Project Report

Data

Our dataset consists of aircraft-related tweets, each labeled with a specific complaint category known as “negativereason.” Examples of these categories include “Bad Flight,” “Late Flight,” “Customer Service Issue,” “Cancelled Flight,” and “Flight Booking Problems.” The goal is to classify each tweet into the correct complaint category.

The script “Prep.py” handles data loading, filtering, and cleaning. Below are the key steps:

1. Read the dataset from “Tweets.csv.”
2. Keep only the columns “text” and “negativereason.”
3. Retain only the five target categories mentioned above.
4. Remove usernames (tokens beginning with “@”), as they do not provide meaningful information.
5. Remove stopwords using NLTK’s English language stopwords to reduce noise from common words (e.g., “the,” “and,” “of,” etc.).
6. Balance the dataset by undersampling each category to match the size of the smallest category, ensuring an equal distribution of classes.
7. Write the final preprocessed data to “tweets_preprocessed.csv.”

Through these steps, we obtain a cleaner, balanced dataset that is more suitable for training and evaluating text classification models.

Feature Engineering

We employ a TF-IDF (Term Frequency–Inverse Document Frequency) vectorizer to transform the raw tweet text into numerical features:

1. Tokenization: Splits the text into tokens (words).
2. TF (Term Frequency): Measures how often a word appears in a tweet.
3. IDF (Inverse Document Frequency): Downweights words that appear very frequently across all tweets, emphasizing less common but potentially more discriminative words.

By combining TF and IDF into TF-IDF, each tweet is converted into a vector that indicates how relevant specific words are to that tweet across the entire corpus. This numeric representation is crucial for downstream classification models.

Modelling

We trained six different classification models on the TF-IDF-transformed data:

1. SVM (Support Vector Machine) – (cross_validate_svm.py)
2. Random Forest – (cross_validate_rf.py)
3. KNN (k-Nearest Neighbors) – (cross_validate_knn.py)

4. Naive Bayes – (cross_validate_nb.py)
5. SGD (Stochastic Gradient Descent) – (cross_validate_sgd.py)
6. XGBoost – (corss_validate_xgb.py)

Each of these scripts follows a similar structure:
• Load the preprocessed CSV (“tweets_preprocessed.csv”).
• Split the data into 10 stratified folds (10-fold cross-validation).
• Convert the text to TF-IDF features.
• Train and evaluate the model fold by fold, computing accuracy, precision (macro), recall (macro), and F1 (macro) for each fold.
• Summarize and save the confusion matrix across all 10 folds to better understand misclassifications.

Perform Evaluations

All scripts store metrics for each fold in a CSV (e.g., cv_results_svm.csv, cv_results_rf.csv, etc.) and compute the mean accuracy, precision, recall, and F1-score across the 10 folds. Using 10-fold cross-validation is advantageous because:

1. Robust Generalization Check:
   – The dataset is divided into 10 segments, and each segment gets a turn as the test set while the other 9 are used for training. This process repeats until every segment has been used for testing exactly once.
   – This reduces the likelihood that a single peculiarity in the train/test split will skew the results.
2. Efficient Use of Data:
   – Especially important for smaller datasets, 10-fold cross-validation ensures every data point is used for both training (in 9 folds) and testing (in 1 fold).
   – This makes the most out of limited data and produces a more reliable performance estimate.
3. More Stable Performance Estimate:
   – By averaging metrics across 10 folds, we obtain a less biased estimate of how well the model generalizes than with a single train/test split.
   – It also provides an opportunity to observe how consistently the model performs from one fold to another, giving insights into its variance.

This consistent, repeatable approach across all models allows for a fair comparison of their average performance, making it easier to identify a true champion model.

Champion Model

Based on our aggregated F1-scores (macro-averaged) and accuracies, the champion model is the Support Vector Machine (SVM). Its ability to find optimal decision boundaries in the high-dimensional TF-IDF feature space often leads to strong performance in text classification tasks. Nonetheless, in scenarios with more complex or voluminous data, ensemble methods like Random Forest or gradient-boosted solutions such as XGBoost may outperform SVM due to their capacity to capture more nuanced patterns. In this particular dataset, however, the relatively straightforward and robust nature of SVM appears to yield the best overall results.

Error Analysis

By examining the aggregated confusion matrices (summed over 10 folds) for each model, we can identify several notable misclassifications among the five complaint categories: “Bad Flight,” “Late Flight,” “Customer Service Issue,” “Cancelled Flight,” and “Flight Booking Problems.” Below are some recurring themes observed across multiple models:

1. “Late Flight” ↔ “Bad Flight”
   • A recurring source of confusion arises when tweets refer to negative experiences tied directly to flight delays. Models sometimes interpret them purely as “Bad Flight” complaints due to general dissatisfaction, rather than specifically classifying them under “Late Flight.” Conversely, strongly negative language in a delayed flight complaint may mislead certain algorithms into labeling it as a more general “Bad Flight.”
2. “Customer Service Issue” ↔ “Flight Booking Problems”
   • Tweets discussing unhelpful or confusing interactions during booking (such as difficulties changing a reservation, errors in ticketing, etc.) are frequently misclassified. The line between a booking issue and poor service can be blurry if users complain about service agents who fail to resolve the booking problem, resulting in ambiguity.
3. “Cancelled Flight” ↔ “Late Flight”
   • Occasionally, a tweet might describe both a delay and a subsequent cancellation, leading some models to choose “Late Flight” when “Cancelled Flight” would be more accurate. This overlap is particularly troublesome in tweets that mention a significant delay prior to the actual cancellation.
4. Sparse Distinguishing Words for Certain Categories
   • Where tweets lack clear indicator words (e.g., “booking,” “delay,” “cancellation”), classifications rely on more general negative sentiment. This vagueness heightens confusion among similar complaint types, such as “Bad Flight” vs. “Customer Service Issue.”
5. Category Overlap in Language
   • Some tweets blend multiple frustrations (e.g., delayed flights, rude staff, booking mishaps). When multiple categories share similar negative sentiment or combined triggers, algorithms struggle to isolate the primary complaint, manifesting in scattered off-diagonal values within the confusion matrices.

Future Directions Based on Misclassifications

• More Targeted Feature Engineering: Incorporating domain-specific keywords (e.g., “gate change,” “cheap ticket upgrade,” “rebooked,” “compensation offered”) may help models better differentiate between “Flight Booking Problems” and “Customer Service Issue.”
• Handling Ambiguous Cases Explicitly: Tweets mentioning both lengthy delays and eventual cancellations might need specialized processing to pinpoint which complaint aspect dominates.
• Contextual Embeddings: Tools like BERT (instead of pure TF-IDF) could help capture nuanced semantic clues, reducing confusion in tweets that mention multiple grievances.

Testing Results

We performed several visualizations and outputs:

1. Category Distribution Plots ("visualize.py"):
– Bar charts for counts and proportions of each "negativereason" in the balanced dataset.

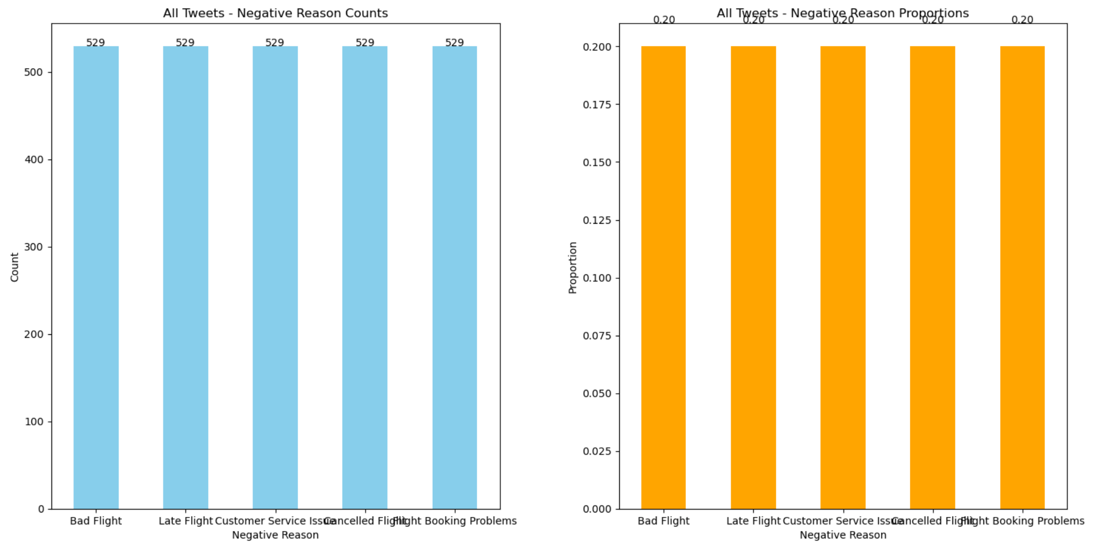
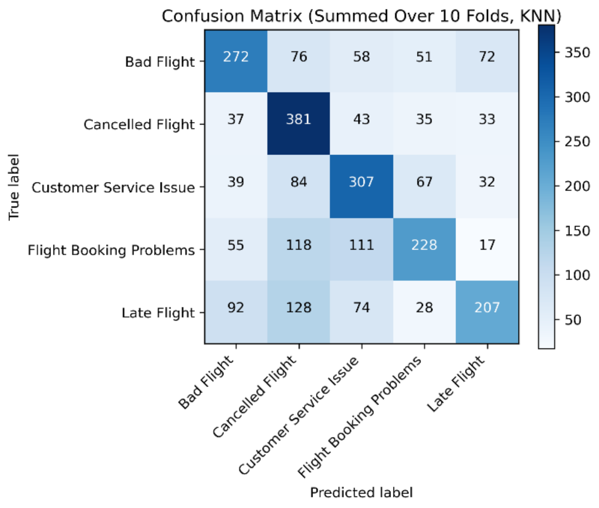


Figure 1

2. Confusion Matrices (Each cross_validate_*.py Script):



	Bad Flight	Cancelled Flight	Customer Service Issue	Flight Booking Problems	Late Flight
Bad Flight	272	76	58	51	72
Cancelled Flight	37	381	43	35	33
Customer Service Issue	39	84	307	67	32
Flight Booking Problems	55	118	111	228	17
Late Flight	92	128	74	28	207

Figure 2a KNN

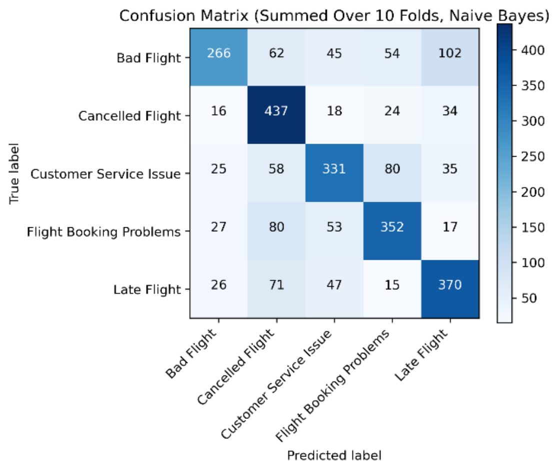


Figure 2b Naïve Bayes

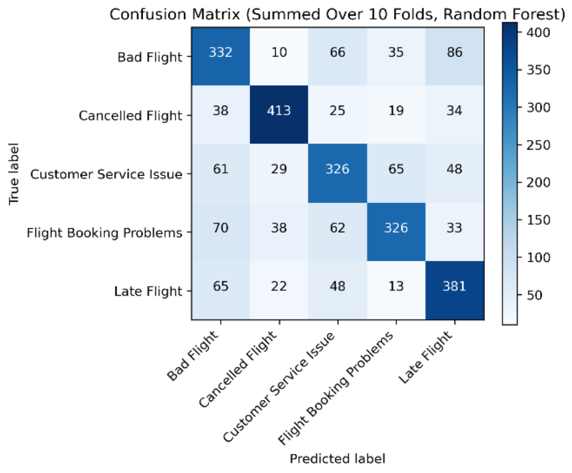


Figure 2c Random Forest

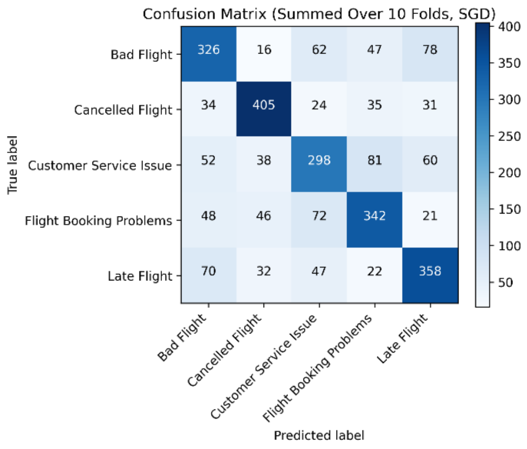


Figure 2d SGD

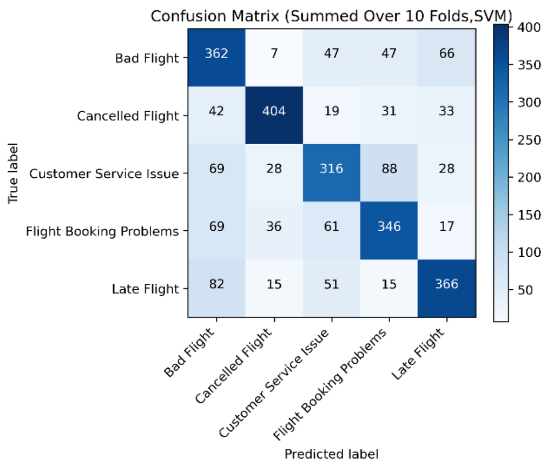


Figure 2e SVM

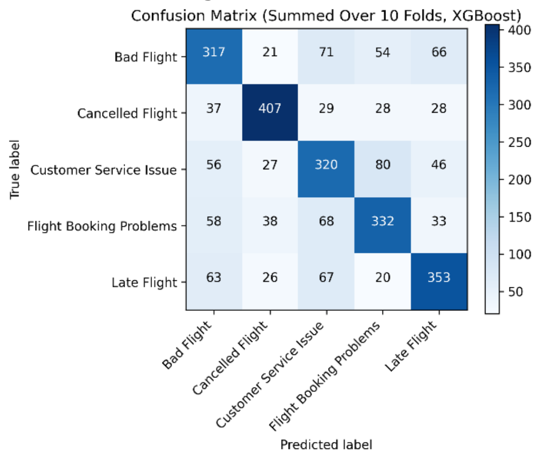


Figure 2f XGBoost

3. Comparison of Metrics:

Test results

Explanation of the results:

• KNN:
This model achieves a mean accuracy of about 0.511 across 10 folds, indicating that it only classifies correctly about half of the test samples on average. The macro precision, recall, and F1 scores are similarly modest at around 0.518, 0.511, and 0.504, respectively, suggesting that KNN struggles to capture more nuanced patterns in this dataset.

• SVM:
SVM shows a consistently strong performance with the highest mean accuracy (~0.689), and comparable macro precision, recall, and F1 (~0.697, 0.689, and 0.691). This indicates that SVM generally finds robust decision boundaries in the feature space, leading to better overall classification across the multiple folds.

• NB (Naive Bayes):
NB achieves a mean accuracy of around 0.665, with macro precision, recall, and F1 hovering close to 0.675, 0.665, and 0.662, respectively. While slightly lower than SVM, NB still provides a reasonable balance between precision and recall, making it a viable choice in contexts with limited data or as a baseline for comparison.

• RF (Random Forest):
The Random Forest model yields a mean accuracy of approximately 0.665, on par with NB. Its macro precision, recall, and F1 values are around 0.672, 0.665, and 0.666, respectively. This indicates a decent overall performance, suggesting that RF can capture interactions among features well but does not surpass SVM on this dataset.

• XGB (XGBoost):
Averaging around 0.654 in accuracy, with macro precision, recall, and F1 near 0.657, 0.654, and 0.654 respectively, XGB performs moderately well. Although it does not reach the level of SVM, it remains robust and might be improved further with hyperparameter tuning given its flexible tree-boosting approach.

• SGD (Stochastic Gradient Descent):
Exhibits a mean accuracy of about 0.658, with macro precision, recall, and F1 nearly 0.659, 0.658, and 0.657, respectively. The performance is close to that of NB and RF. SGD's ability to handle large-scale data efficiently can be advantageous, but it does not outperform SVM on these cross-validation folds.

Summary:
Among the tested classifiers, SVM demonstrates the highest overall accuracy and balanced macro-averaged metrics, making it the top-performing model for this dataset in the experiments shown. The other models—NB, RF, XGB, and SGD—all exhibit moderate performance, while KNN lags behind in this particular comparison.