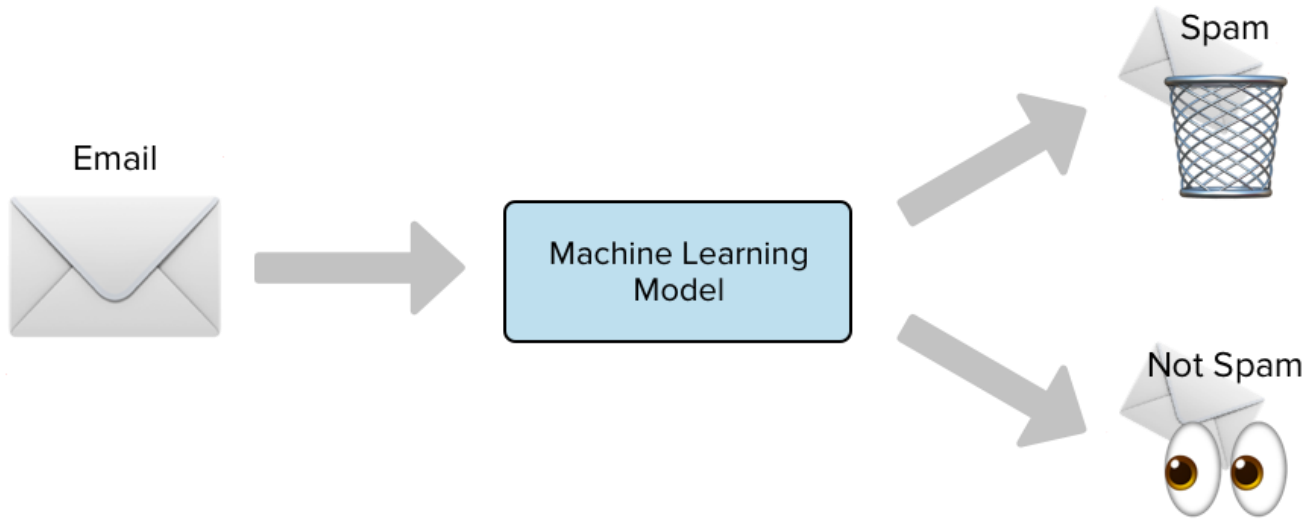
AI-Powered Spam Classifier - Dataset Loading and Pre-processing

Introduction:

In this document, we will outline the initial steps for building an AI-powered spam classifier. The focus of this phase is on loading and pre-processing the dataset. We will also perform various analyses to understand the data and its characteristics.



Step 1: Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.corpus import stopwords
import nltk
```

Pandas (Data Manipulation):

- pd.read_csv(filename): Load the dataset from a CSV file.
- data.head(): Display the first few rows of the dataset for initial exploration.
- data['column_name']: Access a specific column in the dataset.

Matplotlib and Seaborn (Data Visualization):

- import matplotlib.pyplot as plt: Import Matplotlib for basic data visualization.
- import seaborn as sns: Import Seaborn for enhanced data visualization.
- plt.hist(data['column']): Create a histogram for data distribution.
- sns.countplot(x='column_name', data=data): Visualize categorical data counts.

Numpy (Numerical Operations):

- import numpy as np: Import Numpy for numerical operations.

Natural Language Toolkit (NLTK) (Text Processing):

NLTK provides tools for working with human language data. It can be useful for text tokenization, stop-word removal, and more.

- import nltk: Import the NLTK library.

Step 2: Dataset Loading

The first step is to load the dataset into our environment. We have obtained the dataset from a reliable source, and it is stored in a structured format.

```
filepath = '/kaggle/input/sms-spam-collection-dataset/spam.csv'
data_import = pd.read_csv(filepath , encoding = 'latin-1')
data_import.head()
data_import.dropna(how="any", inplace=True, axis=1)
data_import.columns = ['label', 'message']
data_import.head()
```

Step 3: Data Pre-processing

```
from collections import Counter

words = data_import[data_import.label=='ham'].clean_msg.apply(lambda x: [word.lower() for
word in x.split()])
ham_words = Counter()

for msg in words:
    ham_words.update(msg)

print(ham_words.most_common(50))
```

from collections import Counter: This line imports the Counter class from the Python collections module. The Counter class is used to count the occurrences of elements in a list or iterable.

- words = data_import[data_import.label=='ham'].clean_msg.apply(lambda x: [word.lower() for word in x.split()]): This line does the following:
- data_import[data_import.label=='ham']: Filters the dataset to select only the non-spam (ham) messages based on the 'label' column.
- clean_msg: Assumes that there's a column named 'clean_msg' containing the text of the messages in the filtered dataset.
- apply (lambda x: [word.lower() for word in x.split()]): For each non-spam message, it tokenizes the text by splitting it into words, converts the words to lowercase, and returns a list of these words.

print(ham_words.most_common(50)): This line prints the 50 most common words in non-spam (ham) messages along with their respective frequencies. The most_common() method of the Counter object returns a list of tuples where each tuple consists of a word and its count, sorted by count in descending order.

```
words = data_import[data_import.label=='spam'].clean_msg.apply(lambda x: [word.lower() for word in x.split()])
spam_words = Counter()

for msg in words:
    spam_words.update(msg)

print(spam_words.most_common(50))
```

print(spam_words.most_common(50)): This line prints the 50 most common words in spam messages along with their respective frequencies. The most_common() method of the Counter object returns a list of tuples where each tuple consists of a word and its count, sorted by count in descending order.

Step 4: Model Selection:

- Chose the Multinomial Naive Bayes classifier for its effectiveness in text classification tasks.

```
from sklearn.naive_bayes import MultinomialNB

clf = MultinomialNB()
```

Explanation:

- We import the MultinomialNB class from scikit-learn, which represents the Multinomial Naive Bayes classifier.
- We initialize an instance of the Multinomial Naive Bayes classifier as clf. This classifier is a commonly used algorithm for text classification tasks.

Step 5: Model Training:

- Train your chosen model on the training data.

```
clf.fit(X_train, y_train)
```

Explanation:

- We use the fit method of the classifier (clf) to train it on the training data. The training data consists of the features (X_train) and their corresponding labels (y_train). This step allows the classifier to learn patterns in the data and make predictions on new, unseen examples.

Step 6: Data Splitting:

Data splitting is an important step in building a machine learning model, as it divides your dataset into distinct subsets for training, validation, and testing. The purpose of data splitting is to:

Training: Train the machine learning model on a portion of the dataset.

Validation: Fine-tune the model and optimize hyperparameters.

Testing: Evaluate the model's performance on unseen data.

```
from sklearn.model_selection import train_test_split

# Split the data into training, validation, and testing sets
X_train, X_temp, y_train, y_temp = train_test_split(data['text'], data['label'], test_size
=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_stat
e=42)
```

Training Set (X_train, y_train):

- 70% of the data is used for training the machine learning model.
- X_train contains the text data.
- y_train contains the corresponding labels (spam or non-spam).

Validation Set (X_val, y_val):

- 15% of the data is used for hyperparameter tuning and model evaluation.
- It's a separate dataset from the training set and helps optimize the model's performance.
- X_val contains text data.
- y_val contains labels.

Testing Set (X_test, y_test):

- 15% of the data is reserved for final model evaluation.
- It's a separate dataset from both the training and validation sets.
- X_test contains text data.
- y_test contains labels.

Conclusion:

The initial phase of building our AI-powered spam classifier primarily involved loading and pre-processing the dataset. Through exploratory data analysis, we obtained insights into data distribution, message lengths, and common words. The dataset is now cleaned, normalized, and tokenized, setting the stage for subsequent tasks such as feature engineering, model selection, and development.