This document details the implementation of an SMS Spam Classifier model. Here is a detailed explanation of the entire approach, covering the various stages involved in building this.

**1. Data Loading and Initial Exploration:**

The process began by importing necessary Python libraries such as numpy, pandas, matplotlib.pyplot, and seaborn. The SMS spam dataset, located at /kaggle/input/sms-spam-collection-dataset/spam.csv, was loaded into a panda DataFrame using pd.read\_csv() with encoding='ISO-8859-1'.

Initial inspection of the DataFrame using df.head() and df.shape (revealing 5572 rows and 5 columns) provided a glimpse into the data's structure. df.info() showed data types and the presence of null values in unnamed columns. Descriptive statistics were generated using df.describe(). The distribution of the 'Category' column ('ham' or 'spam') was visualised with a pie chart, indicating an imbalance in the dataset with 'ham' being more frequent than 'spam'.

**2. Data Preprocessing:**

Several steps were taken to preprocess the data:

* **Dropping Unnecessary Columns:** Columns named 'Unnamed: 2', 'Unnamed: 3', and 'Unnamed: 4' were dropped from the DataFrame as they likely contained null values and were deemed unnecessary.
* **Renaming Columns:** The remaining columns were renamed to 'Category' and 'Message' for better understanding.
* **Label Encoding:** The categorical values in the 'Category' column ('ham', 'spam') were converted into numerical representations (0 and 1) using LabelEncoder from sklearn.preprocessing.
* **Removing Duplicates:** Duplicate rows were identified and removed using df.drop\_duplicates(keep='first'), resulting in a DataFrame with 5169 rows.
* **Checking for Missing Values:** A final check using df.isnull().sum() confirmed the absence of missing values after these steps.

**3. Feature Engineering:**

New features were engineered from the 'Message' column to extract potentially useful information:

* **num\_characters:** The total number of characters in each message was calculated using df['Message'].apply(len).
* **num\_words:** The number of words in each message was determined by tokenizing the message using nltk.word\_tokenize() and counting the tokens. This required importing nltk and downloading the punkt resource.
* **num\_sentences:** The estimated number of sentences in each message was calculated by splitting the message based on sentence-ending punctuation marks using nltk.sent\_tokenize().

Histograms were used to visualise the distributions of these new numerical features, separated by 'Category' (ham/spam). Pair plots were generated using seaborn.pairplot() to explore the relationships between these numerical features and the 'Category'.

**4. Text Preprocessing (Further):**

The 'Message' column underwent more detailed text preprocessing to prepare it for machine learning models:

* **Lowercasing:** All text was converted to lowercase.
* **Tokenization:** Messages were split into individual words (tokens) using nltk.word\_tokenize().
* **Removing Special Characters:** Punctuation and special symbols were removed from the tokens using string.punctuation.
* **Removing Stop Words:** Common English stop words were removed using the stopwords corpus from nltk.corpus.
* **Stemming:** Words were reduced to their root form using the PorterStemmer from nltk.stem.porter.

A function named transform\_text(text) encapsulated all these preprocessing steps. This function was then applied to the 'Message' column to create a new column called transformed\_text containing the preprocessed messages.

**5. Model Selection and Algorithms Used:**

A variety of machine learning models were considered for the spam classification task:

* Logistic Regression (LogisticRegression)
* Support Vector Machine (SVM) with a linear kernel (SVC)
* Naive Bayes classifiers: GaussianNB, MultinomialNB, BernoulliNB
* Decision Tree (DecisionTreeClassifier)
* Random Forest (RandomForestClassifier)
* AdaBoost (AdaBoostClassifier)
* Gradient Boosting (GradientBoostingClassifier)
* XGBoost (XGBClassifier)

**6. Feature Vectorization:**

The text data in the transformed\_text column was converted into a numerical format using the **Bag-of-Words (BoW)** technique implemented by CountVectorizer from sklearn.feature\_extraction.text. The CountVectorizer was configured to consider a maximum of 3000 of the most frequent words as features (max\_features=3000). The dataset was split into training and testing sets using train\_test\_split from sklearn.model\_selection, with a test size of 0.25 (test\_size=0.25) and a random\_state of 42 for reproducibility. The CountVectorizer was fitted on the training data, and then both the training and testing text data were transformed into numerical vectors.

**7. Model Training and Evaluation:**

Each of the selected models was trained on the training data using the vectorized text features (X\_train) and the encoded 'Category' labels (y\_train). The performance of each trained model was evaluated on both the training and testing datasets (X\_test, y\_test) using the following metrics:

* Accuracy (accuracy\_score)
* Precision (precision\_score)
* Recall (recall\_score)
* F1-score (f1\_score)
* ROC AUC score (roc\_auc\_score)

The evaluation results for each model on both the training and test sets were printed to allow for comparison.

**8. Special Methods and Observations:**

Hyperparameter tuning was performed for several models, including Logistic Regression, SVM, Decision Tree, Random Forest, AdaBoost, and XGBoost, using RandomizedSearchCV from sklearn.model\_selection. This aimed to find the optimal hyperparameters for each model to potentially improve performance.

The initial evaluation results indicated that models like Logistic Regression, Linear SVC, Multinomial Naive Bayes, Bernoulli Naive Bayes, and Random Forest achieved high performance on the training data. On the test data, **Logistic Regression** and **XGBoost** demonstrated strong performance across all evaluation metrics.

**9. Final Model Training and Saving:**

Based on the evaluation, final versions of the **Logistic Regression** and **XGBoost** models were trained on the entire training dataset using specific hyperparameter settings obtained through hyperparameter tuning. These trained models (logreg\_model, xgb\_model), along with the fitted CountVectorizer (cv), were saved using joblib for potential future use or deployment.

**10. ROC Curve Analysis:**

A **Receiver Operating Characteristic (ROC) curve** was plotted for the Logistic Regression and XGBoost models. This visualized the trade-off between the true positive rate (Sensitivity) and the false positive rate (1-Specificity) at different classification thresholds. The Area Under the Curve (AUC) scores for both models were also calculated and displayed on the plot. The ROC AUC scores obtained on the test data were **0.99 for Logistic Regression** and **0.98 for XGBoost**, indicating excellent ability to distinguish between spam and ham messages.

**11. Final Observations:**

The document concludes that both Logistic Regression and XGBoost models performed well in classifying SMS spam messages, with Logistic Regression showing a slightly better ROC AUC score on the test data. The trained models and the vectorizer were saved, making the spam classifier readily deployable.