AI/ML Engineer Task

Luis Felipe Morales Curiel

Introduction

The objective of this task was to build a classification model that accurately categorizes customer queries into predefined categories such as *Flight Changes*, *Customer Account Issues*, and *Mobile App Issues*. This kind of classifier is especially useful in automating customer support routing or analytics. After reviewing the data and possible approaches, a classical machine learning pipeline was chosen due to its speed, simplicity, and ease of deployment.

Methodology

1. Data Exploration

The dataset contained 20,000 examples distributed across 30 categories. Initial exploration revealed that the input format was generally clean: short, single-sentence queries. A bar plot in Solution.ipynb during the data exploration stage that the label distribution data was imbalanced, some categories had significantly more samples than others.

The queries in the dataset vary in length, both in terms of characters and words. On average, a query is about 12.95 words long and contains roughly 71.89 characters. The shortest query in the dataset has just 3 words, while the longest query contains up to 47 words. This suggests that while most queries are concise, there are some edge cases with more detailed requests. Even though this is true, 47-word length is quite simple, so this can be the first hint we can keep a simple approach without the need of going for something more complex like deep learning solutions.

To gain a high-level understanding of the dataset's content, I analyzed the most frequently used words across all queries (excluding stopwords). The most frequent word was "ryanair", which is expected given the dataset's brand-specific context. Then the most common terms include "flight", "travel", "booking", "flights" and "need", which suggests that users most often inquire about itinerary adjustments, luggage rules, and login or access issues.

From here, I extracted the most frequent words per category class using CountVectorizer, excluding standard English stopwords. For most classes, the top terms are intuitive and closely aligned with the category name, indicating good label coherence. For example:

Group bookings -> group, booking, ryanair

Lost and Found -> lost, flight, ryanair

Seat Selection -> seat, flight, ryanair

Now, while the keywords extracted are informative, their ubiquity may limit their discriminative power for classification. Therefore, this could tell us that using TF-IDF weighting might be useful because it downweights frequently occurring words across all classes (such as "ryanair" or "flight") and emphasizes words that are more unique or specific to a particular class (like "voucher", "complaint", or "miles"). This helps the model focus on the most relevant signals for classification rather than common terms that appear in nearly every query.

Also, the strong lexical signal might suggest that a classifier can learn to distinguish these categories with relatively high confidence.

To ensure the dataset was clean and suitable for modeling, we conducted a basic quality check for duplicate and empty queries. The results were as follows:

Duplicate queries: 32

Empty queries: 0

The presence of 32 duplicate queries in a dataset of 20,000 examples is minimal (0.16%) and unlikely to impact model performance significantly. However, they were removed to reduce potential bias, especially if they appear in multiple classes or are disproportionately weighted during training. No empty queries were found, which confirms that the input data is well-structured and ready for text processing without additional null handling.

Finally, to explore whether there is a relationship between query length and category, which could reveal whether certain types of requests tend to require more detailed explanations (plot shown in Solutions.ipynb). While there are noticeable differences in query length across classes (with a length amoung all classes around, the significant overlap between them suggests that length alone is not a sufficiently discriminative feature for classification.

2. Preprocessing and Pipeline

The solution was implemented using a scikit-learn pipeline. Steps of the pipeline are:

- Text cleaning via TF-IDF vectorization (TfidfVectorizer).
- We trained three pipeline variants: one using raw text, one with lemmatization applied for improved generalization, and a final version with both lemmatization and removal of the word “ryanair,” which was overly common but not class-informative.
- Class imbalance was addressed using class_weight='balanced' in the classifier to add class weighting.

3. Model Selection

Two types of models were considered:

- Classical ML (Logistic Regression, Random Forest): Chosen for interpretability, fast inference, and low computational cost.
- Deep Learning (e.g., BERT): Not pursued due to higher deployment complexity and hardware requirements, which weren't necessary given the simplicity of the queries.

By exploring the data, I realized the queries were simple enough to take a classical approach. If queries would demand a higher complexity due to their length or memory requirements, then a more sophisticated model would have been used. However, keeping things simple are cheaper and easy to maintain. Using BERT or any other deep learning classifier would have overkilled the task. Therefore, the final model was a Logistic Regression classifier trained on the TF-IDF representation of the input queries.

4. Training and Evaluation

- Data was split using an 80/20 stratified split.
- For evaluation metrics I used precision, recall, F1-score (good for class imbalance), and accuracy to evaluate model performance. These metrics are more informative for multi-class and imbalanced classification tasks than ROC-AUC, which is better suited for binary problems. F1-score, in particular, balances precision and recall, offering a clearer view of how well each class is handled.
- We evaluated three model variants: a baseline using raw text, one with lemmatized input, and a final version that also excludes the word “ryanair” due to its low discriminative value.

Results

Model with no lemmatization

To evaluate the model’s performance across all classes, we used a confusion matrix, which provides a clear breakdown of correct and incorrect predictions for each category. In this matrix, accurate predictions appear along the diagonal, while off-diagonal entries indicate where the model confuses one class for another. This makes it easy to visually inspect which classes the model struggles with, if any. The results demonstrate very strong performance across all categories, with most classes achieving F1-scores close to or at 1.00. The overall accuracy on the test set is 99%, and both macro and weighted averages for precision, recall, and F1-score are also 0.99, confirming that the model performs consistently well, even across classes with varying support sizes.

Category	Precision	Recall	F1-Score	Support
Airport Services	0.99	0.97	0.98	99
Baggage Policies	1.00	1.00	1.00	152
Business Travel	1.00	1.00	1.00	102
COVID-19 Policies	0.96	0.94	0.95	130
Check-in Procedures	0.99	0.98	0.99	139
Child and Infant Travel	0.98	0.99	0.99	116
Complaints and Feedback	1.00	1.00	1.00	177
Customer Account Issues	0.99	0.99	0.99	142
Duty-Free Shopping	1.00	1.00	1.00	96
Flight Bookings	0.97	0.96	0.97	145
Flight Changes	0.99	0.98	0.98	156
Flight Status	0.98	1.00	0.99	164
Frequent Flyer Miles	1.00	0.98	0.99	94
Group Bookings	0.99	0.99	0.99	141
In-flight Services	0.99	0.98	0.98	125
Lost and Found	0.98	1.00	0.99	171
Loyalty Programs	0.98	0.98	0.98	109
Mobile App Issues	0.99	1.00	0.99	163
Partnerships and Alliances	1.00	1.00	1.00	83
Payment Issues	0.99	1.00	1.00	182
Pet Travel	0.99	0.99	0.99	98
Promotions and Discounts	1.00	1.00	1.00	138
Refunds and Compensation	0.99	0.98	0.99	153
Seat Selection	0.99	0.99	0.99	144
Special Assistance	0.97	0.98	0.97	145
Travel Documentation	0.99	0.97	0.98	153
Travel Insurance	1.00	1.00	1.00	97
Travel Restrictions	0.95	0.98	0.96	124
Travel Vouchers	0.99	1.00	1.00	144
Weather-related Disruptions	0.99	1.00	1.00	112

Overall Performance

- **Accuracy**: 0.99
- **Macro Average**:
  - Precision: 0.99
  - Recall: 0.99
  - F1-score: 0.99
- **Weighted Average**:
  - Precision: 0.99
  - Recall: 0.99
  - F1-score: 0.99

Logistic regression is inherently interpretable, which is one of its strengths. We can extract feature importance per class directly from the model to understand which words influence classification (Please see Solution.ipynb for class feature visualization). As discussed in the data exploration phase, many queries contain strongly class-specific language, suggesting that the model would benefit from clearly defined linguistic patterns. This assumption is confirmed through an analysis of feature importance using the coefficients of the logistic regression model.

For each class, we extracted the top contributing features, the words that most strongly influence the model's decision. For example: In the Airport Services class, terms like airport, services, lounges, and facilities had the highest impact. For Baggage Policies, the model relied heavily on words such as "baggage", "carry", "luggage", and "weight", etc. With this result we can conclude that the model is making decisions based on meaningful, human-understandable patters in the data without complicated coding. Furthermore, due to the simplicity and efficiency of the logistic regression model combined with TF-IDF features, inference is extremely fast, typically completing in a fraction of a second. This makes the solution ideal for deployment in low-latency environments.

Model with lemmatization

So far, we used a very simple approach where almost no preprocessing is being used. However, we can see that in the top 10 first features the model focuses on both plural and singular forms of the same word, for example, "airport" and "airports". Although the model already achieves a performance of 0.99, we can try to improve generalization and reduce feature redundancy by applying lemmatization, which normalizes words to their base form. This may not significantly change the performance on the current dataset, but it could make the model more robust to unseen variations and reduce noise in the feature space, especially if the dataset is expanded in the future.

Following the same approach as before, we evaluate the impact of lemmatization on model performance. While the overall performance metrics remain nearly identical, a few subtle shifts are noticeable: Some classes (e.g., COVID-19 Policies, Pet Travel) showed slightly lower recall with lemmatization, suggesting that in a few cases, removing inflectional variation may have caused a minor loss of nuance. Other classes (e.g., Special Assistance, Check-in Procedures) experienced minor improvements in precision or F1, showing more consistent prediction confidence.

Lemmatization does not significantly alter model performance in this dataset, likely because the TF-IDF + logistic regression pipeline already generalizes well. However, it offers a cleaner, more compact feature space, which could become more beneficial as the dataset scales or becomes more linguistically diverse. It also aligns better with best practices in NLP preprocessing for long-term maintainability and interpretability. Furthermore, it incorporates new features for classification which aligns better for the interpretability of the class.

Model with lemmatization and removing the word "Ryanair"

In addition to the baseline model and the version with lemmatization, I introduced a third model that combines lemmatization with the removal of the word "Ryanair". This adjustment was motivated by the data exploration phase, where it was evident that "Ryanair" was by far the most frequent term across nearly all classes, making it a poor discriminator. As such, it was treated as a domain-specific stop word.

Upon evaluation, this version performed on par with the lemmatized model in terms of macro and weighted average F1-score (0.99), showing slight improvements in precision and recall for specific classes such as Airport Services and Flight Bookings. The removal of "Ryanair" helped reduce noise in the feature space and improved the clarity of top model features without any negative impact on performance.

This supports the idea that carefully analyzing and tailoring the preprocessing pipeline to domain-specific characteristics, even something as simple as excluding a common brand name, can bring minor but meaningful improvements in robustness and interpretability.

Understanding the model's confidence

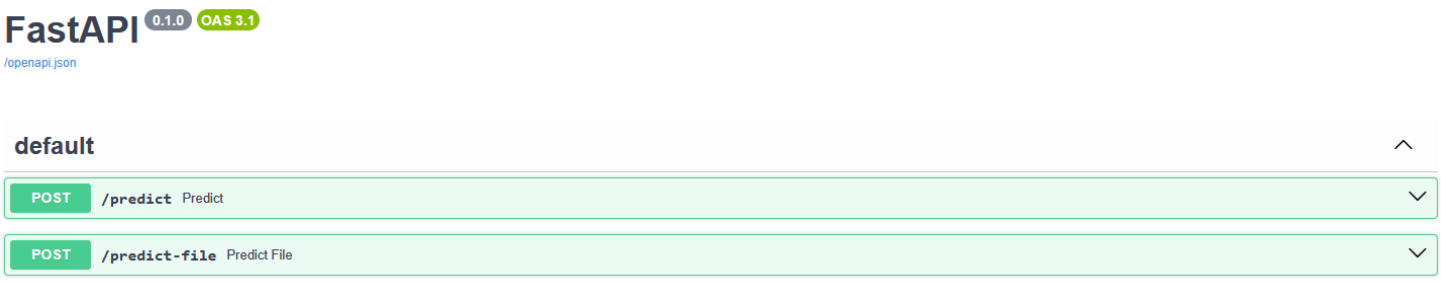
To better understand the model's confidence and reasoning, I explored its class probability outputs using the predict_proba() method. This function provides a full probability distribution over all possible categories for each query, allowing for deeper inspection beyond the top prediction alone.

For instance, the query "Am I able to choose my seat during the flight booking process, and are there any additional fees for certain seats?" is confidently classified as Seat Selection, with a high probability score assigned to the correct class and significantly lower scores for all others ("Flight Bookings", "Baggage Policies", etc.). This illustrates the model's strong certainty when a query contains clearly distinguishable features.

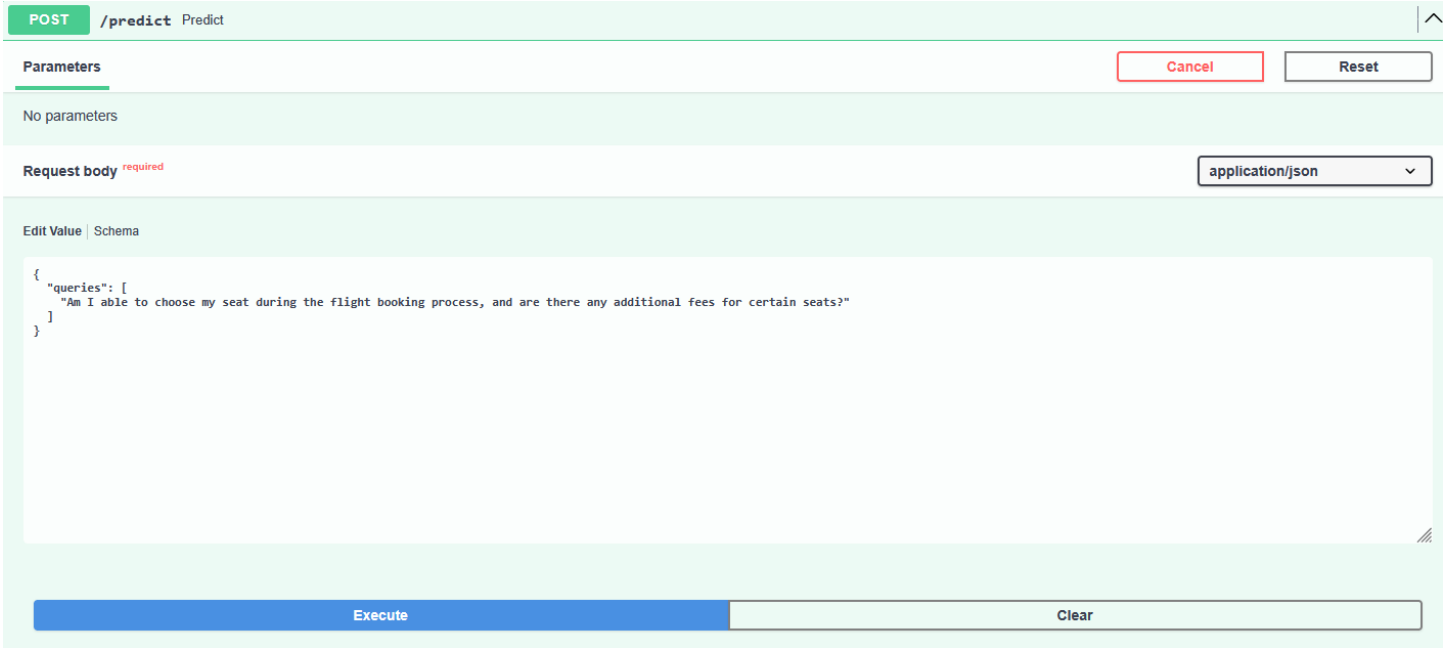
On the other hand, more ambiguous cases reveal interesting nuances. For example, the query "Are masks required on all Ryanair flights?" receives the highest probability for the COVID-19 Policies class, which is appropriate, but the model also assigns non-negligible weight to Travel Documentation. This overlap makes sense from a linguistic perspective, words like "required" and "flights" could be shared among both classes.

5. MLOps Integration

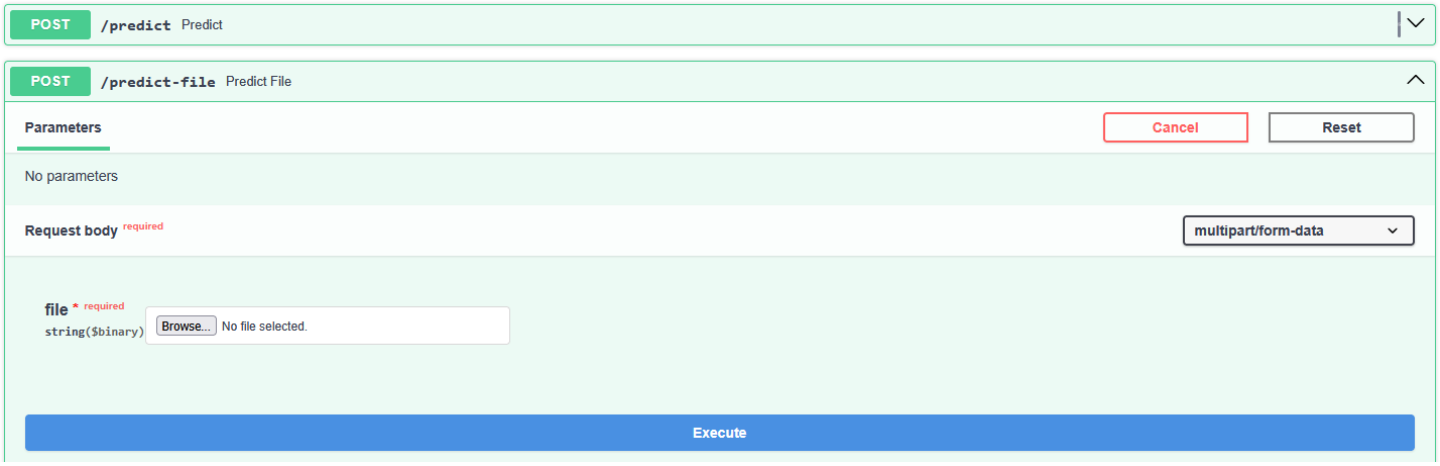
The solution was integrated into a FastAPI web service for production deployment. Two endpoints were created:



- /predict: accepts JSON list of queries for real-time inference. The response is simply the query, lemmatized_query, inference which is the predicted label and the top_5_predictions with the top 5 probabilities for the classification in JSON format. For this write the query you are trying to process, then click execute. Below, a response body will appear showing the probabilities of the result.



- /predict-file: accepts CSV file with query column for batch predictions. If download the response, it will return a JSON files with the query, lemmatized_query, top 5 prob predictions and inference. For this you need to click on browse, look for the csv you are trying to process and click execute. A loading bar will appear and after a couple of seconds the response will be ready for either just visualizing or download it.



Response body


```
[
  {
    "query": "Am I able to choose my seat during the flight booking process, and are there any additional fees for certain seats?",
    "lemmatized_query": "able choose seat flight booking process additional fee certain seat",
    "inference": "Seat Selection",
    "top_1_label": "Seat Selection",
    "top_1_prob": 0.9545,
    "top_2_label": "Flight Bookings",
    "top_2_prob": 0.0048,
    "top_3_label": "Baggage Policies",
    "top_3_prob": 0.0032,
    "top_4_label": "Group Bookings",
    "top_4_prob": 0.0031,
    "top_5_label": "Child and Infant Travel",
    "top_5_prob": 0.0029
  },
  {
    "query": "Am I eligible for a refund if my flight is canceled?",
    "lemmatized_query": "eligible refund flight canceled",
    "inference": "Refunds and Compensation",
    "top_1_label": "Refunds and Compensation",
    "top_1_prob": 0.9268,
    "top_2_label": "Weather-related Disruptions",
    "top_2_prob": 0.0121,
    "top_3_label": "Flight Status",
    "top_3_prob": 0.0066,
    "top_4_label": "Flight Changes",
    "top_4_prob": 0.0037,
```


Download

Response headers

```
content-length: 2687767
content-type: application/json
date: Fri,18 Jul 2025 15:25:42 GMT
server: uvicorn
```

Additional production considerations:

- Models are versioned and saved using joblib
- requirements.txt ensures reproducibility of the environment
- The API can be containerized using Docker
- Can be deployed on a cloud VM or Kubernetes cluster with a WSGI server like Gunicorn
- Logs and metrics can be added with loguru, Prometheus, or Sentry for observability
- CI/CD with GitHub Actions or similar can automate future deployments

This setup makes the solution scalable, portable, and ready for real-world use.

6. Ethical Considerations

- **Bias**: Due to class imbalance, the model may underperform on rare but important categories. This could lead to customer dissatisfaction or misrouting of support tickets.
- **Transparency**: Logistic Regression was chosen partly for interpretability. It's easier to explain than neural network predictions.

- **Privacy**: The model does not store or log personal customer data, but any future deployment should ensure sensitive information is anonymized or encrypted.
- **Fairness**: Ongoing monitoring should be implemented to detect biased behavior or drifting performance over time.