SMS Spam Classification Model Documentation

Objective:

The goal of this project is to develop an SMS classification model capable of distinguishing between **spam** and **non-spam** (ham) messages. The dataset consists of labeled messages, where each message is tagged as either "spam" or "ham." The model will be trained to identify patterns in the messages and predict whether new, unseen messages are spam.

Requirements:

- **Python 3.x**
- **Libraries**:
  - `pandas`
  - `sklearn`
  - `nltk`
  - `numpy`
- **Dataset**: `spam.csv` (or similar), which contains two columns: the message text (`v2`) and the label (`v1`).

Code Structure:

1. Import Required Libraries:

The following libraries are imported to handle data preprocessing, model training, and evaluation:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from nltk.corpus import stopwords
import nltk
```

2. Data Preprocessing:

The data is preprocessed to remove unnecessary words (i.e., stopwords) from the SMS messages.

```
# Download NLTK stopwords
nltk.download('stopwords')

# Load dataset
df = pd.read_csv('/content/spam.csv', encoding='latin-1')  # Load CSV data
print(df.columns)  # Check column names

# Map the label to binary values
df[label_column_name] = df[label_column_name].map({'ham': 0, 'spam': 1})

# Remove stopwords
stop_words = set(stopwords.words('english'))
df[message_column_name] = df[message_column_name].apply(lambda x: ' '.join([word for word in x.split() if word.lower() not in stop_words]))
```

3. Feature Extraction:

`TfidfVectorizer` is used to convert the text messages into numerical features (TF-IDF values).

```
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df[message_column_name])  # Transform messages into feature vectors
y = df[label_column_name]  # Target labels (spam = 1, ham = 0)
```

4. Train-Test Split:

The dataset is split into a training set (80%) and a testing set (20%) for model evaluation.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

5. Model Training and Evaluation:

The code trains and evaluates three different models: **Naive Bayes**, **Logistic Regression**, and **Support Vector Machine (SVM)**.

Naive Bayes Classifier:

```
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)
y_pred_nb = nb_classifier.predict(X_test)

print("Naive Bayes Classifier:")
print("Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Precision:", precision_score(y_test, y_pred_nb, pos_label='spam'))
print("Recall:", recall_score(y_test, y_pred_nb, pos_label='spam'))
print("F1 Score:", f1_score(y_test, y_pred_nb, pos_label='spam'))
```

Logistic Regression Classifier:

```
lr_classifier = LogisticRegression(max_iter=1000)
lr_classifier.fit(X_train, y_train)
y_pred_lr = lr_classifier.predict(X_test)

print("\nLogistic Regression Classifier:")
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("Precision:", precision_score(y_test, y_pred_lr, pos_label='spam'))
print("Recall:", recall_score(y_test, y_pred_lr, pos_label='spam'))
print("F1 Score:", f1_score(y_test, y_pred_lr, pos_label='spam'))
```

Support Vector Machine Classifier:

```
svm_classifier = SVC()
svm_classifier.fit(X_train, y_train)
y_pred_svm = svm_classifier.predict(X_test)

print("\nSupport Vector Machine Classifier:")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Precision:", precision_score(y_test, y_pred_svm, pos_label='spam'))
print("Recall:", recall_score(y_test, y_pred_svm, pos_label='spam'))
print("F1 Score:", f1_score(y_test, y_pred_svm, pos_label='spam'))
```

Improvements and Future Work:

1. Cross-Validation:

Using cross-validation helps estimate the model's performance more reliably by training and testing the model multiple times on different subsets of the data.

```
from sklearn.model_selection import cross_val_score
nb_cv_scores = cross_val_score(nb_classifier, X, y, cv=5, scoring='accuracy')
print("Naive Bayes Cross-Validation Accuracy: ", nb_cv_scores.mean())
```

2. Hyperparameter Tuning:

Use techniques like **GridSearchCV** or **RandomizedSearchCV** to optimize hyperparameters for each classifier.

3. Dealing with Imbalanced Classes:

If the dataset has an imbalanced class distribution, consider using:

- **Class weights**: `class_weight='balanced'` for models like Logistic Regression and SVM.
- **Oversampling**: Use techniques like **SMOTE** (Synthetic Minority Over-sampling Technique) to balance the dataset.

4. Feature Engineering:

You can experiment with:

- **N-grams**: Try using bigrams or trigrams (`ngram_range=(1, 2)`) to capture more context in the messages.
- **Advanced text preprocessing**: Explore lemmatization or stemming to reduce words to their base form.

Conclusion:

The provided code successfully trains and evaluates multiple classifiers for the SMS spam classification task, achieving useful metrics like accuracy, precision, recall, and F1-score. To further enhance the model’s performance, consider implementing cross-validation, tuning hyperparameters, addressing class imbalance, and experimenting with different text preprocessing techniques.
