Name: Neha Tendulkar

Task: Task 3.3

- Preprocessing:

- Lowercasing: Convert the acquired data into lowercase for uniformed comparison. This technique comes in handy Tf-Idf for making features out of our natural language data.
- Removal of punctuations: Technique of text pre-processing where we try to remove unnecessary punctuation symbols because their presence does not make any significance in our text data.
- Removing Stop Words: These are the words that occur very frequently in our text data, but they are of no use. Many libraries have compiled stop words for various languages and we can use them directly and for any specific use case if we feel we can also add a more specific set of stop words to the list.
- Stemming: The main aim for stemming is that we can reduce the vocab size before inputting it into any machine learning model.
- Removing Numbers: Remove any numerical digits since they might not contribute much to spam detection.
- Emoji Removal: Remove emojis, symbols, and non-textual content.

- Feature Extraction:
  Transform the preprocessed text into numerical features that can be used for classification using TF-IDF representation. This involves converting each document into a vector that represents the frequency or importance of words in the document. TF gives us information on how often a term appears in the given data and IDF gives us information about the relative rarity of a term in the collection of this data. By multiplying these values together we can get our final TF-IDF value. The higher the TF-IDF score the more important or relevant the term is; as a

term gets less relevant, its TF-IDF score will approach 0. The data first needs to be converted to a vector of numerical data by a process known as vectorization. TF-IDF vectorization involves calculating the TF-IDF score for every word in your corpus relative to that document and then putting that information into a vector .Thus each document in the corpus would have its own vector, and the vector would have a TF-IDF score for every single word in the entire collection of data.

- Method of Classification:

Using a machine learning algorithm, such as a Naive Bayes classifier, Support Vector Machine (SVM), or a more advanced technique like a Random Forest or even a deep learning model like a Recurrent Neural Network (RNN).

The filter used will be Naïve Bayes because:

- Simplicity: Naive Bayes is a relatively simple algorithm, making it quick to train and efficient for processing large volumes of emails.
- Text Classification: Naive Bayes is particularly well-suited for text classification tasks, making it a strong choice for analyzing the content of emails, identifying spam keywords, and classifying based on text features.
- Handling Imbalanced Data: Naive Bayes can handle imbalanced datasets (where there are significantly more ham emails than spam emails) relatively well.
- Interpretability: Naive Bayes models are highly interpretable, which can be beneficial for understanding why an email was classified as spam.

- Split Data: Split the data into training and testing sets to evaluate the model's performance.

- Evaluate the Model:
    - Use metrics like precision, recall, F1-score, and accuracy to assess the model's performance on the test data.

Precision measures the accuracy of positive spam. High precision indicates a low false positive rate.
Recall measures the model's ability to correctly identify spam. High recall indicates a low false negative rate.
The F1-Score is the harmonic mean of precision and recall and provides a balanced measure of the model's performance.

- Setting Threshold:
- Based on the model's output probabilities or scores, it can set a threshold to classify a text as spam or not spam. This threshold depends on the desired trade-off between false positives and false negatives.

- Possible Challenges Faced:
- Lack of Labelled Data: Obtaining a labelled dataset of spam and non-spam texts can be challenging. Building a high-quality labelled dataset is crucial for training a robust model.
- Evolution of Spam: Spam techniques evolve rapidly, and a model trained on one dataset may not perform well on newer types of spam. Regular model retraining is necessary.
- Class Imbalance: Spam messages are often much less common than non-spam messages, leading to class imbalance issues. Techniques like oversampling, undersampling, or using different evaluation metrics are needed.
- Contextual Understanding: Some spam messages may not contain typical spam keywords but could still be spam based on the context. Building models that can understand context is challenging.

- False Positives: Aggressive spam filters may flag legitimate messages as spam, causing inconvenience to users. Striking the right balance is essential.