Intent Classification using ML and FastAPI

Documentation and Performance Report

Introduction

This assessment task involved developing a complete machine learning pipeline for **intent classification** using a labeled dataset of user queries. The objective was to preprocess raw text data, extract meaningful features using vectorization technique, and evaluate multiple classification algorithms including to classify accurately. The task also included model optimization , robust performance evaluation through metrics like accuracy, precision, recall, and F1-score, and visual analysis using confusion matrices.

Following is the in detailed documentation and Performance report:

1. Dataset Analysis

The dataset consists of user queries categorized into distinct intent classes. Each class represents a specific type of user interaction, enabling targeted intent classification and response generation.

1.1 Web_Search Class

Description: Queries related to searching for specific information or facts online.
Example Queries:

- "Search for the best AI courses available online?"
- "How do I create a website?"
- "What is the weather forecast in New York City?"

1.2 General_Chat Class

Description: Casual, non-specific conversations often involving greetings or small talk.
Example Queries:

- "Hello, how are you doing?"
- "I hope you are doing well!"

- "How is you day going so far?"

1.3 Knowledge_Query Class

Description: Queries seeking specific company-related knowledge or information.
Example Queries:

- "How many days before we need to inform before taking a leave from the office?"
- "What is the company's policy on remote work?"
- " Can I take vacation days for a personal event?"

1.4 Calendar_Schedule Class

Description: Queries related to scheduling, managing events, or setting reminders.
Example Queries:

- " Can you schedule a meeting with the legal team to review the documents?"
- "Can you schedule a meeting for tomorrow at 10 AM?"
- " Please schedule a check-in with the marketing team at 11 AM?"

1.5 Email_Send Class

Description: Queries related to composing or sending emails for business purposes.
Example Queries:

- "Send an email to John regarding the project update."
- "Please email the report to the marketing team."
- "Can you send an email to Mark about the new meeting time?"

1. ML Pipeline Implementation

2.1 Data Preprocessing

Text Cleaning and Normalization: Applied text cleaning by removing stop words, converting to lowercase, and normalizing the text for consistency.

Tokenization and Preprocessing Pipeline: Implemented tokenization and a standard preprocessing pipeline, including TF-IDF vectorization and bag-of-words (BoW) methods.

Class Imbalance Handling: Balanced the dataset by increasing the data for underrepresented classes through augmentation.

2.2 Feature Engineering & Model Selection

- Chose **TF-IDF** and **BoW** for feature extraction, as these techniques are effective for text classification tasks.
- Selected traditional ML classifiers—**Random Forest**, **Logistic Regression**, **Naive Bayes**, and **SVM**—to evaluate performance and determine the best model.
- BoW (Bag-of-Words) is chosen for feature extraction because it provided better model performance with higher accuracy, capturing essential word patterns that helped improve classification accuracy. This approach was particularly effective in distinguishing key terms across different classes.

2. Model Development & Optimization

The following models were trained, compared, and the best one was selected:

Naive Bayes: Trained with different parameter settings and evaluated for performance.

Logistic Regression: Hyperparameters were optimized to improve classification accuracy.

SVM: Tuned with various kernels and regularization parameters for optimal performance.

Random Forest: Various configurations for **n_estimators** and **max_depth** were tested to improve model efficiency.

The **Random Forest** model, after hyperparameter tuning, was selected as the best-performing model based on its superior accuracy and consistency.

3.1 Hyperparameter Tuning

Hyperparameter tuning was performed for each algorithm using the **GridSearchCV** approach. This method tested different parameter combinations for each model, such as **n_estimators**, **max_depth**, and **max_features** for Random Forest, and **kernel** for SVM, among others. By exhaustively searching through predefined parameter grids, the best performing set of hyperparameters for each algorithm was identified. This optimization process ensured that the final models were tuned for maximum performance.

3.2 Cross Validation

Cross-validation with **k=5** was applied to each model to ensure robust evaluation. This technique divided the dataset into 5 folds, allowing each model to be trained and tested on different data subsets, helping to assess its generalizability. The cross-validation scores for each model were compared to identify the best-performing one while minimizing overfitting.

3. Performance Report

Core Metrics: Evaluated models primarily based on accuracy, comparing the performance across different classifiers. Precision, recall, and F1-score were also considered for a comprehensive assessment of model effectiveness.

Visualizations: Generated confusion matrices to visually analyze model predictions and identify misclassifications.

Model Analysis: Focused on accuracy reports and confusion matrices to evaluate model performance and understand the classification results.

Model	Train Set Accuracy	Test Set Accuracy	Best Cross-Validation Score
SVM	97.02%	96.72%	96.19%
Logistic Regression	96.91%	95.90%	96%
Naive Bayes	96.81%	89.34%	90.13%
Random Forest	99.69%	95.90%	95.17%

4.1 Accuracy Reports Analysis

Following are the Accuracy Reports of all the four models:

Random Forest - Classification Report (Test Data):

	precision	recall	f1-score	support
0	1.00	1.00	1.00	23
1	0.92	1.00	0.96	24
2	1.00	0.96	0.98	26
3	1.00	0.87	0.93	23
4	0.89	0.96	0.93	26
accuracy			0.96	122
macro avg	0.96	0.96	0.96	122
weighted avg	0.96	0.96	0.96	122

Gaussian Naive Bayes - Classification Report (Test Data):

	precision	recall	f1-score	support
0	0.74	1.00	0.85	23
1	1.00	0.88	0.93	24
2	0.96	0.88	0.92	26
3	0.88	0.91	0.89	23
4	0.95	0.81	0.88	26

Logistic Regression - Classification Report (Test Data):

	precision	recall	f1-score	support
0	1.00	1.00	1.00	23
1	0.92	1.00	0.96	24
2	1.00	0.92	0.96	26
3	1.00	0.91	0.95	23
4	0.89	0.96	0.93	26
accuracy			0.96	122
macro avg	0.96	0.96	0.96	122
weighted avg	0.96	0.96	0.96	122

SVM - Classification Report (Test Data):

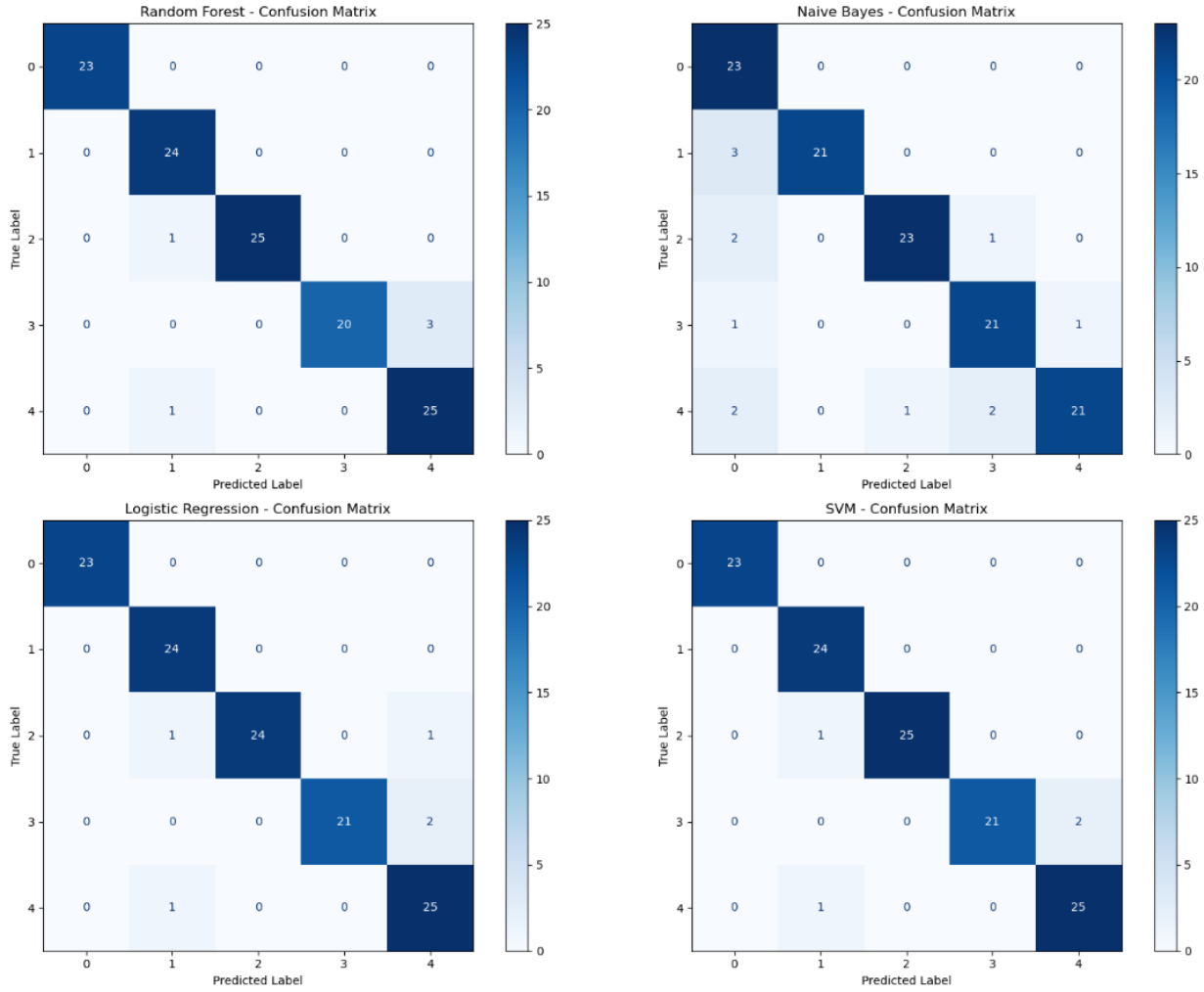
	precision	recall	f1-score	support
0	1.00	1.00	1.00	23
1	0.92	1.00	0.96	24
2	1.00	0.96	0.98	26
3	1.00	0.91	0.95	23
4	0.93	0.96	0.94	26
accuracy			0.97	122
macro avg	0.97	0.97	0.97	122
weighted avg	0.97	0.97	0.97	122

The classification reports confirm the performance observed in the confusion matrices. Random Forest, Logistic Regression, and SVM all achieve high overall accuracy scores (96–97%) and balanced class-wise precision, recall, and F1-scores, typically above 0.90. These metrics

demonstrate that the models are both precise and reliable across all intent categories. In contrast, Gaussian Naive Bayes shows a noticeably lower accuracy of 89%, with reduced F1-scores for several classes (e.g., 0.74 precision for class 0, and only 0.81 recall for class 4), indicating inconsistent predictions. While all models are viable, Random Forest was chosen as the final model due to its ability to handle non-linear relationships, robustness to overfitting, and competitive performance across all evaluation metrics.

4.2 Confusion Matrices Analysis

Following are the Confusion Matrix plots of all the four models:

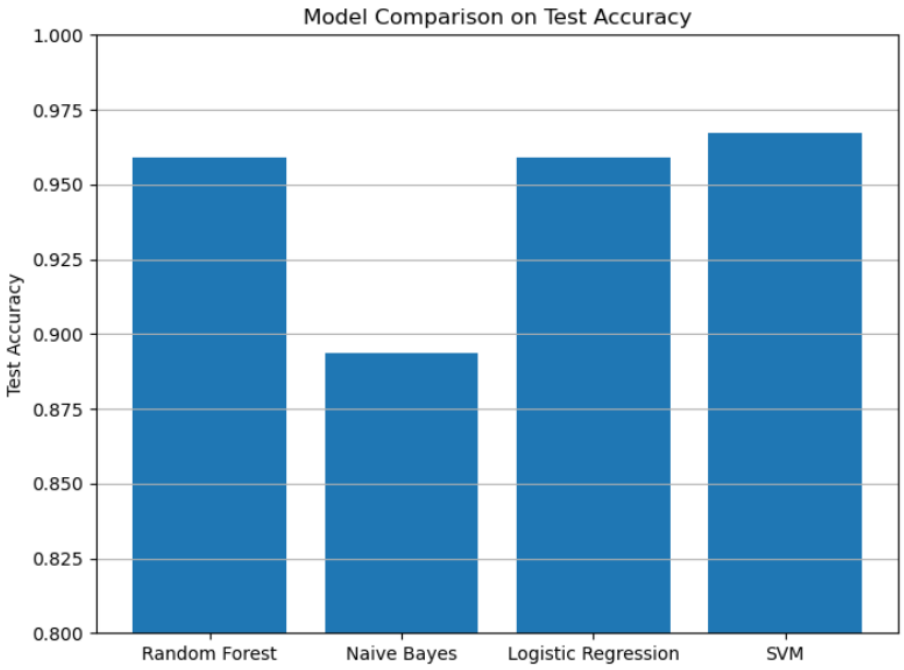


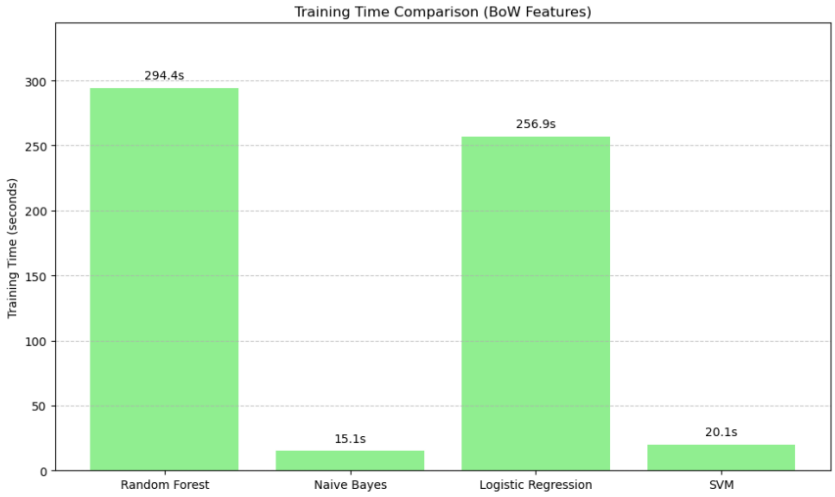
The confusion matrix plots for all four models (Random Forest, Logistic Regression, SVM, and Gaussian Naive Bayes) provide a visual representation of model performance per class. Random

Forest, Logistic Regression, and SVM show strong diagonal dominance, indicating that most predictions fall correctly into their respective classes, especially for class labels 0, 1, 2, and 4. These models exhibit minimal misclassification, with only a few off-diagonal values. On the other hand, the Gaussian Naive Bayes matrix reveals more frequent misclassifications, particularly in classes 1, 2, and 4, where predictions are often confused with neighboring classes. This visual discrepancy aligns with the model's comparatively lower accuracy and highlights its limitations in handling overlapping or less distinct class boundaries.

4.3 Accuracy and Training Time Comparison Plots

Following are the plots for Test Accuracy and Training Time comparison:





Final Model Selection: Random Forest

Despite comparable accuracies between **Logistic Regression**, **SVM**, and **Random Forest**, I chose **Random Forest** due to its ability to handle non-linear data effectively. Additionally, its tree-based structure allows for visualizing splits based on impurity, providing better interpretability and insight into how the model separates classes. This made **Random Forest** the preferred choice for this task.

4. Error Analysis & Insights

Categorizing Errors

- **False Positives**: The model is classifying some **web_search** queries as **general_chat**. This happens when the model misinterprets a query that can be both informational and conversational as a general chat.
- **False Negatives**: **General_chat** queries are being misclassified as **web_search**. This typically occurs when the model encounters short or ambiguous queries, defaulting to the **web_search** class due to its dominance in the dataset.

Misclassified Examples & Patterns

Misclassified examples arise due to ambiguity or overlap between intent classes. The model struggled, especially between **web_search** and **general_chat**. Short, incorrect, or incomplete inputs such as "Hello" were consistently misclassified as **web_search**, which suggests a model bias towards predicting **web_search** for brief, unclear queries.

Misclassified Example	Predicted Class	Correct Class
"What's the weather today in Islamabad?"	general_chat	web_search
"How can I create a website using React and MongoDB?"	general_chat	web_search
"Hello" , "Good morning"	web_search	general_chat
"Set a reminder for a dentist appointment"	web_search	calendar_schedule
"What's the tempereture in New York today?"	general_chat	web_search

Reasons for Misclassification

Class Imbalance: The primary reason for these misclassifications is **class imbalance**. Since **web_search** has the highest number of examples (257), the model overfits to this class. This leads the model to favor **web_search**, especially for ambiguous or incomplete , lack of context input queries are provided by user.

Model Bias Toward Frequent Classes: Due to the overrepresentation of **web_search**, the model develops a bias toward predicting this class, even when the input might be ambiguous or incomplete, such as with short greetings or unclear queries.

Insufficient Data for General Chat: The **general_chat** class lacks enough variety in conversational examples (e.g., greetings, small talk), making it difficult for the model to accurately classify short, vague conversational queries. This leads to misclassification, especially when the input doesn't fit the **web_search** mold.

5. Steps for Improvement

Following are steps performed for improving the model:

Data Pruning of Web_Search Class

To improve the dataset and reduce redundancy, **data pruning** was applied to the **web_search** class. Several steps were taken to ensure the model would not overfit and could generalize better:

1. **Removal of Redundant Queries**: Sentences that began with repetitive phrases like "search for this" or "search for that" were removed, as they did not provide additional meaningful variation for the model
2. **Reclassification of Queries**: Queries that could fit better in **general_chat**, like "What's the best place to travel in Europe in 2023?", were deleted from the **web_search** class, ensuring that each class contained only the most relevant examples.
3. **Elimination of Repetitive Phrasing**: Queries with repetitive patterns, especially those that asked about finding affordable items or services in different categories, were removed to ensure the model wasn't biased towards these similar queries.

Data Augmentation and Class Balancing

To improve the model's performance, I increased the data for the **knowledge_query** (262), **calendar_schedule** (261), **email_send** (260), and **general_chat** (260) classes. This augmentation was specifically aimed at providing the model with more **diverse data** in areas where it was previously struggling. By adding a wider range of examples to these classes, the model can now generalize better and make more accurate predictions.

In addition, this increased data has helped achieve a **balanced dataset**, addressing the issue of class imbalance. Previously, the model was biased toward the more dominant classes, which led to suboptimal performance. With a more balanced representation across all classes, the model is now better equipped to handle a variety of input queries, leading to improved overall accuracy.

Before Data Augmentation:

Class	Number of Queries
web_search	257
general_chat	236
calendar_schedule	235
knowledge_query	235
email_send	234

After Data Augmentation:

Class	Number of Queries
web_search	256
general_chat	260
calendar_schedule	261
knowledge_query	262
email_send	260

Conclusion

After increasing the data for **knowledge_query**, **calendar_schedule**, **email_send**, and **general_chat** classes, the model's performance has improved. The enhanced dataset has helped achieve better balance across the classes, particularly benefiting the **web_search** and **general_chat** classifications. With this balanced data, the model has shown better accuracy on **general_chat** queries and fewer ambiguous predictions for **web_search**. However, challenges still remain in distinguishing between **general_chat** and **web_search** for some queries, as the model continues to misclassify certain inputs as **general_chat**.

6. Suggested Improvements:

1. **Increased Data Augmentation**: Expanding the data through additional **data augmentation** could further help improve the model's performance. By adding more diverse examples, particularly for the **general_chat** and **web_search** classes, the model

will be better equipped to distinguish between subtle variations in these categories. Incorporating techniques such as paraphrasing, synonym replacement, and even generating synthetic data could provide the model with more varied and rich examples to learn from, enhancing its generalization capability.

2. **Feature Engineering**: Enhancing the feature set can improve the model's ability to capture nuanced differences between **general_chat** and **web_search** queries. By extracting additional features, such as specific keywords, contextual cues, or n-grams (bigrams/trigrams), the model can gain better insights into the meaning and intent behind the queries. This will help it separate classes more effectively, particularly in cases where the difference is subtle.

3. **Use of Pretrained Models**: Leveraging **pretrained models** like **BERT** or **RoBERTa** could further enhance the model's ability to handle complex language patterns. These models, trained on vast datasets, can capture deeper semantic relationships and better differentiate between similar queries, such as those belonging to **general_chat** and **web_search**.