Machine Learning Project on Text Spam Detection

INTRODUCTION

- Brief overview of the project's goals and objectives.

GOALS:

- **Developing Effective Spam Detection**: The primary goal is to design and implement machine learning models capable of accurately identifying spam messages from legitimate ones (ham) in various forms of communication such as emails, text messages, or social media posts in a text form.
- **Enhancing Cybersecurity Measures**: By effectively detecting and filtering out spam, the project aims to contribute to strengthening cybersecurity measures for individuals, organizations, and businesses, reducing the risks associated with malicious activities and potential security breaches.

OBJECTIVES:

- **Data Collection and Preprocessing**: Gather and preprocess a diverse dataset containing labeled examples of both spam and non-spam messages, ensuring data quality and integrity for training machine learning models.
- **Model Development**: Implement and train machine learning models, such as K-Nearest Neighbors (KNN) and Naive Bayes (NB), leveraging their ability to analyze text data and make predictions based on learned patterns.
- **Performance Evaluation**: Evaluate the performance of the developed models using appropriate metrics such as accuracy, precision, recall, and F1-score to assess their effectiveness in distinguishing between spam and legitimate messages.
- **Performance Improvement**: Explore techniques to enhance the performance of the models, including feature engineering, hyperparameter tuning, and model selection, aiming to achieve higher accuracy and precision in spam detection.
- **Documentation and Communication**: Document the entire process, including data collection, model development, evaluation, and performance improvement efforts, to facilitate transparency, reproducibility, and knowledge sharing within the cybersecurity community.

DATA COLLECTION:

Source of the Data:

The data for this project was sourced from a publicly available dataset (https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset) and supplemented with additional collected data to ensure diversity and representativeness.

- **Collected Data**: To enhance the diversity of the dataset and capture newer trends in spam messages, we collected additional data from various sources, including online forums, social media platforms, and real-world email accounts. This supplementary data ensured a comprehensive representation of spam messages across different communication channels.

Description of the Dataset:

- **Size**: The combined dataset comprises a total of 5572 samples, containing both spam and non-spam (ham) messages collected. The dataset was balanced to ensure an equal distribution of spam and non-spam messages, mitigating potential biases during model training.
- **Format**: Each sample in the dataset is represented as a text document containing the content of the message. For email datasets, the samples include email subjects and bodies, while for SMS datasets, the samples consist of the text of the messages. The dataset is organized into a structured format, stored in a CSV (Comma Separated Values) file. The first column contained information about the input message whether the message was either ham (not spam) or spam and the second column contained the actual message. There were three other empty columns in the input data which were later removed.

Preprocessing Steps Applied:

Before using the dataset for model training, several preprocessing steps were applied to ensure data quality and consistency:

1. **Label Encoding**: Ham converted are to 0 and spam are converted to 1 using label encoder class.
2. **Tokenization**: Breaking down each message into individual tokens (words or phrases) to facilitate further analysis.
3. **Normalization**: Converting text to lowercase and removing any unnecessary characters, punctuation, or special symbols to standardize the text representations.

4. **Stop Words Removal**: Eliminating common stop words (e.g., "and", "the", "is") that do not contribute significantly to the meaning of the messages.
5. **Stemming or Lemmatization**: Reducing words to their base or root form to consolidate variations of the same word and reduce feature dimensionality.
6. **Handling Missing Values**: Addressing any missing or null values in the dataset through imputation or removal, ensuring completeness of the data.

Exploratory Data Analysis (EDA):

Summary Statistics:

1. **Overall Dataset Statistics**:
    - Total number of samples (spam and non-spam).
    - Distribution of spam vs. non-spam messages.
    - Average length of messages (in terms of words or characters).
    - Frequency of unique words or phrases in the dataset.
2. **Class-wise Statistics**:
    - Mean, median, and standard deviation of message length for spam and non-spam messages.
    - Top frequent words or phrases in spam and non-spam messages.

Visualizations:

1. **Histograms**:
    - Distribution of message lengths for both spam and non-spam messages.
    - Histogram of word frequencies to visualize the most common words across the dataset.
2. **Word Clouds**:
    - Word clouds for spam and non-spam messages separately, showcasing the most frequent words.
    - Comparison word clouds to highlight differences in word usage between spam and non-spam messages.
3. **Bar Plots**:
    - Bar plot showing the distribution of message types (spam vs. non-spam).
    - Bar plot of the top N most frequent words in the dataset

Key Insights:

- Identification of frequent words or phrases unique to spam messages, which could serve as important features for model training.
- Understanding the distribution of message lengths and potential differences between spam and non-spam messages.
- Visualization of class imbalances, if any, and potential strategies for handling them during model training.
- Insights into the overall structure and composition of the dataset, informing decisions regarding feature engineering and model selection.

TF-IDF Vectorizer:

The TF-IDF (Term Frequency-Inverse Document Frequency) Vectorizer is a widely used technique in natural language processing (NLP) for converting a collection of text documents into numerical feature vectors. It's particularly useful in text mining and information retrieval tasks.

Term Frequency (TF):

- Term Frequency measures how frequently a term (word) appears in a document.
- It is calculated as the ratio of the number of times a term occurs in a document to the total number of terms in the document.
- The intuition behind TF is that terms that occur frequently in a document are important in describing the content of that document.

Inverse Document Frequency (IDF):

- Inverse Document Frequency measures how important a term is across the entire document collection.
- It is calculated as the logarithm of the ratio of the total number of documents to the number of documents containing the term.
- Terms that occur in many documents will have a lower IDF value, while terms that occur in few documents will have a higher IDF value.
- The intuition behind IDF is to reduce the weight of terms that are common across many documents, thus emphasizing terms that are more unique or specific to individual documents

The TF-IDF (Term Frequency-Inverse Document Frequency) Vectorizer is a widely used technique in natural language processing (NLP) for converting a collection of text documents into numerical feature vectors. It's particularly useful in text mining and information retrieval tasks. Let's break down how TF-IDF Vectorizer works:

TF-IDF Vectorizer:

- The TF-IDF Vectorizer combines the concepts of TF and IDF to generate a numerical representation (vector) for each document in the corpus.
- It calculates the TF-IDF score for each term in each document, where the TF-IDF score is the product of the TF and IDF values for that term.
- TF-IDF Vectorizer typically operates on preprocessed text data, where preprocessing steps may include tokenization, lowercasing, removing stop words, and stemming or lemmatization.
- The output of TF-IDF Vectorizer is a sparse matrix where each row represents a document, and each column represents a unique term in the entire corpus. The cell values correspond to the TF-IDF scores of the respective terms in the corresponding documents.
- TF-IDF Vectorizer can be customized with various parameters, such as the choice of TF and IDF calculation methods, the use of n-grams, and the handling of out-of-vocabulary terms.

MODEL SELECTION:

We used many models including Naïve Bayes, Support Vector Machine, KNN, Linear Regression, Decision Trees, Boosted Trees, Random Forest to train on our data and picked the models with highest accuracy and precision(KNN, Naïve Bayes).

	Algorithm	Accuracy	Precision
1	KN	0.900387	1.000000
2	NB	0.959381	1.000000
8	ETC	0.977756	0.991453
5	RF	0.970019	0.990826
0	SVC	0.972921	0.974138
6	AdaBoost	0.962282	0.954128
10	xgb	0.971954	0.950413
4	LR	0.951644	0.940000
9	GBDT	0.951644	0.931373
7	BgC	0.957447	0.861538
3	DT	0.935203	0.838095

MODEL IMPROVEMENT:

max_features(TF-IDF):

- The `max_features` parameter limits the number of features (terms) to use.
- Experiment with different values to control the dimensionality of the feature space.
- Using too many features may lead to overfitting, while too few features may result in underfitting.
- Changed the max_features parameter in TF-IDF to improve the model performance.

ENSEMBLE METHODS:

Voting and stacking classifiers are ensemble learning techniques that combine multiple individual classifiers to improve predictive performance. They are particularly useful when the individual classifiers have different strengths and weaknesses, as they can leverage the diversity of the base models to make more accurate predictions. Here's why voting and stacking classifiers might be used in a machine learning project:

1. **Improved Robustness**: Voting and stacking classifiers can help improve the robustness of predictions by reducing the variance associated with individual classifiers. By combining the predictions of multiple models, the ensemble model can provide more stable and reliable predictions, especially in cases

where individual models might make errors due to noise or variability in the data.

2. **Increased Accuracy**: Ensemble methods often achieve higher predictive accuracy compared to individual classifiers by leveraging the collective knowledge of multiple models. Voting classifiers, in particular, combine the predictions of multiple classifiers using a majority voting scheme, while stacking classifiers learn to combine the outputs of base models using a meta-learner, both aiming to improve overall prediction accuracy.
3. **Handling Different Types of Models**: Voting and stacking classifiers allow for the combination of different types of base models, including decision trees, support vector machines, logistic regression, etc. This flexibility enables leveraging the strengths of various algorithms and handling different aspects of the data effectively.
4. **Capturing Complementary Information**: Individual classifiers may excel in capturing different aspects or patterns present in the data. By combining the predictions of multiple models, ensemble methods can capture complementary information and exploit diverse perspectives, leading to better overall performance.
5. **Model Averaging**: Ensemble methods effectively perform model averaging, which can help mitigate the risk of overfitting and improve generalization performance. By aggregating the predictions of multiple models, ensemble methods can smooth out individual model biases and reduce the impact of outliers.
6. **Adaptability and Flexibility**: Voting and stacking classifiers are highly adaptable and flexible, allowing for easy integration of additional models or modifications to the ensemble structure. This adaptability makes them suitable for various machine learning tasks and datasets.

Overall, voting and stacking classifiers are powerful techniques for improving predictive performance and robustness in machine learning projects, making them valuable tools in situations where the goal is to maximize accuracy and mitigate the limitations of individual classifiers.

PICKLING OF MODEL:

- Using the `pickle` module to store machine learning models is a common practice in Python. It allows you to save trained models to disk and load them later for making predictions or further training.
- It's essential to save the trained model along with any preprocessing steps or feature transformations to ensure consistency when loading the model later.
- While `pickle` is convenient for saving and loading models, it's important to be aware of potential security risks when unpickling data from untrusted sources.

Consider using alternative serialization libraries like `joblib` for large NumPy arrays or more secure formats like `HDF5` for complex models.

- `joblib` is another serialization library often used with scikit-learn models, especially when dealing with large numpy arrays. It offers better performance and is more efficient for storing large objects like NumPy arrays.