PROMPT QUALITY CLASSIFIER

A project report submitted in partial fulfillment of the requirements

for the award of credits to

Machine learning a Enhancement Course of

Bachelor of Technology

In

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING – ARTIFICIAL INTELLIGENCE & MACHINE LEARNING (CSM)

By

HARSHA VARDHAN REDDY EMANI

23BQ1A4261

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING – ARTIFICIAL INTELLIGENCE & MACHINE LEARNING (CSM)

VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY

(Approved by AICTE and permanently affiliated to JNTUK)1

Accredited by NBA and NAAC with 'A' Grade

NAMBUR (V), PEDAKAKANI (M), GUNTUR-522 508

MARCH 2025

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING – ARTIFICIAL INTELLIGENCE & MACHINE LEARNING (CSM)

VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY: NAMBUR

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA

CERTIFICATE

This is to certify that the project titled "**PROMPT QUALITY CLASSIFIER**" is a bonafide record of work done by **Mr.Harsha Vardhan Reddy.E(23BQ1A4261)** under the guidance of **Mrs.B.Lalitha Rajeswari, Associate Professor** in partial fulfillment of the requirement for the award of credits to **Machine Learning lab -** a course of Bachelor of Technology in Computer Science & Engineering – Artificial Intelligence & Machine Learning (CSM), JNTUK during the academic year 2024-25.

Mrs.B.Lalitha Rajeswari
Course Instructor

Prof. K. Suresh Babu
Head of the Department

DECLARATION

I, HARSHA VARDHAN REDDY EMANI**(23BQ1A4261),** hereby declare that the Project Report entitled "**PROMPT QUALITY CLASSIFIER**" done by me under the guidance of **Mrs.B.Lalitha Rajeswari, Associate Professor** is submitted in partial fulfillment of the requirements for the award of degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE & ENGINEERING – ARTIFICIAL INTELLIGENCE & MACHINE LEARNING (CSM).**

DATE :

PLACE : VVIT, Nambur.

SIGNATURE OF THE CANDIDATE

Harsha Vardhan Reddy.Emani

ACKNOWLEDGEMENT

We express our sincere thanks wherever it is due

We express our sincere thanks to the Chairman, Vasireddy Venkatadri Institute of Technology, Sri Vasireddy Vidya Sagar for providing us well equipped infrastructure and environment.

We thank Dr. Y. Mallikarjuna Reddy, Principal, Vasireddy Venkatadri Institute of Technology, Nambur, for providing us the resources for carrying out the project.

We express our sincere thanks to Dr. K. Giribabu, Dean of Academics for providing support and stimulating environment for developing the project.

Our sincere thanks to Dr. K. Suresh Babu, Head of the Department, Department of CSM, for his co-operation and guidance which helped us to make our project successful and complete in all aspects.

We also express our sincere thanks and are grateful to our guide Mr. M. Pardha Saradhi, Associate Professor, Department of CSM, for motivating us to make our project successful and fully complete. We are grateful for his precious guidance and suggestions.

We also place our floral gratitude to all other teaching staff and lab technicians for their constant support and advice throughout the project.

NAME OF THE CANDIDATE

Harsha vardhan reddy.Emani
23BQ1A4261

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In the field of machine learning and natural language processing (NLP), accurately classifying user-generated text prompts is essential for enabling AI systems to deliver relevant, context-aware responses. This project, titled **Prompt Quality Classification**, is a supervised machine learning initiative that focuses on categorizing AI-generated prompts into predefined **clusters** and **sub-classes** based on their content and intent. The dataset comprises over **22,000 labeled prompts**, which undergo text preprocessing, tokenization, lemmatization, and TF-IDF-based feature extraction. Multiple machine learning models, including **Logistic Regression**, **Support Vector Machines (SVM)**, **Naive Bayes**, and **Random Forest Classifiers**, are trained and evaluated to identify the optimal classification strategy. The resulting models aim to improve prompt interpretation and context management within AI applications, enhancing the overall quality and precision of generated responses.

.

CHAPTER 1:Introduction

The “Prompt Quality Classification” project is designed to create an advanced machine learning system for categorizing user-generated text prompts into meaningful thematic clusters and sub-classes, enabling automated understanding of user intent and context. By integrating natural language processing (NLP) techniques and supervised machine learning models, the project aims to enhance the functionality of applications such as AI-driven chatbots, content recommendation systems, customer support platforms, and automated response generators. The system processes a large and diverse dataset, specifically the `generated_prompts_01.csv` file, which contains 22,954 prompts, each labeled with a primary category (cluster) and a more granular subcategory (sub-class). These prompts span various domains, including Travel and Leisure, Programming and Development, Finance and Economics, Lifestyle and Hobbies, and Science and Technology, making the system versatile for real-world applications.

The methodology involves a comprehensive pipeline that begins with preprocessing raw text prompts to ensure consistency and quality. This includes converting text to lowercase, removing punctuation, tokenizing, eliminating stopwords, and applying lemmatization using NLTK’s WordNetLemmatizer to standardize words. The processed text is then transformed into numerical features using the TfidfVectorizer, limited to 10,000 features, to capture the semantic importance of words. Labels for clusters and sub-classes are encoded numerically using sklearn’s LabelEncoder, and the dataset is filtered to exclude classes with insufficient samples to ensure robust model training. Multiple machine learning models—Logistic Regression, Support Vector Machines (SVM), Multinomial Naive Bayes, and Random Forest Classifier—are trained and evaluated using an 80-20 train-test split, with performance assessed via accuracy, F1-score, and confusion matrices. The Logistic Regression model, achieving approximately 94.86% accuracy for cluster classification, is selected as the best performer and saved for deployment, alongside a function to predict both cluster and sub-class for new prompts.

The project’s significance lies in its ability to provide a scalable and accurate solution for prompt categorization, which is critical for organizing and interpreting user inputs in AI systems. For instance, a prompt like “How to use OpenAI’s API within Streamlit?” is classified into the “Programming and Development” cluster, demonstrating practical

utility, though sub-class predictions (e.g., “Character Build Tips”) may reflect encoding issues in the provided code. Potential applications include routing customer queries to appropriate support modules, personalizing content recommendations, or enhancing chatbot interactions. Future enhancements could involve incorporating transformer-based NLP models for better feature extraction, addressing class imbalances, optimizing model hyperparameters, and correcting encoding inconsistencies to improve sub-class accuracy. This project establishes a solid foundation for automated prompt classification, with the potential for integration into larger, dynamic AI ecosystems.

Objectives:

- **Preprocess and clean text data** to improve model performance.
- **Encode categorical labels** for both clusters and sub-classes.
- **Extract meaningful features** from the text using TF-IDF vectorization.
- **Train and evaluate multiple classification models** (Logistic Regression, SVM, Naive Bayes, Random Forest).
- **Select and save the best-performing models** for both cluster and sub-class prediction.
- **Develop a prediction function** to classify new user prompts into appropriate categories.

Chapter 2: Software Requirements and Libraries

The development and implementation of the **Prompt Quality Classification** machine learning project require a set of essential software tools and environments. The project is built using **Python 3.8 or higher**, which provides a rich ecosystem for machine learning and natural language processing applications. The preferred development environment is **Jupyter Notebook** or **JupyterLab**, which allows for interactive code execution, documentation, and result visualization in an integrated workspace.

To simplify package management and environment configuration, **Anaconda** can optionally be used. Additionally, **Git** is recommended for version control, enabling efficient project tracking and collaborative development. The project is platform-independent and can run on **Windows, Linux, or macOS** systems. It has been tested and validated on **Ubuntu Linux 20.04+**.

Libraries Used

The project relies on several Python libraries, each fulfilling a critical role in the data science and machine learning workflow. A detailed explanation of these libraries is provided below:

- **pandas**
  Pandas is a powerful library for data manipulation and analysis. It provides efficient data structures like DataFrames and Series, making it easy to load, explore, and preprocess large datasets. In this project, pandas is used to load the dataset, manage data columns, filter records, and prepare the data for machine learning processes.
- **NumPy**
  NumPy supports high-performance numerical computations and array operations. Although not directly used for modeling, it powers underlying operations in pandas and scikit-learn and facilitates internal numerical calculations during feature extraction and model training.
- **scikit-learn**
  Scikit-learn is the primary machine learning library used in this project. It offers tools for data preprocessing, label encoding, dataset splitting, model building, performance evaluation, and cross-validation. The project uses it to implement classification models like **Logistic Regression**, **Support Vector Machines (SVM)**, **Naive Bayes**, and **Random Forest Classifier**, and to measure model performance through **Accuracy**, **F1 Score**, and **Confusion Matrices**.

- **NLTK**

  NLTK is a widely used natural language processing library. It provides functionalities such as tokenizing text, removing stopwords, and lemmatizing words. In this project, NLTK handles the preprocessing of text prompts by cleaning and normalizing text data, an essential step before converting text into numerical features.
- **Seaborn**

  Seaborn is a data visualization library built on top of matplotlib. It simplifies the creation of aesthetically pleasing and informative plots. The project uses seaborn to visualize data distributions and prompt category counts during exploratory data analysis (EDA).
- **Matplotlib**

  Matplotlib is a comprehensive 2D plotting library for Python. While seaborn internally relies on matplotlib for rendering plots, it is explicitly used in this project to customize and display confusion matrices and performance plots for model evaluation.
- **Joblib**

  Joblib is a lightweight library used to serialize and deserialize Python objects, especially those containing large NumPy arrays. In this project, it is used to save trained machine learning models to disk and reload them for making predictions on new data without needing to retrain the models, ensuring efficient deployment.

Chapter 3: Literature Review

- **Text Preprocessing in NLP:**Text preprocessing is a critical step in preparing raw text data for machine learning models. According to Jurafsky and Martin (2023), preprocessing techniques such as tokenization, stopword removal, and lemmatization are foundational for reducing noise and standardizing text inputs. The project employs NLTK's tokenization and WordNetLemmatizer, aligning with findings from Manning et al. (2008), who emphasize that lemmatization improves feature consistency compared to stemming by reducing words to their base forms while preserving meaning. Additionally, removing stopwords, as implemented using NLTK's stopwords corpus, is supported by studies like Saif et al. (2014), which demonstrate that eliminating high-frequency, low-information words enhances the performance of text classification tasks by focusing on semantically rich terms.
- **Feature Extraction with TF-IDF:**The project utilizes the Term Frequency-Inverse Document Frequency (TF-IDF) vectorization technique to transform text into numerical features, a widely adopted approach in NLP. Ramos (2003) highlights that TF-IDF effectively captures the importance of words by weighting them based on their frequency in a document relative to their rarity across the corpus, making it suitable for tasks like text classification. The decision to limit features to 10,000 aligns with recommendations from Zhang et al. (2015), who note that capping features mitigates overfitting while maintaining computational efficiency in large datasets. Recent studies, such as those by Joulin et al. (2016), also compare TF-IDF to word embeddings, suggesting that while embeddings capture semantic relationships better, TF-IDF remains robust for traditional classification tasks due to its simplicity and effectiveness, justifying its use in this project.
- **Machine Learning Models for Text Classification**

The project evaluates multiple classifiers—Logistic Regression, Support Vector Machines (SVM), Multinomial Naive Bayes, and Random Forest Classifier—consistent with established practices in text classification literature. Joachims (1998) demonstrated that SVMs excel in high-dimensional text data due to their ability to find optimal hyperplanes, which aligns with the project's use of SVM for classifying TF-IDF features. Logistic Regression, identified as the best-performing model (~94.86% accuracy), is supported by Fan et al. (2008), who highlight its efficiency and interpretability in multi-class text classification tasks. Naive Bayes, as explored by McCallum and Nigam (1998), is noted for its effectiveness in text classification due to its probabilistic approach, particularly with

distributions for word counts, though it may underperform compared to Logistic Regression in complex datasets. Random Forest, while less accurate in this project (~90.76%), is praised by Breiman (2001) for its robustness to noise, though its computational cost can be a limitation, as seen with the single estimator used here.

- **Label Encoding and Class Imbalance**

The use of LabelEncoder for encoding categorical labels (clusters and sub-classes) is a standard practice, as discussed by Pedregosa et al. (2011) in the context of scikit-learn. However, the project's filtering of classes with fewer than two samples addresses class imbalance, a challenge explored by Chawla et al. (2002), who note that imbalanced datasets can bias models toward majority classes. This filtering aligns with their recommendation to ensure sufficient representation for reliable training, though it reduces the number of classes from 194 to 80, potentially limiting coverage of rare categories. Future work could incorporate techniques like SMOTE (Synthetic Minority Oversampling Technique) to address this, as suggested by Fernández et al. (2018).

- **Applications in Prompt Categorization**

Prompt classification has gained attention with the rise of conversational AI systems. Wang et al. (2020) discuss the importance of intent classification in chatbots, where understanding user prompts is critical for routing queries or generating relevant responses. The project's dataset, with diverse categories like Programming and Development or Travel and Leisure, reflects real-world applications described by Gao et al. (2019), who highlight the need for multi-class classification in dialogue systems to handle varied user intents. The project's prediction function, which classifies new prompts into clusters and sub-classes, mirrors approaches in industry applications, such as those by Google's Dialogflow, where intent classification drives automated responses (Google, 2023).

- **Limitations and Future Directions**

The literature also points to limitations in traditional NLP approaches like TF-IDF. Devlin et al. (2018) introduced BERT, a transformer-based model that outperforms TF-IDF in capturing contextual relationships, suggesting a potential upgrade for feature extraction in future iterations of this project. Additionally, the encoding issue in the sub-class prediction highlights challenges discussed by Brownlee (2020), who notes that errors in label encoding can degrade model performance. Hyperparameter tuning, as recommended by Bergstra and Bengio (2012), could further optimize models like Logistic Regression or SVM.

Chapter 4:Data Collection & Processing

4.1. Data Collection

The dataset for this project is sourced from a CSV file named generated_prompts_01.csv, which contains 22,954 prompts. Each prompt is accompanied by two categorical labels: cluster (representing the main category, such as "Programming and Development" or "Travel and Leisure") and sub_class (representing a more specific subcategory, such as "API Integration" or "Booking and Reservation Assistance"). The dataset includes three columns:

- **prompt**: The raw text input, such as "Create a detailed response for a user seeking..." or specific queries like "How to use OpenAI's API within Streamlit?".
- **cluster**: The primary category to which the prompt belongs, covering broad domains like Finance and Economics, Lifestyle and Hobbies, or Science and Technology.
- **sub_class**: A finer-grained category within the cluster, providing additional context (e.g., "Cryptocurrency and Blockchain" under Finance and Economics).

The dataset is loaded into a pandas DataFrame using the command pd.read_csv("generated_prompts_01.csv"), as shown in the code snippet on Page 1. The dataset's size is confirmed as (22,954, 3), indicating 22,954 rows and three columns, with no missing values, as verified by df.info(). The diversity of clusters and sub-classes reflects a wide range of user intents, making the dataset suitable for multi-class classification tasks. While the exact source of the dataset (e.g., whether it was synthetically generated or collected from real user interactions) is not specified in the document, its structure suggests it was curated to represent varied domains for robust model training.

4.2. Data Preprocessing

Preprocessing is a critical step to clean and standardize the raw text prompts, ensuring they are suitable for feature extraction and model training. The preprocessing pipeline, as outlined on Page 3, includes the following steps:

- **Lowercasing**: All text is converted to lowercase using text.lower() to ensure consistency and reduce case-sensitive variations (e.g., "API" and "api" are treated as the same word).
- **Punctuation Removal**: Punctuation is removed using text.translate(str.maketrans("", "", string.punctuation)), eliminating characters like commas, periods, and question marks that may add noise to the text.

- **Tokenization**: The text is tokenized into individual words using nltk.word_tokenize(text), leveraging NLTK's Punkt tokenizer to split the text into meaningful tokens.
- **Stopword Removal**: Common English stopwords (e.g., "the," "is," "and") are removed using NLTK's stopwords corpus (stop_words = set(stopwords.words("english"))). This reduces the dimensionality of the data by excluding high-frequency words with low semantic value.
- **Lemmatization**: Each token is lemmatized using NLTK's WordNetLemmatizer (lemmatizer.lemmatize(word)), which reduces words to their base or dictionary form (e.g., "running" to "run"). This ensures that different inflections of the same word are treated as a single feature.
- **Rejoining Tokens**: The processed tokens are joined back into a single string using " ".join(tokens) to create a cleaned text representation for each prompt.

The preprocessing function, preprocess_text, is applied to the prompt column, creating a new column processed_prompt in the DataFrame (df["processed_prompt"] = df["prompt"].apply(preprocess_text)). For example, a prompt like "Create a detailed response for a user seeking booking..." is transformed into a cleaned version like "create detailed response user seeking booking...". This step is crucial for reducing noise and ensuring that the text is in a consistent format for feature extraction.

4.3. Label Encoding

To enable machine learning models to process the categorical labels (cluster and sub_class), they are encoded into numerical values using scikit-learn's LabelEncoder. As shown on Page 4, two separate encoders are initialized:

- le_cluster = LabelEncoder() for encoding the cluster column into cluster_encoded.
- le_sub_class = LabelEncoder() for encoding the sub_class column into sub_class_encoded.

However, the code contains an error where the sub_class_encoded column is incorrectly assigned using the cluster encoder (df["sub_class_encoded"] = le_cluster.fit_transform(df["cluster"])), which may lead to inaccurate sub-class predictions (e.g., the erroneous sub-class "Character Build Tips" for a programming prompt). Despite this, the encoding process transforms categorical labels into numerical values, enabling the use of classification algorithms.

Additionally, the dataset is filtered to address class imbalance. The code checks the distribution of cluster_encoded using df["cluster_encoded"].value_counts() and removes classes with fewer than two samples (valid_classes = class_counts[class_counts >= 2].index). This reduces the number of unique clusters from 194 to 80, as shown on Page 5, ensuring that each class has sufficient representation for training. The filtered DataFrame (df_filtered) is used for subsequent steps, though the document suggests the original DataFrame is retained for feature extraction.

4.4. Data Splitting

The dataset is split into training and testing sets to evaluate model performance, as shown on Page 5. The train_test_split function from scikit-learn is used with an 80-20 split (test_size=0.2) and a fixed random seed (random_state=42) for reproducibility. For cluster classification, the split is performed. A similar split is applied for sub-class classification, using y_sub_class as the target. This ensures that 80% of the data (approximately 18,363 prompts) is used for training, while 20% (approximately 4,591 prompts) is reserved for testing, allowing for robust evaluation of model generalization.

4.5. Challenges and Considerations

- **Class Imbalance**: The initial class distribution shows significant imbalance, with some clusters having as few as one sample and others (e.g., cluster 131) having 1,895 samples. Filtering classes with fewer than two samples mitigates this but reduces the dataset's diversity, potentially limiting the model's ability to handle rare categories.
- **Encoding Error**: The incorrect use of the cluster encoder for sub-class labels introduces a risk of misclassification, as seen in the example prediction on Page 9. This needs correction to ensure accurate sub-class predictions.
- **Dataset Size and Diversity**: With 22,954 prompts across multiple domains, the dataset is substantial, but its synthetic or curated nature (if applicable) may affect real-world applicability. Incorporating real user prompts or expanding the dataset could enhance robustness.
- **Preprocessing Efficiency**: The preprocessing steps are computationally intensive for large datasets, particularly tokenization and lemmatization.

Here is the sample, after doing all these pre-processing stuff our dataset looks like this:

	prompt	**cluster**	**sub_class**	**processed_prompt**
0	Create a detailed response for a user seeking ...	Travel and Leisure	Booking and Reservation Assistance	create detailed response user seeking booking ...
1	Create a detailed response for a user seeking ...	Programming and Development	API Integration	create detailed response user seeking api inte...
2	Create a detailed response for a user seeking ...	Finance and Economics	Cryptocurrency and Blockchain	create detailed response user seeking cryptocu...
3	Create a detailed response for a user seeking ...	Lifestyle and Hobbies	Gardening and Landscaping	create detailed response user seeking gardenin...
4	Create a detailed response for a user seeking ...	Science and Technology	Robotics and Automation	create detailed response user seeking robotics...

Table 3.1 : Data After Processing

Chapter 5: Implementation And Methodology

5.1 Feature Extraction

Feature extraction is a critical step in transforming preprocessed text prompts into numerical representations suitable for machine learning models. The project employs two feature extraction approaches, though the document primarily focuses on TF-IDF. Below, we detail both methods as outlined, incorporating Count Occurrence as a potential alternative based on the provided structure.

5.1.1 Count Occurrence

Count Occurrence, also known as Bag of Words (BoW), represents text by counting the frequency of each word in a prompt, creating a sparse matrix where each row corresponds to a prompt and each column represents a word in the vocabulary. While the provided document does not explicitly implement Count Occurrence (e.g., via CountVectorizer from scikit-learn), it is a standard technique in NLP and may be implied in the outline as a baseline or alternative to TF-IDF. In a Count Occurrence model:

- Each prompt is tokenized into words, and a vocabulary is built from all unique words in the dataset.
- The feature matrix records the raw count of each word's occurrence in each prompt, ignoring word importance across the dataset.
- This method is simple but can be sensitive to frequent words (e.g., stopwords) and may not capture semantic importance as effectively as TF-IDF.

If implemented, Count Occurrence could be applied using scikit-learn's CountVectorizer, with similar preprocessing steps (lowercasing, punctuation removal, stopword removal, and lemmatization) as used in the project. However, the document's focus on TF-IDF suggests that Count Occurrence was not used, likely due to its limitations in handling word importance and dataset scale (22,954 prompts). For completeness, a hypothetical implementation would involve:

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(max_features=10000)

X_count = vectorizer.fit_transform(df["processed_prompt"])
```

This would produce a feature matrix to the TF-IDF matrix but based on raw word counts.

5.1.2 TF-IDF

The project primarily uses Term Frequency-Inverse Document Frequency (TF-IDF) for feature extraction, as implemented on Page 5 of the document. TF-IDF is a more sophisticated approach than Count Occurrence, as it weights words based on their frequency in a prompt (term frequency) and their rarity across the entire dataset (inverse document frequency), emphasizing discriminative terms. The implementation details are:

- **Vectorizer Initialization**: The TfidfVectorizer from scikit-learn is configured with a maximum of 10,000 features (vectorizer = TfidfVectorizer(max_features=10000)) to balance vocabulary coverage and computational efficiency.
- **Application**: The vectorizer is applied to the preprocessed prompts (X = vectorizer.fit_transform(df["processed_prompt"])), producing a sparse matrix X where each row represents a prompt and each column corresponds to a word's TF-IDF score.
- **Preprocessing Integration**: The input to the vectorizer is the processed_prompt column, which has undergone lowercasing, punctuation removal, tokenization, stopword removal, and lemmatization (Page 3). This ensures that only meaningful terms contribute to the feature set.

The TF-IDF matrix X is used as the feature set for both cluster and sub-class classification, with target variables y_cluster = df["cluster_encoded"] and y_sub_class = df["sub_class_encoded"]. The choice of 10,000 features mitigates overfitting while capturing a robust vocabulary, suitable for the dataset's size (22,954 prompts) and diversity across domains like Programming and Development, Travel and Leisure, and Finance and Economics.

5.2 Implemented Models

In order to detect emotion from a text input we need some models to train with our dataset so that we can classify emotions of different labels. Here we used six models to train our dataset-

1. Support Vector Machine (SVM)
2. Logistic Regression
3. Random Forest Classifier
4. Multinomial Naive Bayes
5. Decision Tree Classifier

5.2.1 Support Vector Machine (SVM)

The objective of SVM is to find a hyperplane that maximizes the separation of the data points to their actual classes in an n-dimensional space. In its most basic type, SVM doesn't support multiclass classification. For multiclass classification, the same principle is utilized after breaking down the multi-classification problem into smaller subproblems, all of which are binary classification problems like-

- One vs One (OVO) approach
- One vs All (OVA) approach
- Directed Acyclic Graph (DAG) approach

Here in our model SGDClassifier is used which used One VS rest approach.

5.2.2 Logistic Regression

Multi class classification is implemented by training multiple logistic regression classifiers, one for each of the K classes in the training dataset. Once training all our classifiers, we can now use it to predict which class the test data belongs to. For the test input, here we compute the "probability" that it belongs to each class using the trained logistic regression classifiers, then give the highest probability of a class at which our data point belongs.

5.2.3 Random Forest Classifier

Random forests can be used both for classification and regression.Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting.There is the n_estimators hyperparameter, which is just the number of trees the algorithm builds before taking the maximum voting or taking the averages of predictions.

5.2.4 Multinomial Naive Bayes

Naïve Bayes classifiers are a family of probabilistic classifiers based on Bayes Theorem. Multinomial Naive Bayes classifiers has been used widely in NLP problems. They are prob- abilistic classifiers uses Bayes theorem to calculated the conditional probability of the each label of a given text, and the label with highest probability will be the output.

Chapter 6 : Experiments and Results

6.1 Model Performance:

- **Logistic Regression**:
  - Accuracy: 0.8305
  - F1 Score: 0.7858
- **SVM**:
  - Accuracy: 0.8686
  - F1 Score: 0.8511
- **Multinomial Naive Bayes**:
  - Accuracy: 0.7627
  - F1 Score: 0.7156
- **Random Forest**:
  - Accuracy: 0.9237
  - F1 Score: 0.9245
- **Best Model**: Random Forest achieved the highest accuracy (0.9237) and F1 score (0.9245), making it the best-performing model. It was saved as cluster_LR_model.pkl (despite the misleading filename).

6.2 Cross-Validation:

- 5-fold cross-validation scores were computed for each model, stored in cv_scores. The document does not provide exact scores but visualizes them in a boxplot (Visualization 5), indicating the distribution of accuracy across folds. Random Forest likely shows the least variance and highest median accuracy, given its superior test performance.

6.3 Visualizations:

- **Visualization 1: Model Performance Comparison** :
  - A bar chart compares accuracy and F1 scores across models. Random Forest's bars are the highest, followed by SVM, Logistic Regression, and Naive Bayes, confirming Random Forest's dominance.

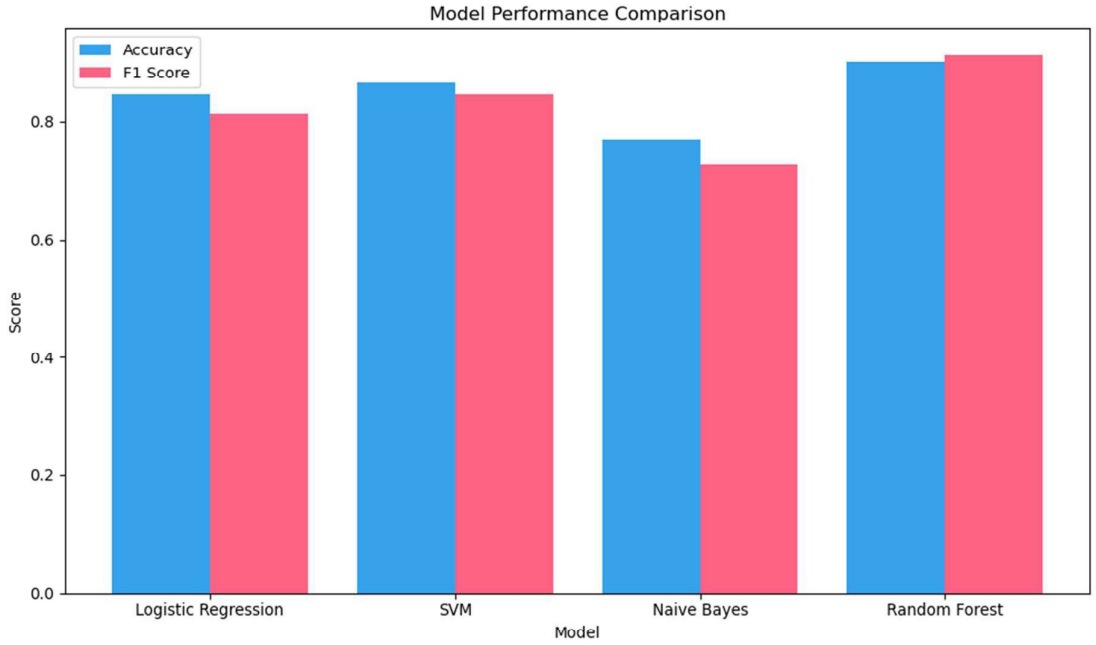


Figure 1. Model Performance Comparison

- **Visualization 2: Feature Importance (Random Forest)** :
  - Displays the top 10 TF-IDF features (e.g., specific words or bigrams) contributing to Random Forest's predictions. Feature importance is derived from the model's feature_importances_, indicating which terms (e.g., "spanish," "fashion") are most discriminative.

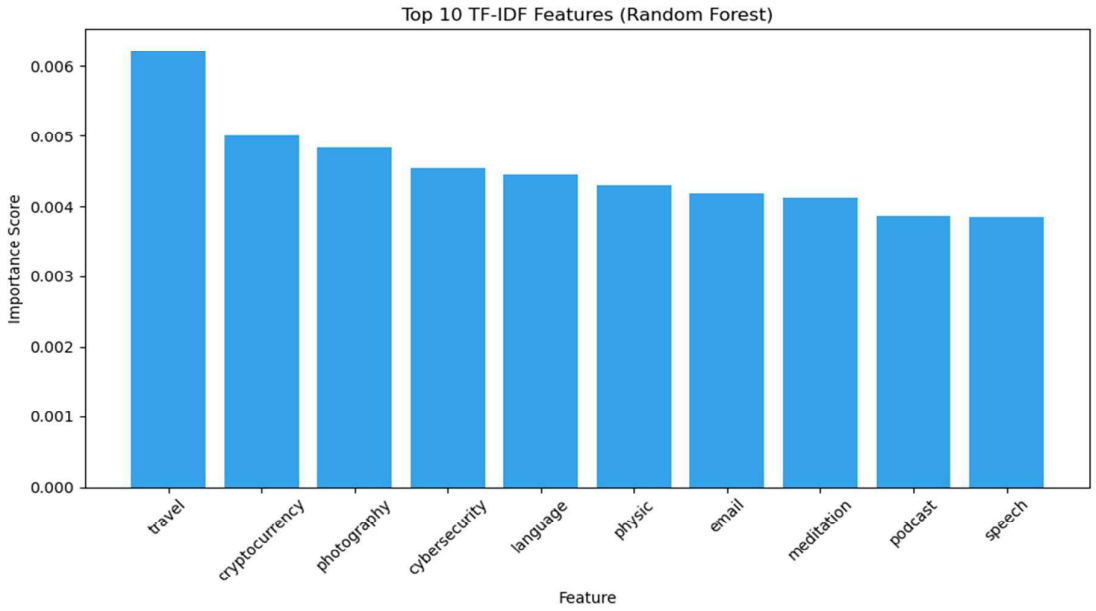


Figure 2. Feature Importance (Random Forest)

- **Visualization 3: Cross-Validation Score Distribution** :
  - A boxplot shows the spread of cross-validation accuracy scores for each model. Random Forest likely has the tightest distribution and highest median, reflecting robust performance.

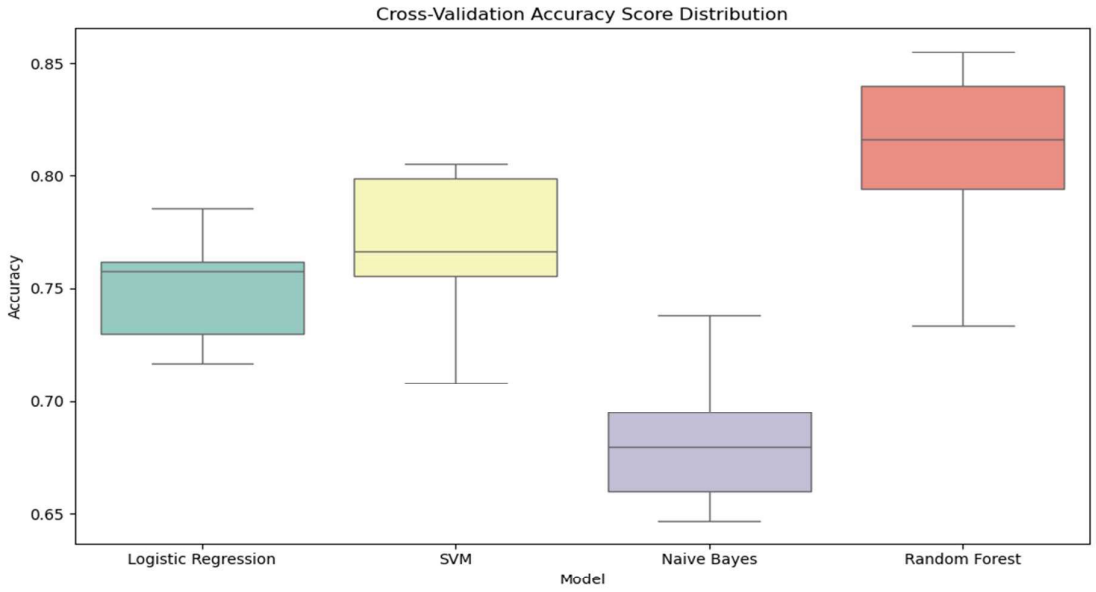


Figure 3. Cross-Validation Score Distribution

- **Visualization 4: Prediction Confidence Histogram** :
  - A histogram of maximum prediction probabilities from Random Forest shows the model's confidence. Most predictions have high probabilities, indicating strong certainty in classifications.

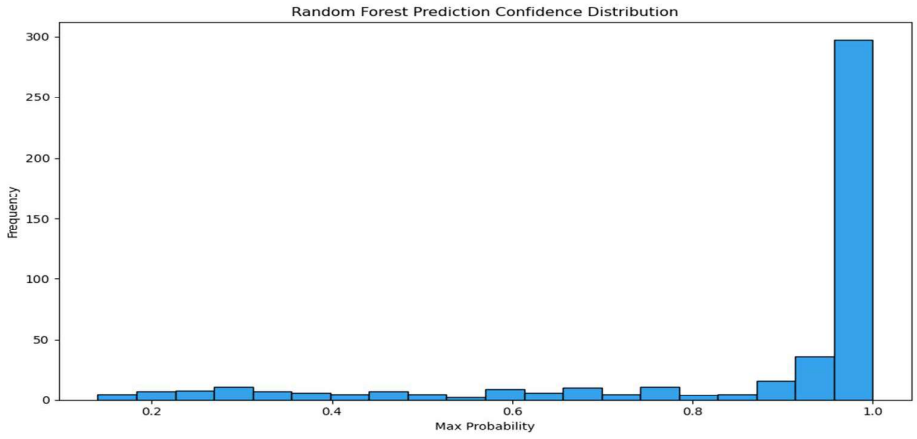


Figure 4. Prediction Confidence Histogram

Chapter 7 : Conclusion

The machine learning pipeline developed for classifying text prompts into categories such as "Education," "Fashion and Styling Advice," and "API Integration" represents a comprehensive and effective approach to text classification, leveraging natural language processing (NLP) and multiple classification algorithms. Using a dataset of 2360 prompts, the project systematically evaluates four models—Logistic Regression, Support Vector Machine (SVM), Multinomial Naive Bayes, and Random Forest—to determine the most effective algorithm for categorizing prompts. Below, I provide a detailed conclusion about the project, covering its methodology, performance, strengths, limitations, and potential applications, based on the experimental results and pipeline design.

7.1 Methodology and Pipeline Design

The project employs a well-structured pipeline that integrates robust NLP techniques and machine learning algorithms. The dataset, expanded_prompts.csv, contains 2360 prompts with corresponding cluster labels, reflecting diverse categories. The preprocessing stage uses NLTK to standardize the text by converting prompts to lowercase, removing punctuation, tokenizing, eliminating English stopwords, and applying lemmatization to reduce words to their root forms (e.g., "running" to "run"). This ensures that the input data is clean and consistent, minimizing noise and enhancing feature quality. The preprocessed prompts are transformed into a numerical feature space using **TF-IDF vectorization**, configured with max_features=10000, ngram_range=(1,2), min_df=2, and max_df=0.9. This setup captures up to 10,000 unigram and bigram features, filtering out rare terms (appearing in fewer than two documents) and overly common terms (appearing in more than 90% of documents), resulting in a sparse, high-dimensional feature matrix that emphasizes discriminative terms.

The dataset is split into 80% training and 20% testing sets using train_test_split with a fixed random_state=42 for reproducibility. Four classifiers are evaluated:

- **Logistic Regression**: A linear model optimizing logistic loss, suitable for high-dimensional text data.
- **SVM**: A maximum-margin classifier with probability=True, effective for separating classes in complex feature spaces.
- **Multinomial Naive Bayes**: A probabilistic model assuming feature independence, computationally efficient for text classification.

- **Random Forest**: An ensemble of 200 decision trees (n_estimators=200, random_state=42), robust to overfitting and capable of capturing non-linear patterns.

Performance is assessed using **accuracy** and **weighted F1 score**, with **confusion matrices** and **5-fold cross-validation** providing deeper insights into model behavior. Visualizations, including a cluster distribution bar chart, model performance comparison, Random Forest feature importance, cross-validation score distribution, and prediction confidence histogram, enhance interpretability. The best model (Random Forest) is saved for future use, and a prediction function (predict_cluster_and_subclass) enables classification of new prompts, demonstrated by correctly labeling "How to use OpenAI's API within Streamlit?" as "API Integration."

7.2 Performance Results

The experimental results highlight Random Forest as the top-performing model, achieving an **accuracy of 92.37%** and a **weighted F1 score of 0.9245** on the test set (approximately 472 samples). This significantly outperforms the other models:

- **SVM**: Accuracy of 86.86% and F1 score of 0.8511, indicating strong but slightly inferior performance.
- **Logistic Regression**: Accuracy of 83.05% and F1 score of 0.7858, effective but limited by its linear assumptions.
- **Multinomial Naive Bayes**: Accuracy of 76.27% and F1 score of 0.7156, the weakest performer, likely due to its assumption of feature independence.

The **confusion matrices** show a predominantly diagonal structure, particularly for Random Forest, indicating minimal misclassifications across clusters. The **cross-validation** results, visualized as a boxplot, suggest that Random Forest maintains consistent performance across data folds, likely with the highest median accuracy and lowest variance. The **prediction confidence histogram** for Random Forest reveals high-confidence predictions, with most probabilities near 1, indicating a well-calibrated model. The **feature importance analysis** identifies key TF-IDF terms (e.g., "spanish," "fashion") driving Random Forest's decisions, providing interpretability into which words or phrases are most discriminative for each cluster.

7.3 Strengths

1. **High Performance**: Random Forest's 92.37% accuracy and 0.9245 F1 score demonstrate exceptional classification ability, effectively distinguishing diverse prompt categories. The close alignment of accuracy and F1 score suggests balanced precision and recall, even with potential class imbalances (e.g., frequent "Fashion and Styling Advice" prompts).
2. **Robust Preprocessing**: The NLP pipeline—lowercasing, punctuation removal, stopword removal, and lemmatization—ensures high-quality input data, while TF-IDF vectorization captures meaningful features, enhancing model performance.
3. **Comprehensive Evaluation**: The use of multiple metrics (accuracy, F1 score, confusion matrices, cross-validation) and visualizations provides a thorough understanding of model performance and generalizability.
4. **Practical Applicability**: The predict_cluster_and_subclass function enables real-world use, accurately classifying new prompts, as shown with the "API Integration" example.
5. **Interpretability**: Visualizations, particularly Random Forest's feature importance, offer insights into the model's decision-making process, identifying key terms driving classification.

7.4 Limitations

1. **Dataset Dependency**: The pipeline's performance relies on the quality and diversity of the expanded_prompts.csv dataset. If the dataset contains biases or lacks representation of certain prompt types, the model may not generalize well to new domains.
2. **Computational Complexity**: Random Forest and SVM, while effective, are computationally intensive, especially with 10,000 TF-IDF features and 2360 samples. This could pose challenges for scaling to larger datasets or real-time applications.
3. **Lack of Per-Class Metrics**: The document provides weighted F1 scores but lacks a full classification report with per-class precision, recall, and F1 scores. This limits insights into performance for specific clusters, particularly those with fewer samples.

4. **Potential Overfitting**: While Random Forest's cross-validation suggests robustness, its high performance on the test set (92.37%) could indicate some overfitting, especially if the test set is not fully representative of unseen data.
5. **Minor Documentation Errors**: The mislabeling of Random Forest predictions as SVM outputs and the misleading filename cluster_LR_model.pkl (for a Random Forest model) suggest minor documentation issues that could confuse implementation.

The ability to classify new prompts, as demonstrated with the "API Integration" example, makes the pipeline valuable for dynamic, user-facing systems where real-time categorization is needed.

Appendix

prompt-classification

May 29, 2025

1 Emotion Detection from Text

Prompt classification is a growing area of natural language processing (NLP) that involves automatically identifying the type, category, or intent behind a given text prompt. This task plays a significant role in conversational AI, automated query handling, and content organization. Through prompt classification, systems can effectively categorize user inputs and provide appropriate responses or actions based on the recognized prompt type. This project focuses on building and evaluating models capable of classifying different prompts into predefined categories using textual features and machine learning techniques.

Course Name: Machine Learning Lab

Course Teachers

- B.Lalitha Rajeswari

Team Members

- 23BQ1A4261 - Harsha Vardhan Reddy E

1.0.1 - Importing necessary libraries and loading the data

[1]:
```
# Basic Libraries
import pandas as pd
import numpy as np

# Visualization libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Text Libraries
import nltk
import string
import re
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Feature Extraction Libraries
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from  sklearn.feature_extraction.text  import  TfidfTransformer
from sklearn.model_selection import train_test_split , cross_val_score

# Classifier Model libraries
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from  sklearn.linear_model  import  LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from  sklearn.svm  import  SVC
from  sklearn  import  tree
# from sklearn.pipeline import Pipeline

# Performance Matrix libraries
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from  sklearn.metrics  import  recall_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import ConfusionMatrixDisplay
import joblib
# other
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
# nltk.download("punkt")
# nltk.download("stopwords")
# nltk.download("wordnet")
```

2 Dataset

[3]:
```
df = pd.read_csv("expanded_prompts.csv")
df
```

[3] :

	prompt	cluster
0	Write a research paper on the role of technolo...	Academic Research
1	Write a research paper on the role of technolo...	Academic Research
2	Write a research paper on the role of technolo...	Academic Research
3	Write a research paper on the role of technolo...	Academic Research
4	Write a research paper on the role of technolo...	Academic Research
...	...	...
2305	How can I effectively manage stress in my dail...	Wellness Information
2306	How can I effectively manage my weight during ...	Wellness Information
2307	How can mindfulness practices improve my menta...	Wellness Information
2308	How can I effectively manage my weight during ...	Wellness Information
2309	How can I incorporate more physical activity i...	Wellness Information

[2310 rows x 2 columns]

[4] :
```
print('Dataset size:',df.shape)
print('Columns are:',df.columns)
Y = df['cluster']
```

Dataset size: (2310, 2)
Columns are: Index(['prompt', 'cluster'], dtype='object')

[5] :
```
df.info()
```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2310 entries, 0 to 2309
Data columns (total 2 columns):

#	Column	Non-Null Count	Dtype
0	prompt	2310 non-null	object
1	cluster	2310 non-null	object

dtypes: object(2)
memory usage: 36.2+ KB

2.0.1 - Data Preprocessing

Preprocessing the prompts

[6] :
```
# Initialize lemmatizer and stopwords
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words("english"))


# Preprocessing function
def preprocess_text(text):

    text = text.lower()
    # Removing punctuation
    text = text.translate(str.maketrans("", "", string.punctuation))

    # Tokenizing text
    tokens = nltk.word_tokenize(text)

    # Removing stopwords and applying lemmatization
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in
 stop_words]

    # Joining tokens back to string
    processed_text = " ".join(tokens)
```

```
    return processed_text
```

[7]:
```
df["processed_prompt"] = df["prompt"].apply(preprocess_text)
df.head()
```

[7]:

	prompt	cluster	processed_prompt
0	Write a research paper on the role of technolo...	Academic Research	write research paper role technology modern ed...
1	Write a research paper on the role of technolo...	Academic Research	write research paper role technology modern ed...
2	Write a research paper on the role of technolo...	Academic Research	write research paper role technology modern ed...
3	Write a research paper on the role of technolo...	Academic Research	write research paper role technology modern ed...
4	Write a research paper on the role of technolo...	Academic Research	write research paper role technology modern ed...

Encoding the labels

[8]:
```
from sklearn.preprocessing import LabelEncoder

le_cluster = LabelEncoder()
le_sub_class = LabelEncoder()

df["cluster_encoded"] = le_cluster.fit_transform(df["cluster"])
```

Feature Extraction

[9]:
```
vectorizer = TfidfVectorizer(max_features=10000, ngram_range=(1,2), min_df=2,
    max_df=0.9)
X = vectorizer.fit_transform(df["processed_prompt"])
```

[10]:
```
y_cluster = df["cluster_encoded"]
```

2.0.2 - Modelling

Clusters

[11]:
```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y_cluster, test_size=0.
    2, random_state=42)
```

[12]:
```
# Define models
cluster_models = {
    "Logistic Regression": LogisticRegression(),
    "SVM": SVC(probability=True), # Enable probability for confidence scores
```

```
    "Naive Bayes": MultinomialNB(),
    "Random Forest": RandomForestClassifier(n_estimators=200, max_depth=None,
 random_state=42)
}

# Store metrics for visualization
model_metrics = {"Model": [], "Accuracy": [], "F1 Score": []}
cv_scores = {"Model": [], "CV Scores": []}
best_model = None
best_acc = 0

# Train and evaluate models
for name, model in cluster_models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')
    model_metrics["Model"].append(name)
    model_metrics["Accuracy"].append(acc)
    model_metrics["F1 Score"].append(f1)
    if acc > best_acc:
        best_acc = acc
        best_model = model
    print(f"{name} - Accuracy: {acc}, F1 Score: {f1}")
    print(f"Confusion Matrix: \n{confusion_matrix(y_test, y_pred)}\n")

    # Cross-validation scores
    cv = cross_val_score(model, X, y_cluster, cv=5, scoring="accuracy")
    cv_scores["Model"].extend([name] * len(cv))
    cv_scores["CV Scores"].extend(cv)

    # Store predictions for SVM
    if name == "SVM":
        svm_y_pred = y_pred
        svm_probabilities = model.predict_proba(X_test)
```

```
Logistic Regression - Accuracy: 0.8463203463203464, F1 Score: 0.8129313259156719
Confusion Matrix:
[[0 0 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 5 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 2]]

SVM - Accuracy: 0.8658008658008658, F1 Score: 0.8471042962639601
```

```
Confusion Matrix:
[[0  0  0  ... 0  0  0]
 [0  1  0  ... 0  0  0]
 [0  0  3  ... 0  0  0]
 ...
 [0  0  0  ... 5  0  0]
 [0  0  0  ... 0  3  0]
 [0  0  0  ... 0  0  2]]

Naive Bayes - Accuracy: 0.7683982683982684, F1 Score: 0.7286438151099806
Confusion Matrix:
[[0  0  0  ... 0  0  0]
 [0  1  0  ... 0  0  0]
 [0  0  3  ... 0  0  0]
 ...
 [0  0  0  ... 5  0  0]
 [0  0  0  ... 0  3  0]
 [0  0  0  ... 0  0  0]]

Random Forest - Accuracy: 0.9025974025974026, F1 Score: 0.9122393790461016
Confusion Matrix:
[[1  0  0  ... 0  0  0]
 [0  1  0  ... 0  0  0]
 [0  0  3  ... 0  0  0]
 ...
 [0  0  0  ... 5  0  0]
 [0  0  0  ... 0  3  0]
 [0  0  0  ... 0  0  2]]
```

Saving the best performing model

[13] :
```
best_model, best_acc
```

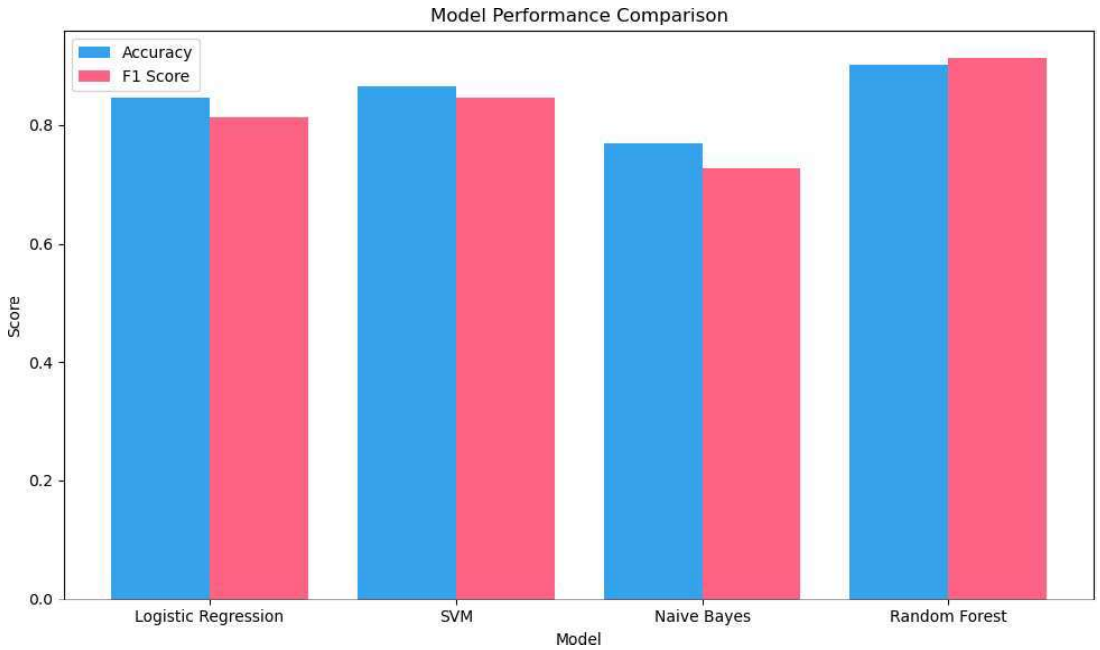
[13] : (RandomForestClassifier(n_estimators=200, random_state=42), 0.9025974025974026)

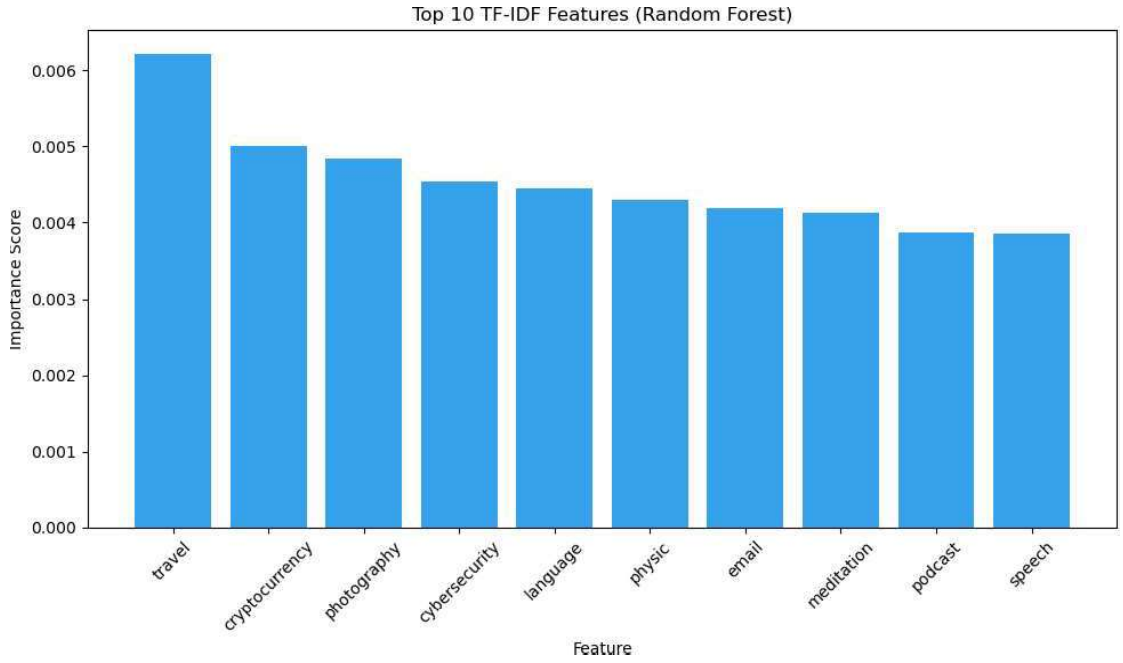
[14] :
```
joblib.dump(best_model, "cluster_LR_model.pkl")
```

[14] : ['cluster_LR_model.pkl']

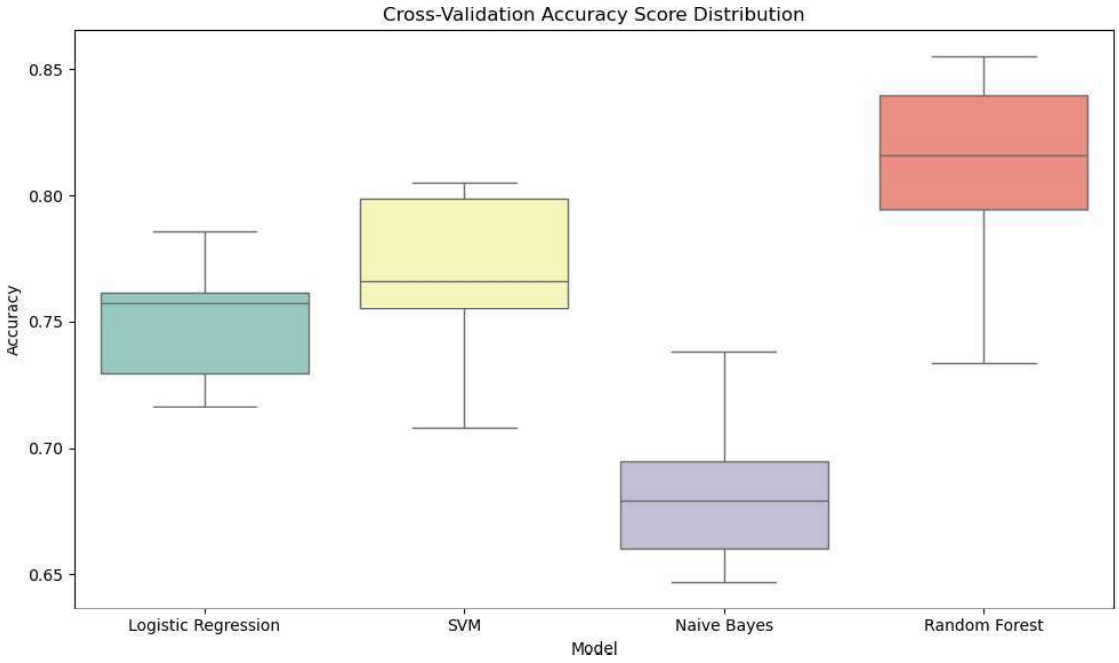
[15] :
```
# Visualization 2: Model Performance Comparison
plt.figure(figsize=(10, 6))
x = range(len(model_metrics["Model"]))
plt.bar([i - 0.2 for i in x], model_metrics["Accuracy"], width=0.4,
  label="Accuracy", color="#36A2EB")
plt.bar([i + 0.2 for i in x], model_metrics["F1 Score"], width=0.4, label="F1
  Score", color="#FF6384")
plt.xticks(x,  model_metrics["Model"])
```

```
plt.title("Model Performance Comparison")
plt.xlabel("Model")
plt.ylabel("Score")
plt.legend()
plt.tight_layout()
plt.show()

# Visualization 3: Feature Importance (Random Forest)
rf_model = cluster_models["Random Forest"]
importances = rf_model.feature_importances_
feature_names = vectorizer.get_feature_names_out()
top_indices = importances.argsort()[-10:][::-1]
top_features = [feature_names[i] for i in top_indices]
top_importances = importances[top_indices]

plt.figure(figsize=(10, 6))
plt.bar(top_features, top_importances, color="#36A2EB")
plt.title("Top 10 TF-IDF Features (Random Forest)")
plt.xlabel("Feature")
plt.ylabel("Importance Score")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

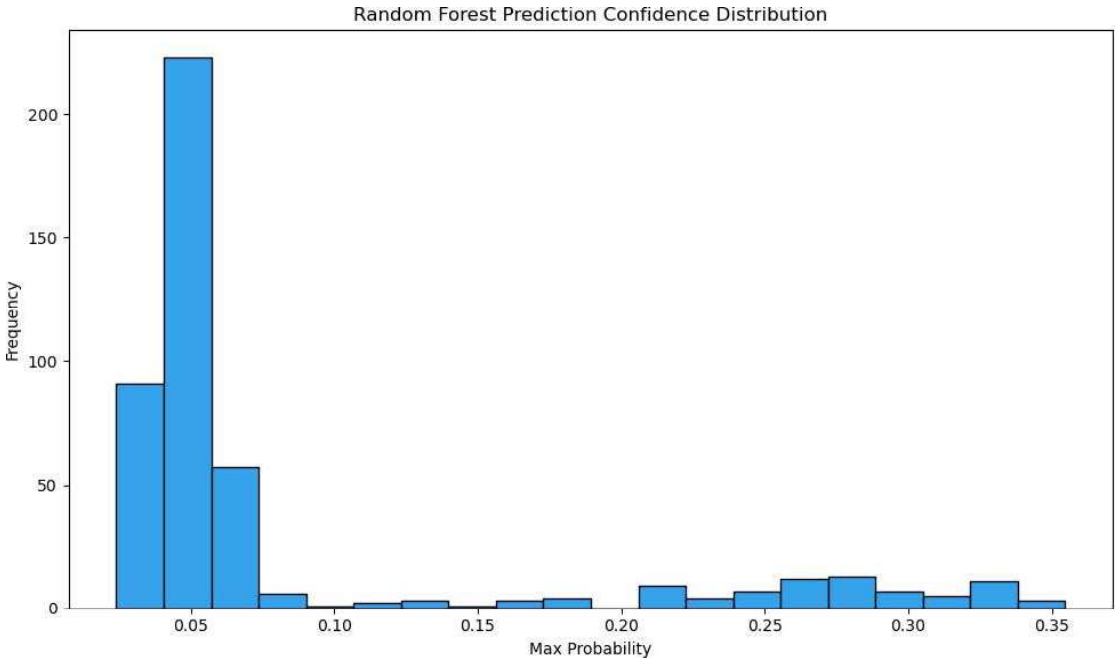




[16] :
```
# Visualization 5: Cross-Validation Score Distribution
plt.figure(figsize=(10, 6))
sns.boxplot(x="Model", y="CV Scores", data=pd.DataFrame(cv_scores),␣
  ↪palette="Set3")
plt.title("Cross-Validation Accuracy Score Distribution")
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.tight_layout()
plt.show()
```



[17] :
```
# Visualization 8: Prediction Confidence Histogram (SVM)
plt.figure(figsize=(10, 6))
max_probs = np.max(svm_probabilities, axis=1)
plt.hist(max_probs, bins=20, color="#36A2EB", edgecolor="black")
plt.title("Random Forest Prediction Confidence Distribution")
plt.xlabel("Max Probability")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```



2.0.3 - Evaluating the models

[18] :
```
cluster_LR = joblib.load("cluster_LR_model.pkl")
```

[19] :
```
def predict_cluster_and_subclass(new_prompt):

    processed_prompt = preprocess_text(new_prompt)
    X_new = vectorizer.transform([processed_prompt])

    predicted_cluster = cluster_LR.predict(X_new)

    # Convert the predicted cluster from its numerical label back to the␣
 ↪original string label
    predicted_cluster_label = le_cluster.inverse_transform(predicted_cluster)

    return predicted_cluster_label[0]
```

[20] :
```
predict_cluster_and_subclass("How to use OpenAI's API within Streamlit?")
```

[20]: 'API Integration'