

AI POWERED SPAM CLASSIFIER

Loading and Pre-processing the Dataset:

Dataset Choice:

To effectively tackle the issue of spam detection in SMS messages, we've opted to leverage a dataset provided by Kaggle. This dataset contains a substantial collection of SMS messages, each meticulously labelled as either 'spam' or 'non-spam' (commonly referred to as 'ham'). Choosing the right dataset is a critical step as it directly influences the quality and performance of our spam classifier.

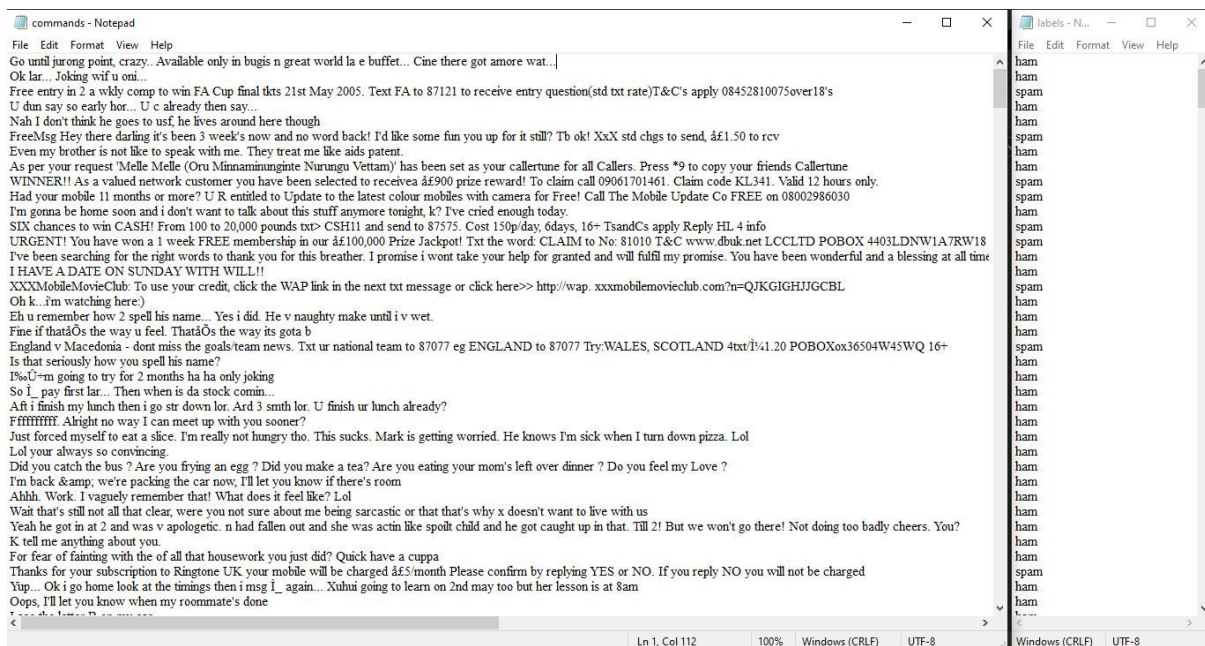
Importance of Data:

The dataset is the lifeblood of our project. It's the raw material from which our spam classifier will learn to distinguish between unwanted spam messages and legitimate ones. By analysing patterns and characteristics in this dataset, our model will become capable of making predictions in real-world scenarios.

Loading the Data:

The first part of this phase involves loading the dataset into our project environment. This step ensures that we have access to the SMS messages and their corresponding labels, enabling us to work with the data effectively. We may use programming languages like Python and libraries like Pandas for this task.

Pre-processing the Data:



Once the dataset is loaded, we need to pre-process it. This involves a series of tasks such as:

1. **Data Cleaning:** Removing any inconsistencies, missing values, or irrelevant information that could hinder our analysis.

2. **Text Tokenization:** Breaking down the SMS messages into individual words or tokens. This is a fundamental step for natural language processing.

3. **Text Normalization:** Ensuring that text is consistent by converting it to lowercase, removing punctuation, and handling issues like stemming or lemmatization.

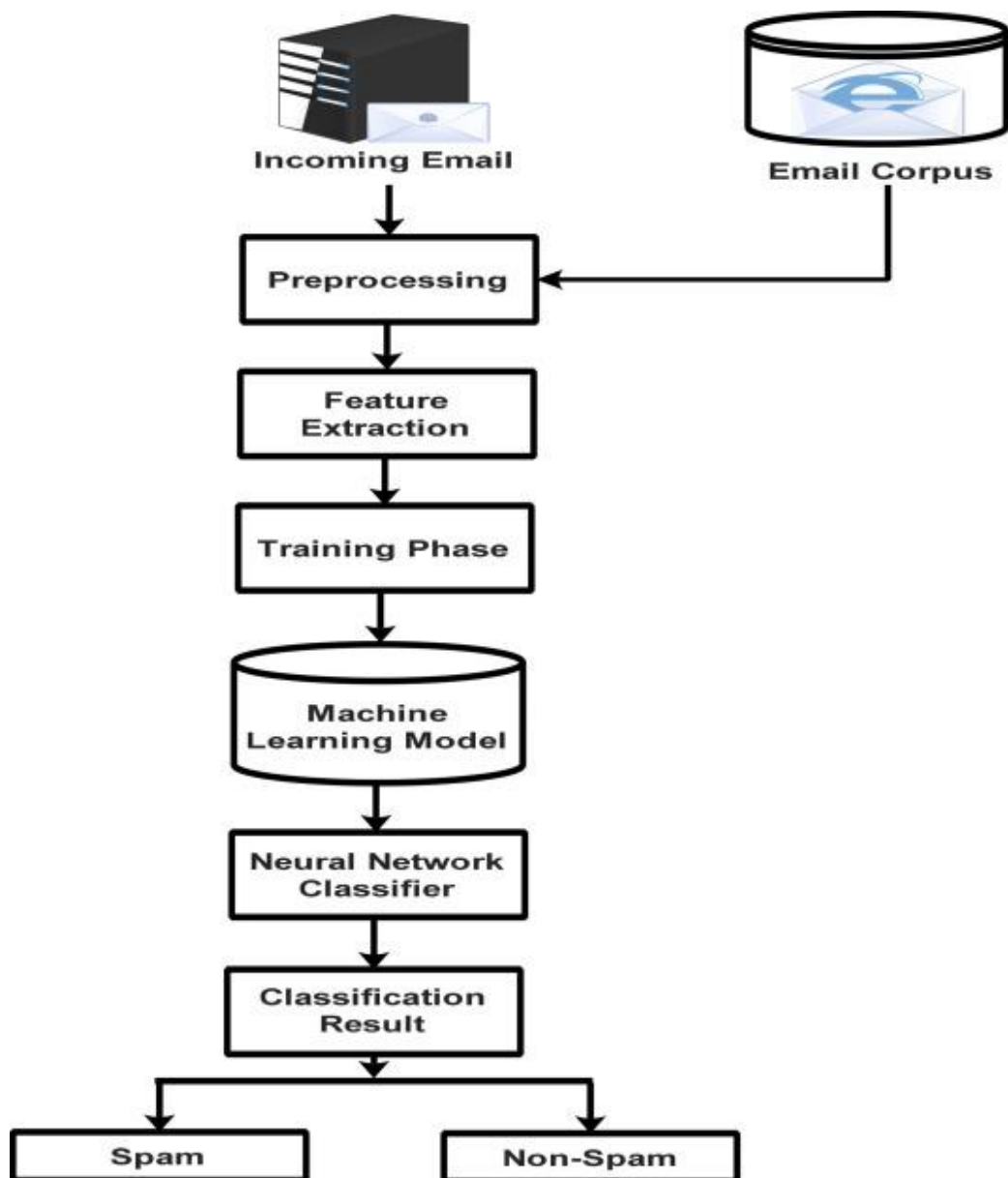
4. **Feature Extraction:** Converting the text data into numerical features that machine learning algorithms can work with. Common techniques include TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings.

Proper data pre-processing is pivotal because it sets the stage for the subsequent phases of our project. Clean, organized, and structured data allows us to build effective machine learning models. The quality of our spam classifier heavily depends on how well we manage the data in this phase.

By the end of this phase, we will have a well-organized dataset ready for analysis, model development, and evaluation. This foundational work is essential for harnessing the power of machine learning and natural language processing to distinguish spam from legitimate SMS messages accurately. Our success in building an effective spam classifier largely hinges on the precision and care with which we handle the data in this phase.

Advantages of pre-processing the data:

1. Improved model performance.
2. Noise reduction.
3. Handling the missing data.
4. Data standardization.
5. Better handling of categorical data.
6. Enhanced robustness.



PYTHON CODE:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import joblib
print("All the libraries imported successfully\n")
```

```
labels = open("labels.txt", "r")
labels = labels.read()
labels = list(labels.split("\n"))
commands = open("commands.txt", "r")
commands = commands.read()
commands = list(commands.split("\n"))
print("Dataset read successfully\n")
```

```
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(commands)
```

```
X_train, X_val, y_train, y_val = train_test_split(X, labels, test_size=0.2,
random_state=42)
model = SVC(kernel='linear')
print("Training is started. . . . .\n")
model.fit(X_train, y_train)
print("Training is completed successfully\n")
```

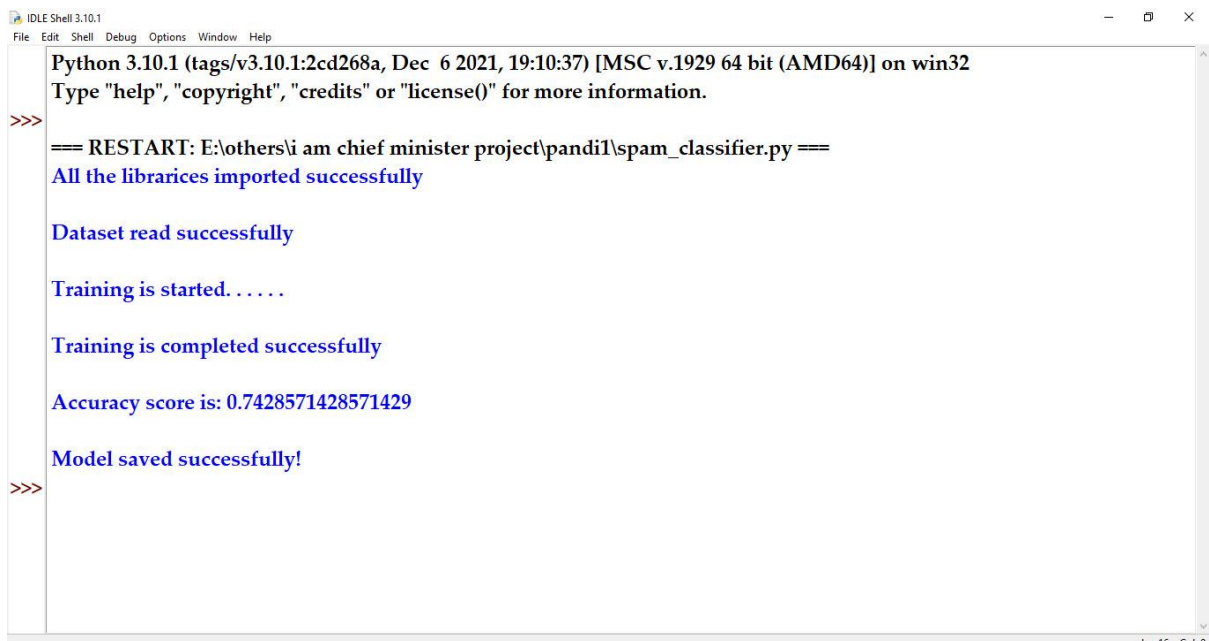
```
predictions = model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)

print("Accuracy score is:", accuracy, "\n")


joblib.dump(model, "model.pkl")

print("Model saved successfully!")
```



```
IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: E:\others\i am chief minister project\pandi1\spam_classifier.py ==
All the librarices imported successfully

Dataset read successfully

Training is started. . . . .

Training is completed successfully

Accuracy score is: 0.7428571428571429

Model saved successfully!
>>>
```

1. Importing Libraries:

The code starts by importing necessary libraries, including:

- `TfidfVectorizer`` from ``sklearn.feature_extraction.text`` to convert text data into TF-IDF (Term Frequency-Inverse Document Frequency) features.
- `SVC`` (Support Vector Classification) from ``sklearn.svm`` for the SVM classifier.
- `train_test_split`` from ``sklearn.model_selection`` to split the dataset into training and validation sets.
- `accuracy_score`` from ``sklearn.metrics`` to calculate the accuracy of the classifier.

- ``joblib`` for saving the trained model to a file.

2. Reading Data:

- The code opens two text files: "labels.txt" and "commands.txt."
- It reads the content of these files into two separate variables
- ``labels``: This variable will store the labels or categories for each command.
- ``commands``: This variable will store the textual commands or data.

3. TF-IDF Vectorization:

- The code initializes a ``TfidfVectorizer`` as ``vectorizer``.
- It uses the ``fit_transform`` method to convert the text data in the ``commands`` variable into a TF-IDF representation. This process creates a numerical feature matrix ``X`` where each row corresponds to a command, and each column corresponds to a TF-IDF-weighted term in the command.

4. Data Splitting:

The code uses the ``train_test_split`` function to split the data into training and validation sets. The splitting is done in an 80-20 ratio, with 80% of the data used for training (``X_train`` and ``y_train``) and 20% for validation (``X_val`` and ``y_val``).

5. Model Training:

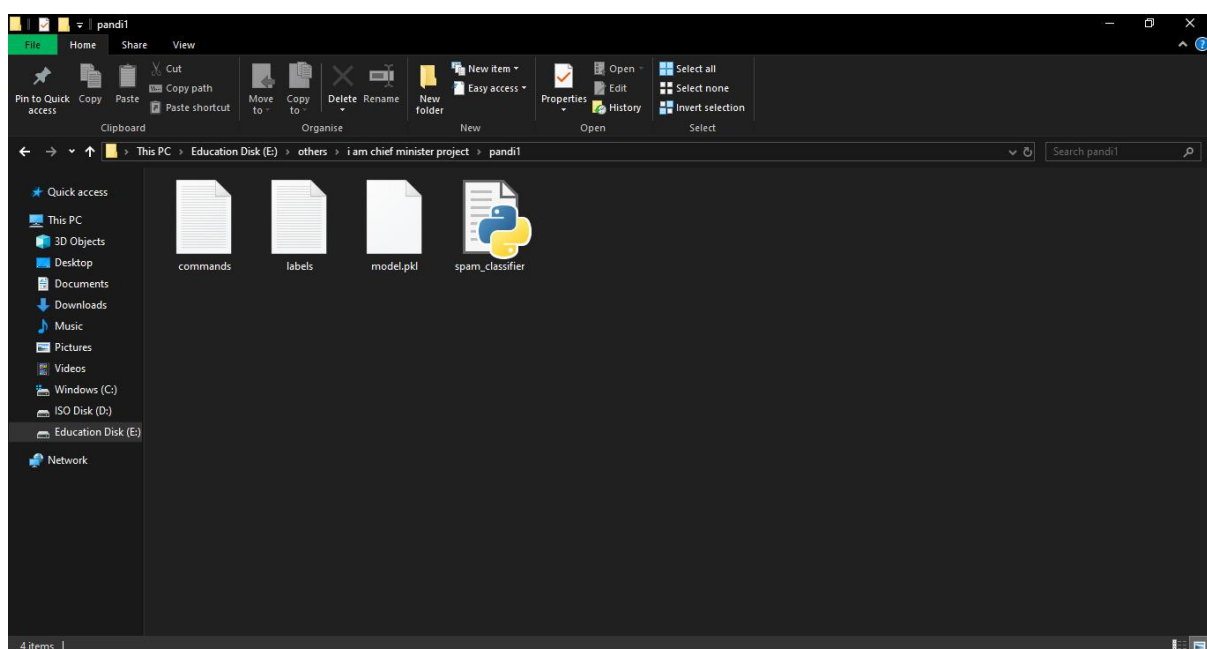
- An SVM classifier is initialized with a linear kernel and assigned to the ``model`` variable.
- The code then fits the model to the training data using the ``fit`` method. This is where the actual training of the classifier takes place.

6. Model Evaluation:

- The code makes predictions on the validation data using the trained model and stores the predictions in the ``predictions`` variable.
- It calculates the accuracy of the model's predictions using the ``accuracy_score`` function and prints the accuracy score to the console.

7. Model Saving:

The trained SVM model is saved to a file named "model.pkl" using the ``joblib.dump`` function. This allows you to later load and use the trained model for making predictions on new data without having to retrain it.



CONCLUSION:

In conclusion, a spam classifier is a valuable tool for filtering unwanted and potentially harmful messages from users' inboxes. It involves the development of a machine learning model trained on labelled data, and its performance can be continually improved through rigorous evaluation, feature engineering, and user feedback. Successful spam classification helps protect users from unwanted and potentially harmful content, enhancing their digital experience.

