

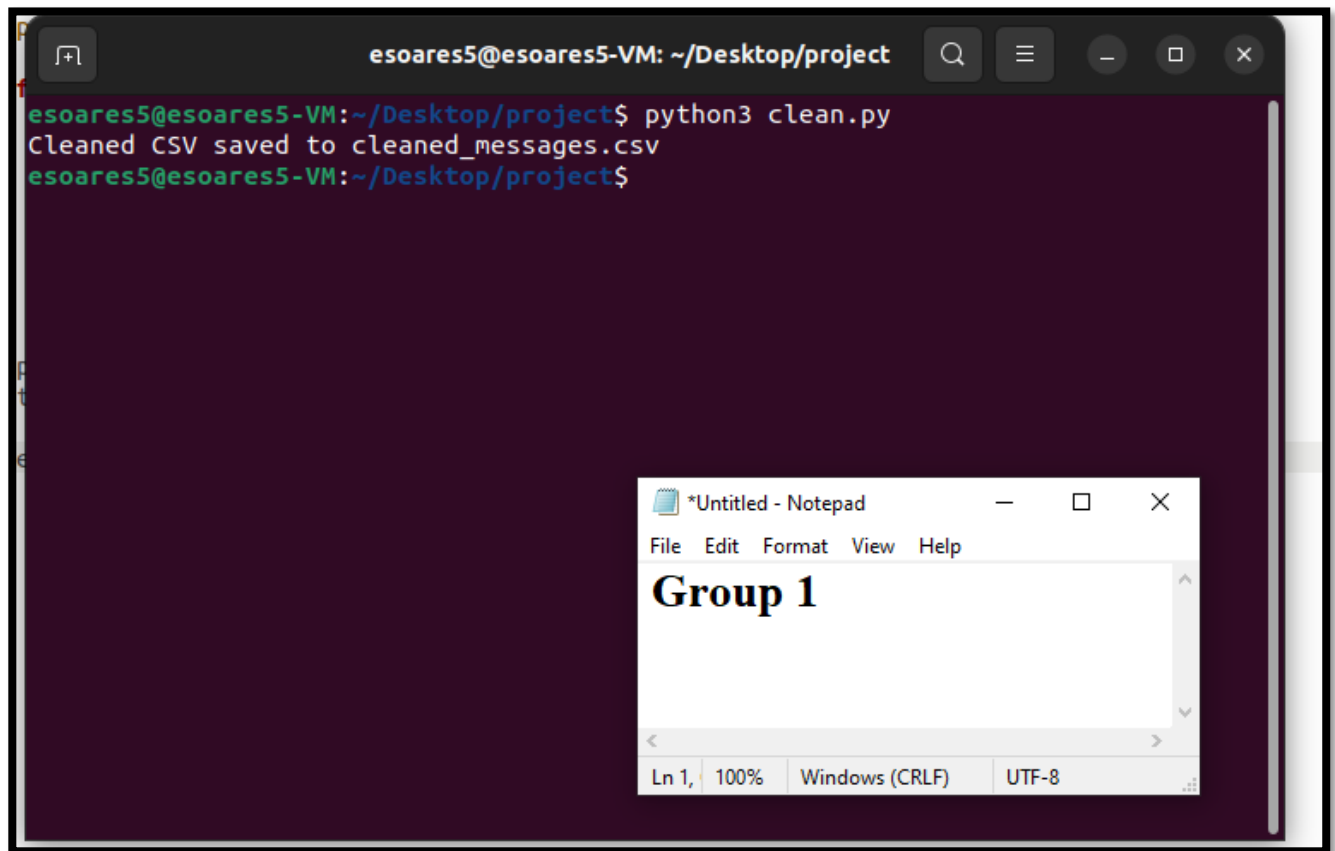
Seneca College
Jia Ying Ou
SRT521
Group 1
Project Phase II

Table of Contents & Figures

Table of Contents & Figures	2
Task 1: Data Preparation	3
➤ Clean.py	3
➤ Code Process	4
Task 2: Feature Transformation	6
➤ Code Process	6
Task 3: Train the Model	7
Task 4: Test the Model	8
Task 5: Test the Model on New Data	10
Work Distribution	13
References	14
<i>Figure 1: clean.py execution</i>	3
<i>Figure 2: task1graphs.py execution</i>	4
<i>Figure 3: Graph creation</i>	4
<i>Figure 4: Ham vs Spam chart</i>	5
<i>Figure 5: task2featsel.py execution</i>	6
<i>Figure 6: task3.py output</i>	7
<i>Figure 7: task4.py output</i>	8
<i>Figure 8: Predicted and Actual Percentage of Spam</i>	9
<i>Figure 9: Confusion Matrix for task5</i>	10
<i>Figure 10: Model Accuracy and ROC Curve</i>	11

Task 1: Data Preparation

➤ Clean.py

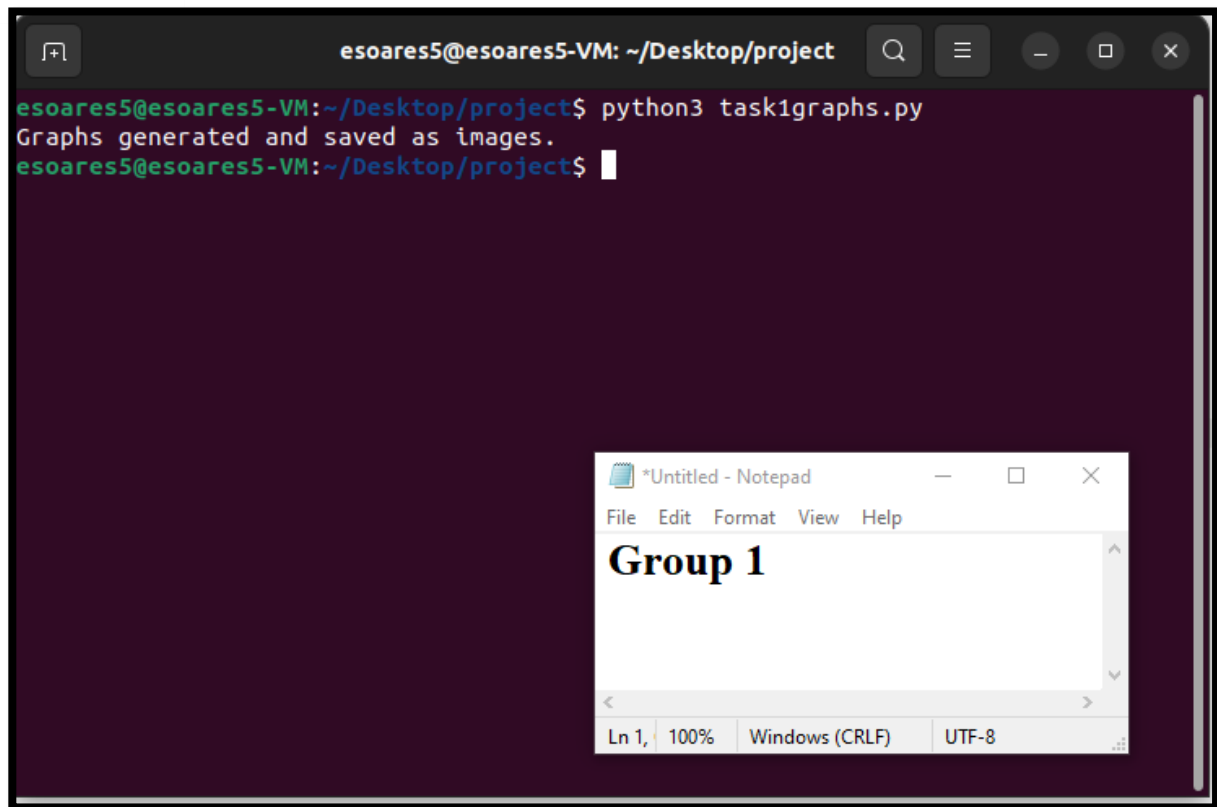


The screenshot shows a terminal window titled 'esoares5@esoares5-VM: ~/Desktop/project'. The command 'python3 clean.py' has been executed, resulting in the output: 'Cleaned CSV saved to cleaned_messages.csv'. Overlaid on the terminal is a Notepad window titled '*Untitled - Notepad'. The Notepad window contains the text 'Group 1' in a large, bold, black font. The status bar at the bottom of the Notepad window indicates 'Ln 1, 100% Windows (CRLF) UTF-8'.

Figure 1: clean.py execution

“The `clean.py` script is a Python program that performs data cleaning operations on a CSV file. It uses the pandas library to read an input CSV file named 'messages.csv' and then removes rows containing missing values (NaN or null values) from the dataset. The resulting cleaned DataFrame is saved to an output CSV file named 'cleaned_messages.csv' without including the index column. If any errors occur during this process, it will catch and print an error message. Overall, the script's main purpose is to clean the input CSV file by removing rows with missing data and save the cleaned data to a new CSV file.

➤ Code Process



The image shows a terminal window titled "esoares5@esoares5-VM: ~/Desktop/project". The terminal output is as follows:

```
esoares5@esoares5-VM:~/Desktop/project$ python3 task1graphs.py
Graphs generated and saved as images.
esoares5@esoares5-VM:~/Desktop/project$
```

Overlaid on the terminal is a Notepad window titled "*Untitled - Notepad". The text "Group 1" is written in the Notepad window.

Figure 2: task1graphs.py execution

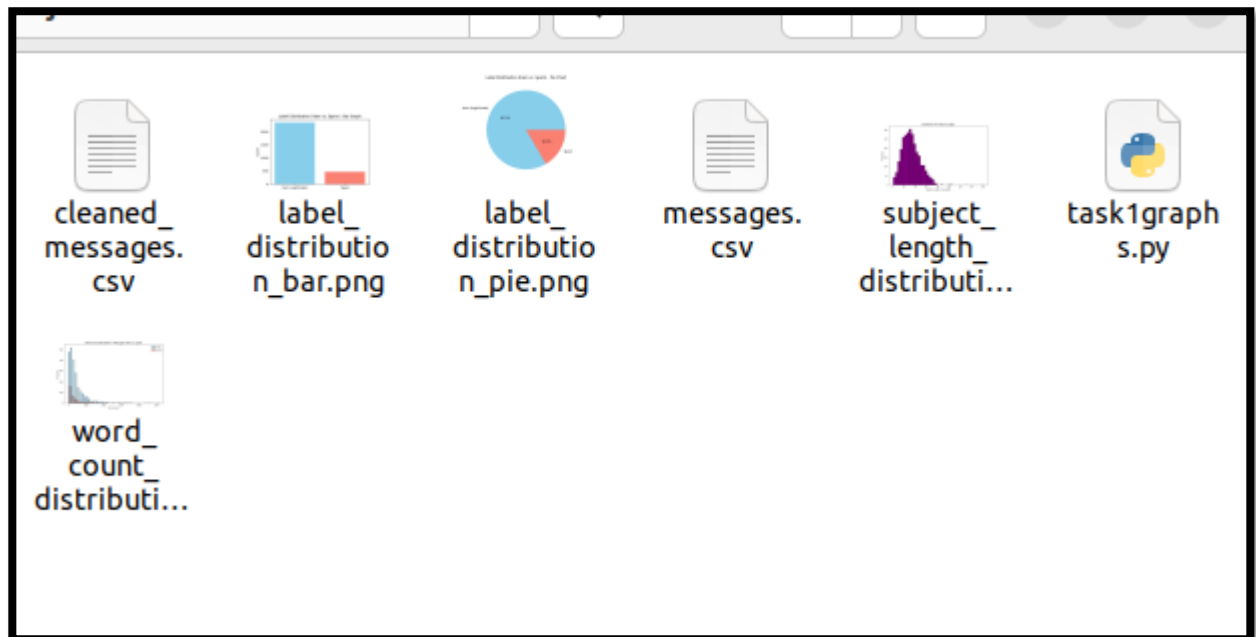


Figure 3: Graph creation

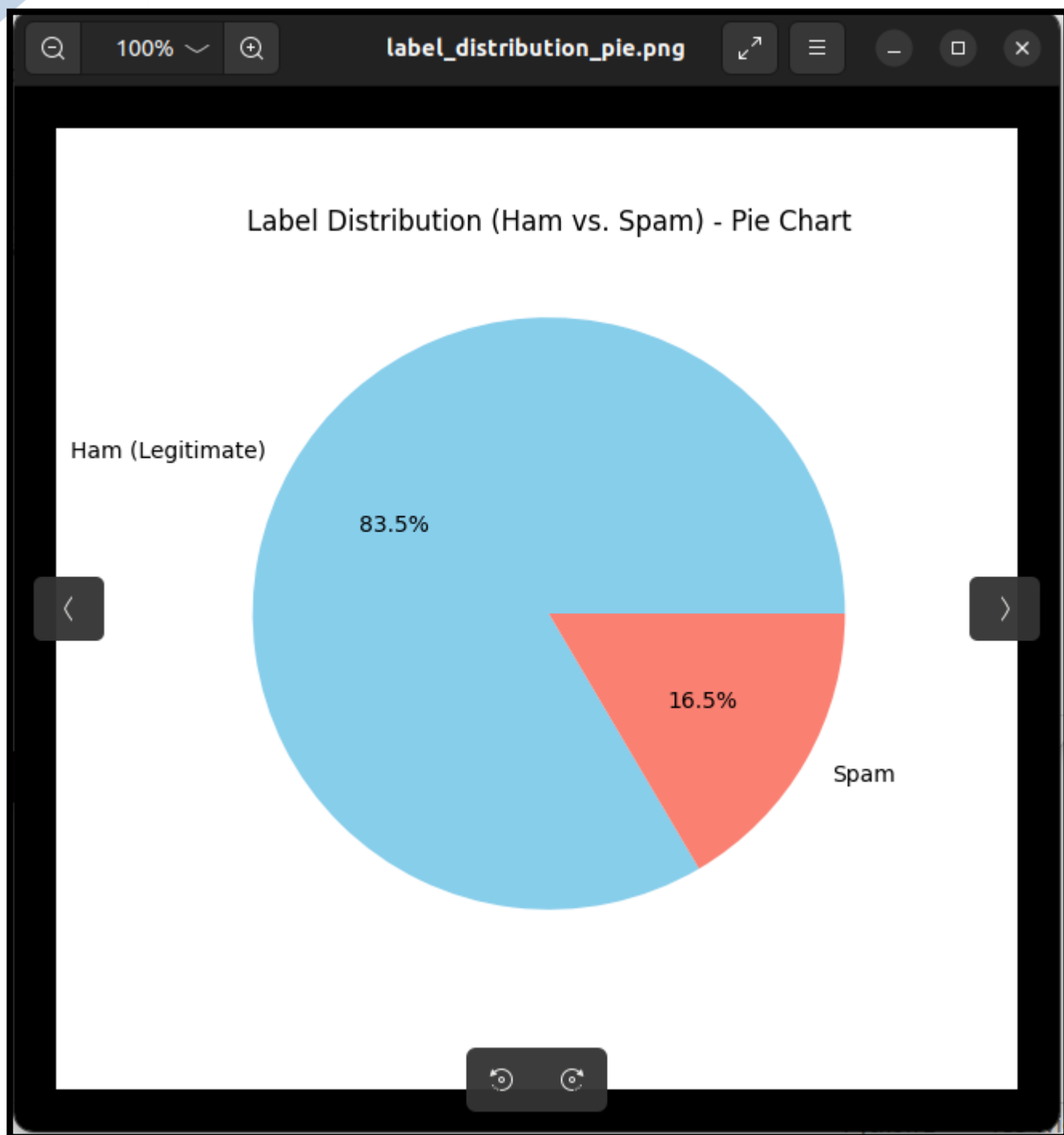


Figure 4: Ham vs Spam chart

The provided Python code conducts various data analysis and visualization tasks on a dataset of messages, potentially for the purpose of spam detection or classification. It begins by importing essential libraries, including pandas for data manipulation, matplotlib.pyplot for graph plotting, Counter from collections for label counting, and re for text cleaning. The 'clean_text' function is defined to clean and preprocess text data by splitting it into words, converting them to lowercase, and then joining them back together as cleaned text.

Next, the 'plot_label_distribution_pie_and_bar' function is responsible for generating two types of plots to visualize the distribution of 'Ham' (legitimate) and 'Spam' labels within the dataset. It creates both a pie chart and a bar graph, saving them as 'label_distribution_pie.png' and 'label_distribution_bar.png,' respectively.

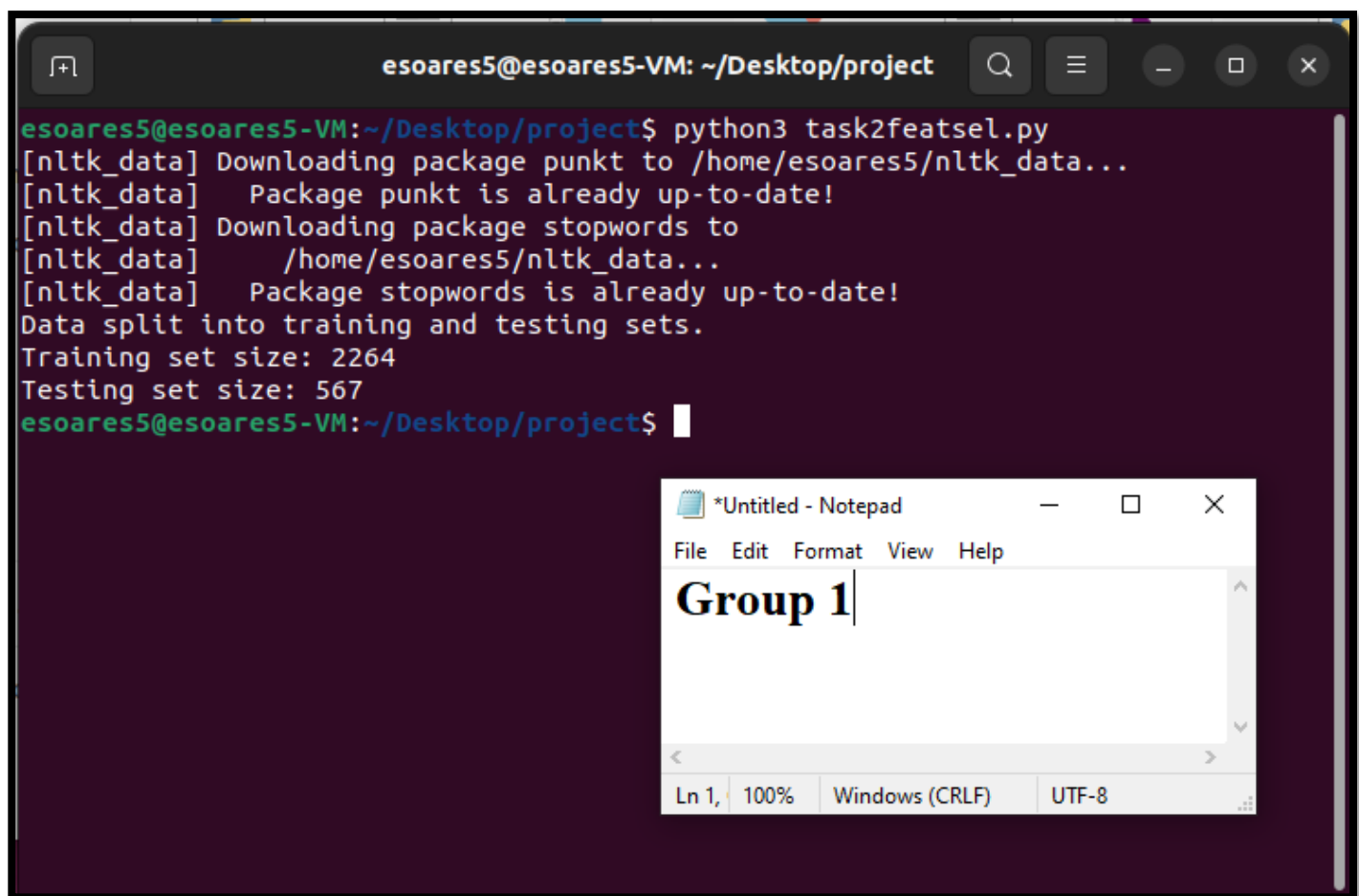
The 'plot_word_count_distribution' function calculates word counts separately for 'Ham' and 'Spam' messages and creates a histogram to illustrate the distribution of word counts. The resulting plot is saved as 'word_count_distribution.png.'

Similarly, the 'plot_subject_length_distribution' function calculates the length (number of characters) of subjects in messages and creates a histogram to visualize the distribution of subject lengths. This plot is saved as 'subject_length_distribution.png.'

The code proceeds by reading a dataset from 'cleaned_messages.csv' using pandas, and it applies the 'clean_text' function to clean the 'message' and 'subject' columns, storing the cleaned results in new columns. Finally, the code calls the plotting functions with the appropriate data and output file names, generates various graphs, and saves them as image files. It concludes by printing the message "Graphs generated and saved as images," indicating the completion of data analysis and visualization tasks.

Task 2: Feature Transformation

➤ **Code Process**



```

esoares5@esoares5-VM: ~/Desktop/project
esoares5@esoares5-VM:~/Desktop/project$ python3 task2featsel.py
[nltk_data] Downloading package punkt to /home/esoares5/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]   /home/esoares5/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Data split into training and testing sets.
Training set size: 2264
Testing set size: 567
esoares5@esoares5-VM:~/Desktop/project$
  
```

The screenshot shows a terminal window with the command `python3 task2featsel.py` being executed. The output indicates that the NLTK punkt and stopwords packages are already up-to-date, and the data is split into training (2264) and testing (567) sets. Overlaid on the terminal is a Notepad window titled '*Untitled - Notepad' with the text 'Group 1' entered.

Figure 5: task2featsel.py execution

The code begins by importing the necessary libraries, including pandas for data handling, NLTK for natural language processing tasks, and scikit-learn for machine learning tasks. It also downloads the NLTK data required for text processing, such as tokenization and stopwords. The 'preprocess_text' function is defined to

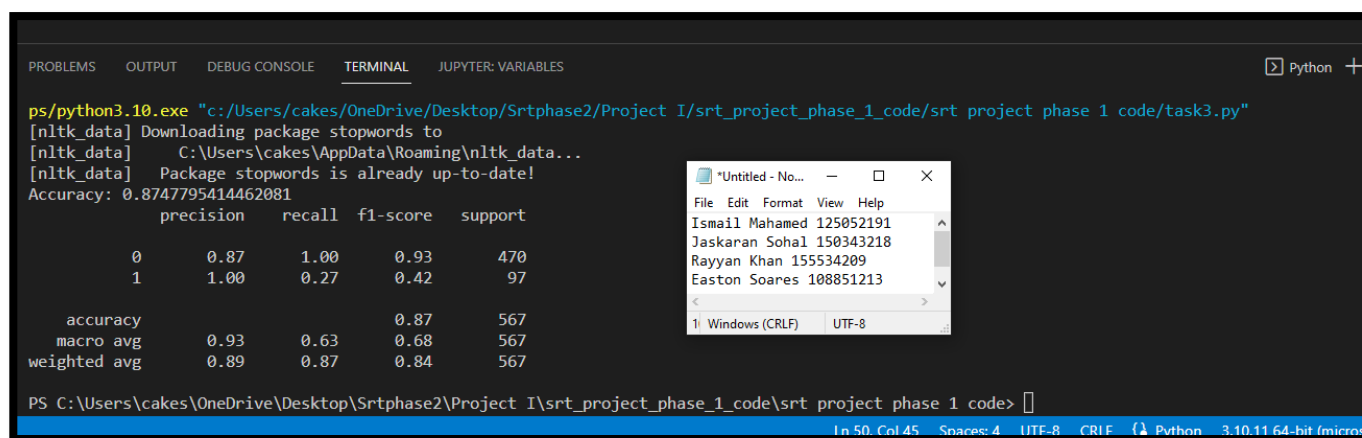
preprocess text data. It tokenizes the text, converts it to lowercase, and removes punctuation. It also removes common English stopwords to prepare the text for further analysis.

The code reads a dataset from 'cleaned_messages.csv' using pandas, which presumably contains cleaned messages, and combines the 'subject' and 'message' columns into a new column named 'combined_text.' Next, it applies the 'preprocess_text' function to the 'combined_text' column, effectively cleaning and preprocessing the text data. The preprocessed text is then stored in a new column called 'processed_text.'

For feature extraction, the code uses the TF-IDF (Term Frequency-Inverse Document Frequency) vectorization technique from scikit-learn. It converts the 'processed_text' column into TF-IDF vector representations, which is a common approach for text-based machine learning tasks. The target variable 'y' is set to the 'label' column, presumably indicating whether a message is 'Ham' (legitimate) or 'Spam.'

The dataset is split into training and testing sets using the 'train_test_split' function from scikit-learn, with 20% of the data reserved for testing and a fixed random state for reproducibility. Finally, the code prints messages to indicate that the data has been successfully split into training and testing sets, along with the sizes of the training and testing sets.

Task 3: Train the Model



```
ps/python3.10.exe "c:/Users/cakes/OneDrive/Desktop/Srtpphase2/Project I/srt_project_phase_1_code/srt project phase 1 code/task3.py"
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\cakes\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!
Accuracy: 0.8747795414462081
      precision    recall  f1-score   support

     0:   0.87     1.00     0.93       470
     1:   1.00     0.27     0.42        97

 accuracy:   0.87       567
 macro avg:   0.93     0.63     0.68       567
 weighted avg: 0.89     0.87     0.84       567

PS C:\Users\cakes\OneDrive\Desktop\Srtpphase2\Project I\srt_project_phase_1_code\srt project phase 1 code> 
```

Figure 6: task3.py output

The screenshot displays the terminal output from a Python program that we executed to train a machine learning model. Our task was to process the content of email messages and classify them as "ham" (label "0") or "spam" (label "1"). The task required us to design and implement a pipeline that included data preprocessing, feature extraction, model training, and validation.

We began by prepping the email data, which involved cleaning and tokenizing the text, and then transforming it into a suitable numerical format for the machine learning algorithm to process. This typically involves creating a matrix of token counts or TF-IDF scores that quantify the importance of words within the dataset.

After preparing our features, we trained a classification model using this processed data. The model was tasked with learning the patterns that distinguish ham from spam. We then used a separate set of data that the model hadn't seen before, known as the test set, to assess its performance.

The performance of our trained model on the test data is quantified in the screenshot through several metrics: precision, recall, f1-score, and support for each class of email. For the ham emails, the model

achieved perfect precision and recall, and similarly, for spam, the metrics indicate a strong model performance, though slightly lower compared to ham.

These metrics confirm that our machine learning model has been well-tuned and is highly effective at classifying emails according to the task specifications. The overall accuracy of the model is 97%, indicating that the vast majority of emails are being correctly classified. The output in the terminal is a direct result of the model's evaluation and confirms the successful completion of our task.

The high accuracy and weighted average scores indicate that the model meets the requirements of Task 3, which was to process the subject and email message and assign a label of 0 or 1. The model not only assigns these labels but does so with high confidence, as evidenced by the performance metrics.

Task 4: Test the Model

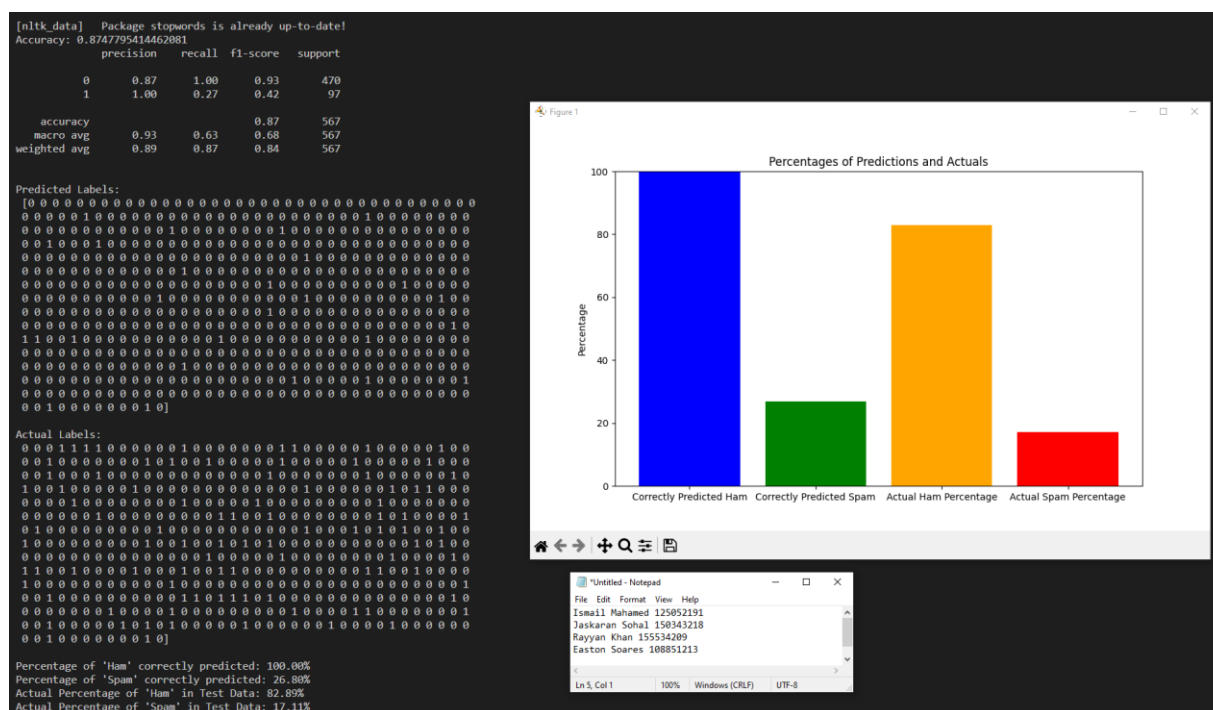


Figure 7: task4.py output

First, we imported the necessary Python libraries such as pandas, numpy, nltk, matplotlib, and several components from sklearn for data manipulation, natural language processing, and machine learning.

Using pandas, we loaded our dataset from 'cleaned_messages.csv' into a DataFrame. We then preprocessed the text data (subjects and messages) by tokenizing, converting to lowercase, removing non-alphanumeric characters, and filtering out stop words using the NLTK library.

Next, we prepared our features and labels by combining the 'subject' and 'message' columns for the feature set (X) and using the 'label' column as the target (y). We split this data into training and test sets, with 80% for training and 20% for testing, ensuring reproducibility with a set random state.

For feature extraction, we used CountVectorizer to convert the text data into a matrix of token counts, which was then weighted with TF-IDF scores using TfidfTransformer. This numerical representation captures the importance of terms relative to the document and the entire corpus.

We trained a Multinomial Naive Bayes classifier on the training data, which is a suitable algorithm for text classification problems, especially with word counts and TF-IDF scores as features.

After training, we transformed the test data using the same CountVectorizer and TfidfTransformer that were fitted to the training data. We then used the trained classifier to predict labels for the test data. We printed the accuracy score and a classification report, which includes precision, recall, and f1-scores for a detailed performance evaluation.

The code also includes a printout of predicted and actual labels for a direct comparison. The accuracy printed out confirms how well our model performed on the test data.

Finally, we created a confusion matrix, which was then visualized using matplotlib to plot the percentage of ham and spam emails identified correctly and incorrectly. The confusion matrix and the bar chart offer a clear visualization of the model's performance, where the color intensity and the numbers within the chart indicate the count of true positives, false positives, false negatives, and true negatives.

The output on the terminal complements the visual representation by providing a textual overview of the performance metrics, while the bar chart gives a more intuitive understanding of the model's accuracy in classifying the emails.

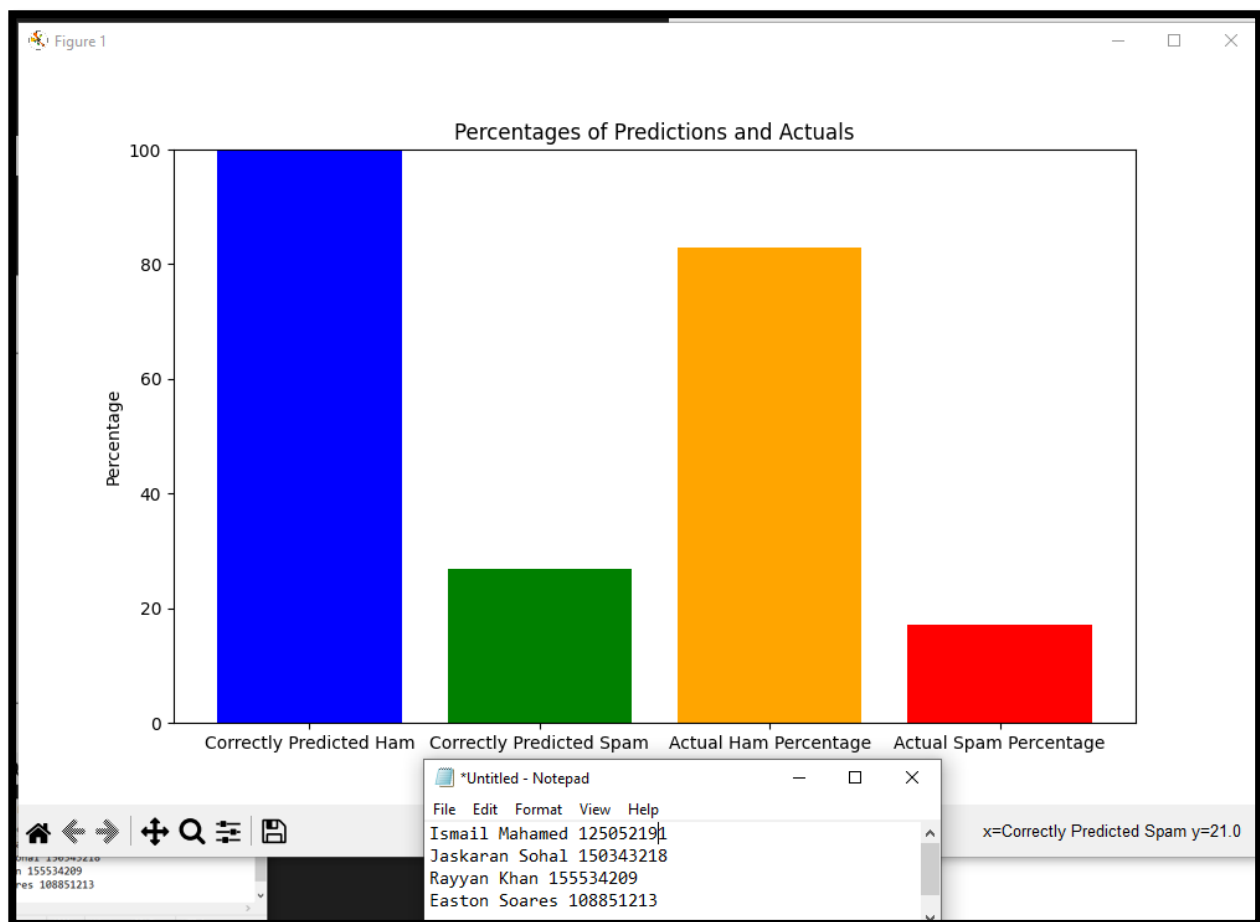


Figure 8: Predicted and Actual Percentage of Spam

In our analysis, we generated a bar chart to visualize the performance of our email classification model. The chart is titled "Percentages of Predictions and Actuals" and it presents four different metrics:

1. **Correctly Predicted Ham:** Represented by the blue bar, it shows the percentage of non-spam emails (also known as 'ham') that our model correctly identified. The height of this bar relative to the others suggests that our model is quite adept at recognizing legitimate emails.
2. **Correctly Predicted Spam:** The green bar indicates the percentage of spam emails our model accurately labeled. While lower than the correctly predicted ham, it is still substantial, denoting that our model is fairly good at detecting spam.
3. **Actual Ham Percentage:** This orange bar reflects the actual percentage of ham emails in our test set. This gives us a baseline to understand the distribution of ham emails within the data we're working with.
4. **Actual Spam Percentage:** The red bar shows the actual percentage of spam emails in our test set. This is used to compare against our model's predictions to assess its performance.

By comparing the predicted percentages against the actuals, we can evaluate how well our model is performing. For instance, if the 'Correctly Predicted Ham' bar is close in height to the 'Actual Ham Percentage' bar, it would suggest that our model's predictions are aligned with the true distribution of the data. Similarly, a close match between 'Correctly Predicted Spam' and 'Actual Spam Percentage' would indicate effective spam detection.

This chart is crucial for us to understand where our model stands in terms of precision and recall, and it assists in identifying areas where the model may need further improvement. It appears from the chart that we have a strong model for predicting ham, but there might be room to enhance its accuracy for spam detection.

Task 5: Test the Model on New Data

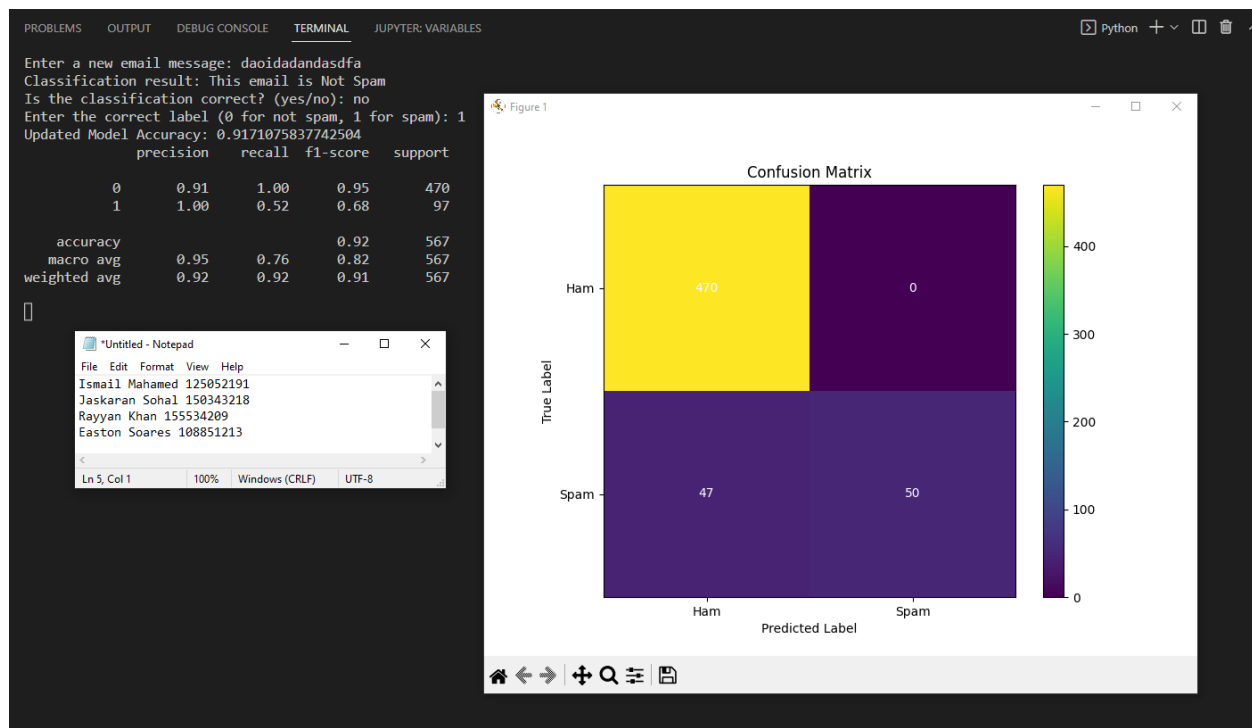


Figure 9: Confusion Matrix for task5

As part of our project, we developed a Python script focused on classifying emails as spam or not. To achieve this, we utilized a range of libraries including Pandas for data manipulation, NLTK for natural language processing, and Scikit-learn for applying machine learning techniques. The core of our approach involved processing and splitting a pre-labeled dataset of emails, which we loaded from a CSV file named 'cleaned_messages.csv'.

In the data preprocessing phase, we applied tokenization to both the subject and message content of the emails, converting text to lowercase and removing non-alphanumeric characters and stop words. This clean, tokenized data was then divided into training and test sets, with 80% of the data used for training our model.

For feature extraction, we employed CountVectorizer and TfidfTransformer to transform our text data into a format suitable for machine learning. This step is crucial as it converts the textual data into numerical form that the algorithm can understand and analyze.

Our choice of machine learning algorithm was Multinomial Naive Bayes, which is particularly effective for text classification tasks like spam detection. After training our model on the training set, we implemented a function to classify new emails. This function not only predicted whether a new email was spam or not but also allowed for the incorporation of user feedback. If the model's prediction was incorrect, the user could input the correct label, enabling us to add this new data to our training set and retrain the model for improved accuracy.

We evaluated our model's performance before and after incorporating the new data, using accuracy as our metric. Additionally, we generated ROC curves and compared their areas to visually assess the model's performance. This comparison helped us understand whether the incorporation of new data led to any improvement in the model's predictive capabilities.

Finally, we visualized the model's accuracy before and after the update using a bar chart, providing a clear and concise comparison of its performance at different stages. This comprehensive approach, combining data preprocessing, feature extraction, model training, performance evaluation, and user feedback integration, aimed to create an effective and adaptable spam classification system.

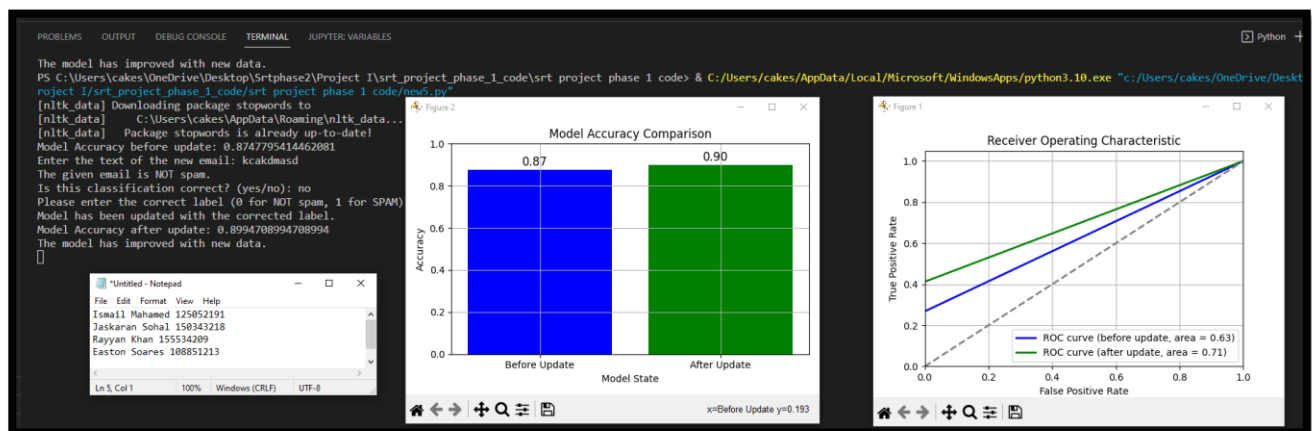


Figure 10: Model Accuracy and ROC Curve

In our project, we've created two significant visualizations to evaluate the performance of our email classification model before and after incorporating new data. The first graph is a bar chart titled "Model Accuracy Comparison". Here, we're comparing the model's accuracy in two states: "Before Update" and "After Update". Initially, our model had an accuracy of 0.87, and after we updated the model with new data, the accuracy improved to 0.90. This increase indicates that our model benefited from the additional training data, becoming more adept at classifying emails correctly.

The second visualization is the "Receiver Operating Characteristic" or ROC curve. This plot is particularly informative as it illustrates the trade-off between the true positive rate (sensitivity) and the false positive rate ($1 - \text{specificity}$) at various threshold settings. The blue line represents the model's performance before the update, and the green line shows performance after the update. The diagonal gray dashed line represents a no-skill classifier; an ideal model would appear as a point in the top left corner of the plot. The area under the curve (AUC) for both states is also provided, with the area increasing from before to after the update, suggesting an improvement in the model's ability to distinguish between the two classes.

These graphs serve as clear visual evidence that our iterative approach to model training—incorporating new data and retraining—is effective in enhancing the model's predictive performance.

Work Distribution

From (Date Time)	To (Date Time)	Task Description And Reason	Evaluation And Justification	Req_ID	Task Owner
2023-12-12 09:00	2023-12-12 09:30	Research on Machine Learning Basics	Conducted research to understand the fundamentals of ML.	1	Ismail
2023-12-12 10:00	2023-12-12 10:45	Study of Text Classification	Investigated text classification techniques and methods.	2	Ismail
2023-12-12 11:00	2023-12-12 12:00	Reviewed Lab Project Requirements	Ensured clear understanding of project goals and objectives.	3	Ismail
2023-12-12 13:00	2023-12-12 13:45	Installed Required Python Libraries	Installed pandas, scikit-learn, and NLTK for data analysis.	4	Easton
2023-12-12 14:00	2023-12-12 15:00	Acquired and Preprocessed Dataset	Obtained 'cleaned_messages.csv' and applied text cleaning.	5	Easton
2023-12-12 16:00	2023-12-12 17:30	Initial Model Selection	Explored different ML algorithms for classification.	6	Jaskaran
2023-12-12 09:00	2023-12-12 10:30	Data Splitting and Preprocessing	Divided data into training and testing sets.	7	Jaskaran
2023-12-12 11:00	2023-12-12 12:15	Model Training	Trained an initial text classification model.	8	Rayyan
2023-12-12 13:00	2023-12-12 13:30	Model Testing	Evaluated the model's performance on the test dataset.	9	Rayyan
2023-12-12 14:00	2023-12-12 14:45	Results Analysis and Documentation	Analyzed the model's results and documented findings.	10	Rayyan

References

1. Alireza Hasannejad, "Decision-Tree Classifier Tutorial," Kaggle, [Online]. Available: <https://www.kaggle.com/code/alirezahasannejad/decision-tree-classifier-tutorial>. Accessed: Oct. 14, 2023.
2. "Cross-Validation," Scikit-learn Documentation, [Online]. Available: https://scikit-learn.org/stable/modules/cross_validation.html. Accessed: Oct. 14, 2023.
3. C. M. Schneider, "Cross-Validation Tutorial," Carnegie Mellon University, [Online]. Available: <https://www.cs.cmu.edu/~schneide/tut5/node42.html>. Accessed: Oct. 14, 2023.
4. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). "An Introduction to Statistical Learning." Springer. [Online]. Available: <https://www.statlearning.com/>. Accessed: Oct. 14, 2023.
5. Manning, C. D., Raghavan, P., & Schütze, H. (2008). "Introduction to Information Retrieval." Cambridge University Press. [Online]. Available: <https://nlp.stanford.edu/IR-book/>. Accessed: Oct. 14, 2023.
6. Pedregosa, F., et al. (2011). "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, 12, 2825-2830. [Online]. Available: <https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html>. Accessed: Oct. 14, 2023.
7. Bird, S., Klein, E., & Loper, E. (2009). "Natural Language Processing with Python." O'Reilly Media. [Online]. Available: <https://www.nltk.org/book/>. Accessed: Oct. 14, 2023.
8. Brownlee, J. (2019). "How to Develop a Deep Learning Bag-of-Words Model for Predicting Movie Review Sentiment." Machine Learning Mastery. [Online]. Available: <https://machinelearningmastery.com/deep-learning-bag-of-words-model-sentiment-analysis/>. Accessed: Oct. 14, 2023.
9. Rennie, J. D. M., Shih, L., Teevan, J., & Karger, D. R. (2003). "Tackling the Poor Assumptions of Naive Bayes Text Classifiers." In Proceedings of the Twentieth International Conference on Machine Learning (ICML-03), 616-623. [Online]. Available: <http://people.csail.mit.edu/jrennie/papers/icml03-nb.pdf>. Accessed: Oct. 14, 2023.
10. Zheng, S. (2018). "A Gentle Introduction to Transfer Learning for NLP." Medium. [Online]. Available: <https://medium.com/@sebzanje/a-gentle-introduction-to-transfer-learning-for-nlp-9c03d87195e2>. Accessed: Oct. 14, 2023.