediction as a binary classification problem, aiming to categorize user searches into those likely to result in a booking (positive class) and those that wouldn't (negative class). To achieve this, a structured data science pipeline was implemented, starting with efficient data integration from Parquet to CSV format, optimizing data types for scalability. A comprehensive exploratory data analysis (EDA) was then conducted, not just to describe the data, but to strategically uncover patterns and biases. For instance, the discovery of a potential self-fulfilling prophecy in the "rank" feature prompted careful consideration during feature engineering. Recognizing the importance of feature engineering in predictive power, time-based, interaction, and search context features were meticulously crafted. Categorical encoding was employed to transform non-numerical data into a machine-readable format, and outlier handling ensured model robustness. Addressing the inherent class imbalance, undersampling and class weighting were implemented to prevent the model from being biased towards the majority class. The LightGBM algorithm was selected for its efficiency and high accuracy in handling large datasets, and group-aware splitting based on "searchId" ensured the model's generalizability to unseen search contexts. Rigorous model evaluation, using AUC-ROC and AUC-PR, along with feature importance analysis and optimal threshold finding, allowed for fine-tuning the model's performance. Finally, Bayesian optimization was used to efficiently navigate the hyperparameter space, leading to an optimized model. This structured, analytical approach, combining data preprocessing, feature engineering, and model optimization, enabled the successful prediction of hotel booking probabilities.

The results show that the LightGBM model achieved an AUC-ROC of 0.8702, indicating strong discriminatory power, while the AUC-PR of 0.3543 reflects moderate performance in predicting the minority booking class. In this regard, undersampling of the majority class was applied with a 25% ratio with respect to the minoriy class which resulted in an AUC-ROC of 0.8826 and AUC-PR of 0.7652. Showing great promise for exploring data collection for the minority class.

As the time limit of this case study was limited, further exploration and model tuning, model training, and data collection remains as the next steps to help the model capture more underlying patterns of the booking data.

Solution Modules:

Efficient Data Integration (`data_integration.py`):

- I began by loading the dataset from a Parquet file, chosen for its efficient storage

and retrieval of large datasets.
- Optimizing data types (converting `float64` to `float32` and `int64` to `int32`) was crucial for memory efficiency, enabling faster processing and scalability.
- Saving the processed data to a CSV format provided a standardized, easily accessible format for subsequent analysis.

Insightful Exploratory Data Analysis (EDA) (`data_exploration.py`):

- EDA was not merely descriptive; it was a strategic investigation. I aimed to uncover underlying patterns and potential biases that could impact model performance.
- Analyzing seasonality, feature distributions, and correlations allowed us to understand the complex relationships within the data.
- Specifically, analyzing the rank effect highlighted a potential self-fulfilling prophecy, prompting us to consider this bias during feature engineering and model interpretation.
- The goal of the EDA was to provide a very deep look into the data, and provide visual and statistical evidence for the next steps.

Feature Engineering: The Key to Predictive Power:

Transformative Feature Engineering (`feature_engineering.py`):

- Feature engineering was approached as a critical step in extracting meaningful information from the raw data.
- Time-based features (stay duration, day of week) were created to capture temporal patterns in booking behavior.
- Outlier handling (capping numerical values) was essential for robustness, preventing extreme values from skewing the model.
- Categorical encoding (ordinal, one-hot, target encoding) converted categorical data into numerical representations suitable for machine learning.
- Interaction features (rank_price_interaction, review_score_count) were designed to capture complex relationships between variables.
- Search context features (hotels_per_search) provided a deeper understanding of the search behavior.
- Undersampling was used to combat the imbalanced data. This was done to allow the model to learn the minority class.
- Group-aware splitting (based on `searchId`) ensured that test data represented unseen search contexts, leading to more reliable evaluation.

Model Selection and Optimization:

LightGBM for Efficiency and Accuracy (`model_training.py`):

- LightGBM was chosen for its speed, scalability, and ability to handle large datasets with high accuracy.
- Class weighting and undersampling were employed to address class imbalance and improve the model’s ability to predict bookings.

Rigorous Model Evaluation (`model_evaluation.py`):

- Model evaluation went beyond basic metrics. I used AUC-ROC and AUC-PR to assess the model’s ability to rank bookings correctly.
- Feature importance analysis provided insights into the most influential factors driving bookings.
- Optimal threshold finding allowed us to fine-tune the model’s decision boundary for specific performance goals (e.g., maximizing precision or recall).
- Predicting on new data, provided a way to ensure the model was working

correctly.

Bayesian Optimization for Hyperparameter Tuning (`model_tuning.py`):

- Bayesian optimization was used to efficiently search the hyperparameter space, finding the best combination of parameters to maximize model performance.
- This approach allowed us to fine-tune the model for optimal accuracy without exhaustive grid search.

Steps to Run the ML Pipeline (using Docker container)

1. Build the Docker Image

From the project root directory, open the terminal and run the following command:

```
docker build -t my-pipeline-image-v1 .
```

Wait for Docker to build the image.

2. Run the ML Pipeline

Once the image is built, run the following command to execute the machine learning pipeline inside a container:

```
docker run --rm -it my-pipeline-image-v1
```

3. View Results

- The results will be displayed in the terminal.
- Visualizations will be generated in the `output` directory.

4. Run ML Pipeline with Bayesian Hyper-Parameter Tuning

To execute the pipeline with Bayesian hyper-parameter tuning, run:

```
docker run --rm -it my-pipeline-image-v1 python run.py --tuning=true
```

5. Run ML Pipeline with a Desired Undersampling Rate

To execute the pipeline with Bayesian hyper-parameter tuning, run:

```
docker run --rm -it my-pipeline-image-v1 python run.py --undersampling=0.25
```

Run ML Pipeline without Docker container

To execute the pipeline without using the docker approach, first:

```
make venv
```

Then, source the environment:

```
source venv/bin/activate
```

Run the following command in terminal:

```
apt-get update && apt-get install -y make libgomp1
```

Then, run:

```
make run-pipeline
```

For hyper-parameter tuning:

```
make run-pipeline TUNING_FLAG=True
```

For undersampling:

```
make run-pipeline UNDERSAMPLING=0.25
```