Building a Smarter AI-Powered Spam Classifier

Team member

621421104021-Harivishwa.s.p

Phase 5 project submission

Project title: AI Powered Spam Classifier

Topic: documenting the complete project and prepare it for submission.

INTRODUCTION:

An AI-powered spam classifier is a computer program or system that uses artificial intelligence techniques, such as machine learning, to automatically identify and filter out unwanted or unsolicited messages, emails, or content, commonly known as spam. This technology employs algorithms to analyze and categorize incoming data based on various features and patterns, helping users avoid the clutter and potential security risks associated with spam. AI-powered spam classifiers have become increasingly effective at distinguishing between legitimate and unwanted messages, making them a valuable tool in maintaining the quality and security of digital communications.

Here are some key components and tools commonly used:

Programming Language: Java is a viable choice, but Python is also a popular language for building machine learning models and text classifiers due to its rich ecosystem of libraries.

Machine Learning Libraries:

- Scikit-Learn: A popular machine learning library in Python that includes various classification algorithms.
- TensorFlow and Keras: For deep learning-based text classification.

Text Processing Libraries:

- NLTK (Natural Language Toolkit) for Python: Offers tools for working with human language data.
- spaCy: Another Python library for advanced natural language processing tasks.

Data Preparation and Analysis:

- Pandas: A Python library for data manipulation and analysis.
- NumPy: Used for numerical operations in Python.

IDE (Integrated Development Environment):

- PyCharm for Python: A popular IDE with powerful debugging and coding assistance.
- Eclipse for Java: A widely used Java IDE.

Data Collection:

For building a spam classifier, you'll need a dataset of labeled examples. Tools for web scraping, data extraction, and preprocessing can be handy.

Jupyter Notebook: Particularly useful for creating and sharing documents that contain live code, equations, visualizations, and narrative text.

Version Control:

- Git: For version control of your code.

Data Visualization:

- Matplotlib and Seaborn: For creating data visualizations.
- Tableau or Power BI: For more advanced data visualization and exploration.

Docker: For containerization and easy deployment of machine learning models.

Database Systems: Depending on your data storage needs, you may require database systems like MySQL, PostgreSQL, or NoSQL databases.

Cloud Computing Platforms: Cloud platforms like AWS, Google Cloud, or Microsoft Azure can be used for scalable machine learning experiments and deployments.

Web Frameworks: If you plan to create a web-based application around your spam classifier, you might need web development frameworks such as Django (Python) or Spring Boot (Java).

Visualization Tools: Tools like TensorBoard (for TensorFlow) or Plotly can help you visualize model performance and results.

Model Deployment: For deploying your spam classifier, you may need server infrastructure or use cloud-based deployment services like AWS Lambda, Google Cloud Functions, or Heroku.

Automated ML Platforms: If you're looking for a more streamlined approach, consider using automated machine learning platforms like Azure AutoML, Google AutoML, or H2O.ai.

Machine Learning Algorithms:

AI-powered spam classifiers utilize machine learning algorithms to automatically recognize and categorize incoming messages or content as spam or not spam. These algorithms learn from patterns and features within the data to make accurate predictions.

Feature Extraction:

These systems extract various features from messages, such as text content, sender information, subject lines, and more. These features are then used to train the AI model.

Training Data:

To build an effective spam classifier, a large dataset of labeled examples is used for training. This dataset typically includes a mix of spam and non-spam (ham) messages, allowing the AI model to learn from real-world examples.

Classification Process:

When a new message arrives, the AI classifier evaluates it based on the patterns it has learned during training. It assigns a probability or a label (spam or not spam) to the message.

Thresholds and Rules:

AI classifiers often use thresholds or rules to determine whether a message is classified as spam. These can be adjusted to control the sensitivity of the classifier, allowing users to customize their spam filtering preferences.

Improvement over Time:

AI-powered spam classifiers can continuously improve their accuracy by retraining on new data. This adaptability is essential to keep up with evolving spam tactics.

Types of Spam:

AI classifiers can identify various types of spam, including email spam, SMS spam, comment spam on websites, and even social media spam. They can also detect phishing attempts and malicious content.

User Experience:

The primary benefit of AI-powered spam classifiers is to enhance the user experience by reducing the volume of unwanted or potentially harmful messages. This helps users focus on genuine and relevant communications.

Challenges:

Despite their effectiveness, these classifiers may occasionally have false positives (legitimate messages classified as spam) and false negatives (spam messages that get through). Striking the right balance is an ongoing challenge.

Security and Privacy:

AI classifiers play a crucial role in safeguarding users against malicious content, scams, and phishing attacks, contributing to overall online security and privacy.

Dataset Link:https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

Phase2:**Innovation-** moving design into innovation to solve the problem

What is Natural Language Processing (NLP) ?

Natural Language processing or NLP is a subset of Artificial Intelligence (AI), where it is basically responsible for the understanding of human language by a machine or a robot.

One of the important subtopics in NLP is Natural Language Understanding (NLU) and the reason is that it is used to understand the structure and meaning of human language, and then with the help of computer science transform this linguistic knowledge into algorithms of Rules-based machine learning that can solve specific problems and perform desired tasks.

LANGUAGE PROCCESS IN PYTHON:

The purpose of this article is to show you how to detect spam in SMS.

For that, we use a dataset from the UCI datasets, which is a public set that contain SMS labelled messages that have been collected for mobile phone spam research. It has one collection composed by 5.574 SMS phone messages in English, tagged according being legitimate (ham) or spam.

Therefore, we will train a model to learn to automatically discriminate between ham / spam. Then we will use “test data” to test the model. Finally to evaluate if our model is efficient, we will calculate Accuracy, Classification report and Confusion Matrix.

Exploratory Data Analysis
To get started, we should first imports all the library, then load the data and rename names columns:

```
# Import library
import pandas as pd
import numpy as np
```

- import string

```
import seaborn as sns
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from collections import Counter
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.model_selection import GridSearchCV
%matplotlib inline
# Load data
data = pd.read_excel('data.xlsx')
# Rename names columns
```

data.columns = ['label', 'messages']

Let’s see a description of our data:

data.describe()

	label	messages
count	5574	5574
unique	2	5171
top	ham	Sorry, I'll call later
freq	4827	30

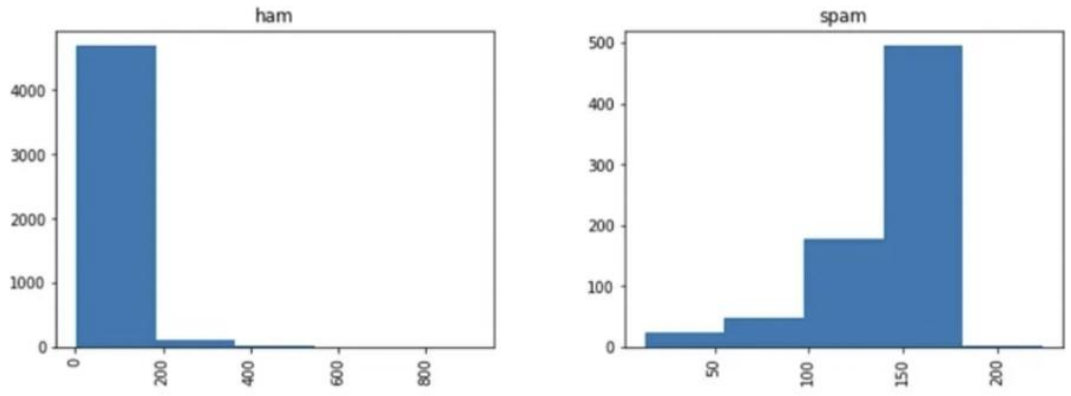
Data Description (Image by Author)

Note that our data contains a collection of 5574 SMS and also we have only 2 label: ham and spam. Now, we create a column called ‘length’ to know how long the text messages are and then plot it against the label:

```
data["length"] = data["messages"].apply(len)
data.sort_values(by='length', ascending=False).head(10)
```

	label	messages	length
1085	ham	For me the love should start with attraction.i...	910
1863	ham	The last thing i ever wanted to do was hurt yo...	790
2434	ham	Indians r poor but India is not a poor country...	629
1579	ham	How to Make a girl Happy? It's not at all diff...	611
2849	ham	Sad story of a Man - Last week was my b'day. M...	588
2158	ham	Sad story of a Man - Last week was my b'day. M...	588
2380	ham	Good evening Sir, hope you are having a nice d...	482
3017	ham	<#> is fast approaching. So, Wish u a v...	461
1513	ham	Hey sweet, I was wondering when you had a mome...	458
2370	ham	A Boy loved a gal. He propsd bt she didnt mind...	446

data.hist(column = 'length', by ='label',figsize=(12,4), bins = 5)



Histogram between lenght and label (Image by Author)
Note that through the histogram, we have been able to discover that spam messages tend to have more characters.

Most likely, most of the data you have come across is numeric or categorical, but what happens when it is of type string (text format)?

As you may have noticed, our data is of type string. Therefore, we should transform it into a numeric vector to be able to perform the classification task. To do this, we use bag-of-words where each unique word in a text will be represented by a number. However, before doing this transformation, we should remove all punctuations and then common words like: [‘I’, ‘my’, ‘myself’, ‘we’, ‘our’, ‘our’ , ‘ourselves’, ‘you’, ‘are’ …]. This process is called tokenization. After this process, we convert our string sequence into number sequences.

Remove all punctuation: Suppose we have the following sentence:
**********Hi everyone!!! it is a pleasure to meet you.**********
and we want to remove !!! and .

First, we load the import string library and do the following:

```
message = "Hi everyone!!! it is a pleasure to meet you."
message_not_punc = [] ii
for punctuation in message:
    if punctuation not in string.punctuation:
            message_not_punc.append(punctuation)
# Join the characters again to form the string.
message_not_punc = ".join(message_not_punc)
print(message_not_punc)
>>> Hi everyone it is a pleasure to meet you
```

2. Remove common words:

To do that, we use the library nltk, i.e, from nltk.corpus import stopwords

Is important to know that stopwords have 23 languages supported by it (this number must be up to date). In this case, we use English language:

```
from nltk.corpus import stopwords
# Remove any stopwords for remove_punc, but first we should to
transform this into the list.
message_clean = list(message_not_punc.split(" "))
# Remove any stopwords
i = 0
while i <= len(message_clean):
 for mess in message_clean:
    if mess.lower() in stopwords.words('english'):
                message_clean.remove(mess)

 i =i +1

print(message_clean)
>>> ['Hi', 'everyone', 'pleasure', 'meet']
```

As you may have noticed, our data is of type string. Therefore, we should transform it into a numeric vector to be able to perform the classification task. To do this, we use bag-of-words where each unique word in a text will be represented by a number. However, before doing this transformation, we should remove all punctuations and then common words like: [‘I’, ‘my’, ‘myself’, ‘we’, ‘our’, ‘our’ , ‘ourselves’, ‘you’, ‘are’ …]. This process is called tokenization. After this process, we convert our string sequence into number sequences.

Remove all punctuation: Suppose we have the following sentence: **********Hi everyone!!! it is a pleasure to meet you.********** and we want to remove !!! and .

First, we load the import string library and do the following:

```
message = "Hi everyone!!! it is a pleasure to meet you."
```

```
message_not_punc = []
for punctuation in message:
    if punctuation not in string.punctuation:
        message_not_punc.append(punctuation)
# Join the characters again to form the string.
message_not_punc = ''.join(message_not_punc)
print(message_not_punc)
>>> Hi everyone it is a pleasure to meet you
```

Phase 3: Development part 1-building the project by loading and preprocessing the dataset.

Let's build a spam classifier program in python which can tell whether a given message is spam or not! We can do this by using a simple, yet powerful theorem from probability theory called Baye's Theorem. It is mathematically expressed as

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)},$$

where A and B are events and $P(B) \neq 0$.

- $P(A)$ and $P(B)$ are the probabilities of observing A and B without regard to each other.
- $P(A \mid B)$, a conditional probability, is the probability of observing event A given that B is true.
- $P(B \mid A)$ is the probability of observing event B given that A is true.

Baye's Theorem

Problem Statement

We have a message m = (w*1*, w*2*, ... , w*n*), where (w*1*, w*2*, ... , w*n*) is a set of unique words contained in the message. We need to find

$$P(spam|w1 \cap w2 \cap \ldots \cap wn) = \frac{P(w1 \cap w2 \cap \ldots \cap wn|spam).\,P(spam)}{P(w1 \cap w2 \cap \ldots \cap wn)}$$

If we assume that occurrence of a word are independent of all other words, we can simplify the above expression to

$$\frac{P(w1|spam).\,P(w2|spam)\ldots P(wn|spam).\,P(spam)}{P(w1).\,P(w2)..P(wn)}$$

In order to classify we have to determine which is greater

$$P(spam|w1 \cap w2 \cap \ldots \cap wn)\ versus\ P(\sim spam|w1 \cap w2 \cap \ldots \cap wn)$$

1. Loading dependencies

```
In [1]: from nltk.tokenize import word_tokenize
        from nltk.corpus import stopwords
        from nltk.stem import PorterStemmer
        import matplotlib.pyplot as plt
        from wordcloud import WordCloud
        from math import log, sqrt
        import pandas as pd
        import numpy as np
        %matplotlib inline
```

We are going to make use of NLTK for processing the messages, WordCloud and matplotlib for visualization and pandas for loading data, NumPy for generating random probabilities for train-test split.

2. Loading Data

```
In [2]: mails = pd.read_csv('spam.csv', encoding = 'latin-1')
        mails.head()
```

Out[2]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

We do not require the columns ‘Unnamed: 2’, ‘Unnamed: 3’ and ‘Unnamed: 4’, so we remove them. We rename the column ‘v1’ as ‘label’ and ‘v2’ as ‘message’. ‘ham’ is replaced by 0 and ‘spam’ is replaced by 1 in the ‘label’ column. Finally we obtain the following dataframe.

	message	label
0	Go until jurong point, crazy.. Available only ...	0
1	Ok lar... Joking wif u oni...	0
2	Free entry in 2 a wkly comp to win FA Cup fina...	1
3	U dun say so early hor... U c already then say...	0
4	Nah I don't think he goes to usf, he lives aro...	0

3. Train-Test Split

To test our model we should split the data into train dataset and test dataset. We shall use the train dataset t0 train the model and then it will be tested on the test dataset. We shall use 75% of the dataset as train dataset and the rest as test dataset. Selection of this 75% of the data is uniformly random.

In [8]:
```
totalMails = mails['message'].shape[0]
trainIndex, testIndex = list(), list()
for i in range(mails.shape[0]):
    if np.random.uniform(0, 1) < 0.75:
        trainIndex += [i]
    else:
        testIndex += [i]
trainData = mails.loc[trainIndex]
testData = mails.loc[testIndex]
```

In [9]:
```
trainData.reset_index(inplace = True)
trainData.drop(['index'], axis = 1, inplace = True)
trainData.head()
```

Out[9]:

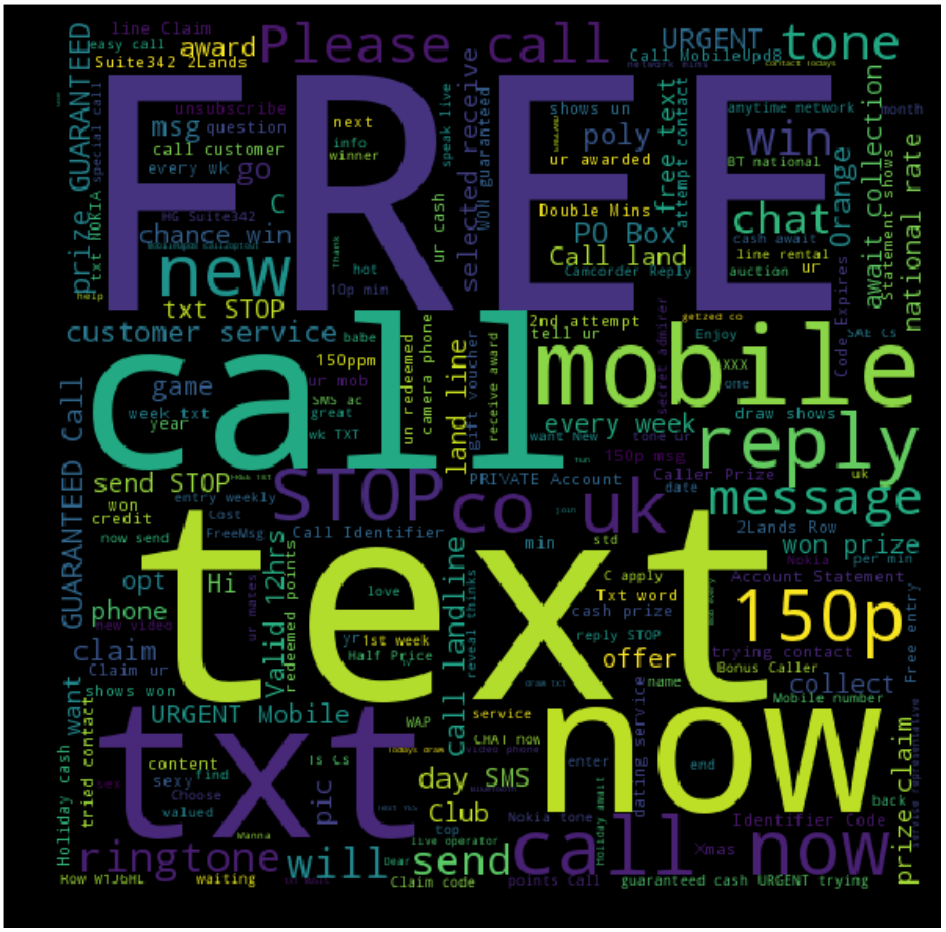
	message	label
0	Go until jurong point, crazy.. Available only ...	0
1	Free entry in 2 a wkly comp to win FA Cup fina...	1
2	U dun say so early hor... U c already then say...	0
3	FreeMsg Hey there darling it's been 3 week's n...	1
4	As per your request 'Melle Melle (Oru Minnamin...	0

4. Visualizing data

Let us see which are the most repeated words in the spam messages! We are going to use WordCloud library for this purpose.

```
In [13]: spam_words = ' '.join(list(mails[mails['label'] == 1]['message']))
         spam_wc = WordCloud(width = 512,height = 512).generate(spam_words)
         plt.figure(figsize = (10, 8), facecolor = 'k')
         plt.imshow(spam_wc)
         plt.axis('off')
         plt.tight_layout(pad = 0)
         plt.show()
```

This results in the following



As expected, these messages mostly contain the words like ‘FREE’, ‘call’, ‘text’, ‘ringtone’, ‘prize claim’ etc.

Similarly the wordcloud of ham messages is as follows:

5. Training the model

We are going to implement two techniques: Bag of words and TF-IDF. I shall explain them one by one. Let us first start off with Bag of words.

Preprocessing: Before starting with training we must preprocess the messages. First of all, we shall make all the character lowercase. This is because ‘free’ and ‘FREE’ mean the same and we do not want to treat them as two different words.

Then we tokenize each message in the dataset. Tokenization is the task of splitting up a message into pieces and throwing away the punctuation characters. For eg.:

Input: Friends, Romans, Countrymen, lend me your ears;

Output: | Friends | | Romans | | Countrymen | | lend | | me | | your | | ears |

The words like ‘go’, ‘goes’, ‘going’ indicate the same activity. We can replace all these words by a single word ‘go’. This is called stemming. We are going to use Porter Stemmer, which is a famous stemming algorithm.

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

We then move on to remove the stop words. Stop words are those words which occur extremely frequently in any text. For example words like ‘the’, ‘a’, ‘an’, ‘is’, ‘to’ etc. These words do not give us any information about the content of the text. Thus it should not matter if we remove these words for the text.

Optional: You can also use n-grams to improve the accuracy. As of now, we only dealt with 1 word. But when two words are together the meaning totally changes. For example, ‘good’ and ‘not good’ are opposite in meaning. Suppose a text contains ‘not good’, it is better to consider ‘not good’ as one token rather than ‘not’ and ‘good’. Therefore, sometimes accuracy is improved when we split the text into tokens of two (or more) words than only word.

```
In [19]: def process_message(message, lower_case = True, stem = True, stop_words = True, gram = 2):
             if lower_case:
                 message = message.lower()
             words = word_tokenize(message)
             words = [w for w in words if len(w) > 2]
             if gram > 1:
                 w = []
                 for i in range(len(words) - gram + 1):
                     w += [' '.join(words[i:i + gram])]
                 return w
             if stop_words:
                 sw = stopwords.words('english')
                 words = [word for word in words if word not in sw]
             if stem:
                 stemmer = PorterStemmer()
                 words = [stemmer.stem(word) for word in words]
             return words
```

Bag of Words: In Bag of words model we find the ‘term frequency’, i.e. number of occurrences of each word in the dataset. Thus for word w,

$$P(w) = \frac{Total\ number\ of\ occurrences\ of\ w\ in\ dataset}{Total\ number\ of\ words\ in\ dataset}$$

and

$$P(w|spam) = \frac{Total\ number\ of\ occurrences\ of\ w\ in\ spam\ messages}{Total\ number\ of\ words\ in\ spam\ messages}$$

TF-IDF: TF-IDF stands for Term Frequency-Inverse Document Frequency. In addition to Term Frequency we compute Inverse document frequency.

$$IDF(w) = \log \frac{Total\ number\ of\ messages}{Total\ number\ of\ messages\ containing\ w}$$

For example, there are two messages in the dataset. 'hello world' and 'hello foo bar'. TF('hello') is 2. IDF('hello') is log(2/2). If a word occurs a lot, it means that the word gives less information.

In this model each word has a score, which is TF(w)*IDF(w). Probability of each word is counted as:

$$P(w) = \frac{TF(w) * IDF(w)}{\sum_{\forall\ words\ x \in train\ dataset} TF(x) * IDF(x)}$$

$$P(w|spam) = \frac{TF(w|spam) * IDF(w)}{\sum_{\forall\ words\ x \in train\ dataset} TF(x|spam) * IDF(x)}$$

Additive Smoothing:So what if we encounter a word in test dataset which is not part of train dataset? In that case P(w) will be 0, which will make the P(spam|w) undefined (since we would have to divide by P(w) which is 0. Remember the formula?). To tackle this issue we introduce additive smoothing. In additive smoothing we add a number alpha to the numerator and add alpha times number of classes over which the probability is found in the denominator.

$$P(w|spam) = \frac{TF(w|spam) \ + \ \alpha}{\sum_{\forall \ words \ x \in \ spam \ in \ train \ dataset} TF(x) \ + \ \alpha \sum_{\forall \ words \ x \in \ spam \ in \ train \ dataset} 1}$$

When using TF-IDF

$$P(w|spam) = \frac{TF(w|spam) * IDF(w) \ + \ \alpha}{\sum_{\forall \ words \ x \in \ train \ dataset} TF(x) * IDF(x) \ + \ \alpha \sum_{\forall \ words \ x \in \ spam \ in \ train \ dataset} 1}$$

This is done so that the least probability of any word now should be a finite number. Addition in the denominator is to make the resultant sum of all the probabilities of words in the spam emails as 1.

When alpha = 1, it is called Laplace smoothing.

6. Classification

For classifying a given message, first we preprocess it. For each word w in the processed messaged we find a product of P(w|spam). If w does not exist in the train dataset we take TF(w) as 0 and find P(w|spam) using above formula. We multiply this product with P(spam) The resultant product is the P(spam|message). Similarly, we find P(ham|message). Whichever probability among these two is greater, the corresponding tag (spam or ham) is assigned to the input message. Note than we are not dividing by P(w) as given in the formula. This is because both the numbers will be divided by that and it would not affect the comparison between the two.

7. Final result

```
In [22]: sc_tf_idf = SpamClassifier(trainData, 'tf-idf')
         sc_tf_idf.train()
         preds_tf_idf = sc_tf_idf.predict(testData['message'])
         metrics(testData['label'], preds_tf_idf)

         Precision:  0.8873239436619719
         Recall:  0.6596858638743456
         F-score:  0.7567567567567568
         Accuracy:  0.94164265129683

In [23]: sc_bow = SpamClassifier(trainData, 'bow')
         sc_bow.train()
         preds_bow = sc_bow.predict(testData['message'])
         metrics(testData['label'], preds_bow)

         Precision:  0.890625
         Recall:  0.5968586387434555
         F-score:  0.7147335423197492
         Accuracy:  0.9344380403458213

In [24]: pm = process_message('I cant pick the phone right now. Pls send a message')
         sc_tf_idf.classify(pm)

Out[24]: False

In [25]: pm = process_message('Congratulations ur awarded $500 ')
         sc_tf_idf.classify(pm)

Out[25]: True
```

Phase4:Development part 2-continue building the project by performing different activities like feature engineering, model training, evaluation

1. Feature Engineering:

Feature engineering involves selecting and creating relevant features from the available data to train the spam classifier. Here are some key activities in feature engineering:

a. Text Preprocessing:

Raw text data needs to be preprocessed to remove noise and standardize the format. This includes steps like lowercasing, removing punctuation, handling special characters, and converting text to a consistent format.

b. Feature Extraction:

Various features can be extracted from text data to improve the classifier's performance. Some common features include word frequency, term frequency-inverse document frequency (TF-IDF), n-grams, and presence of specific patterns or keywords.

c. Feature Selection:

Not all features may contribute equally to the classifier's performance. Feature selection techniques like chi-square test, information gain, or L1 regularization can be used to identify the most informative features.

d. Feature Transformation:

Sometimes, transforming the features can improve the classifier's performance. Techniques like dimensionality reduction (e.g., Principal Component Analysis) or feature scaling (e.g., normalization or standardization) can be applied.

2. Model Training:

Model training involves training a machine learning model using the preprocessed and engineered features. Here are the steps involved in model training:

a. Dataset Split:

The labeled dataset, containing both spam and non-spam (ham) examples, is split into a training set and a validation set. The training set is used to train the model, while the validation set is used to tune

hyperparameters and evaluate the model's performance during training.

b. Model Selection:

Various machine learning algorithms can be used for spam classification, such as Naive Bayes, Support Vector Machines (SVM), Random Forest, or Gradient Boosting. The choice of the algorithm depends on the dataset and the desired performance characteristics.

c. Model Training:

The selected algorithm is trained on the training set using the preprocessed features. The model learns the patterns and relationships between the features and the class labels.

d. Hyperparameter Tuning:

Hyperparameters are parameters that control the behavior of the machine learning algorithm. Techniques like grid search, random search, or Bayesian optimization can be used to find the optimal combination of hyperparameters that maximize the model's performance.

3. Evaluation:

After training the model, it needs to be evaluated to assess its performance. Here are some evaluation activities:

a. Performance Metrics:

Common performance metrics for spam classification include accuracy, precision, recall, F1 score, and area under the receiver operating characteristic curve (AUC-ROC). These metrics provide insights into the model's ability to correctly classify spam and non-spam examples.

b. Cross-Validation:

Cross-validation is a technique used to assess the model's generalization performance. It involves splitting the dataset into multiple folds, training and evaluating the model on different fold combinations to obtain a more robust estimate of the model's performance.

c. Error Analysis:

Analyzing the misclassified examples can provide valuable insights into the model's weaknesses and areas for improvement. It helps identify patterns or specific types of spam that the model struggles to classify correctly.

d. Iterative Refinement:

Based on the evaluation results, the model can be refined by adjusting feature engineering techniques, trying different algorithms, or optimizing hyperparameters. This iterative process helps improve the model's performance over time.

Implementation:

Steps to implement Spam Classification using OpenAI

Now there are two approaches that we will be covering in this article:

1. Using Embeddings API developed by OpenAI

Step 1: Install all the necessary salaries

```
!pip install -q openai
```

Step 2: Import all the required libraries

Python3

```
# necessary libraries
```

```
import openai

import pandas as pd

import numpy as np

# libraries to develop and evaluate a machine learning model

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report, accuracy_score

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report, accuracy_score
```

```
from sklearn.metrics import confusion_matrix
```

Step 3: Assign your API key to the OpenAI environment

Python3

```
# replace "YOUR API KEY" with your generated API key

openai.api_key = "YOUR API KEY"
```

Step 4: Read the CSV file and clean the dataset

Our dataset has 3 unnamed columns with NULL values,

Note: Open AI's public API does not process more than 60 requests per minute. so we will drop them and we are taking only 60 records here only.

Python3

```
# while loading the csv, we ignore any encoding errors and skip any bad line

df = pd.read_csv('spam.csv', encoding_errors='ignore',
on_bad_lines='skip')


print(df.shape)

# we have 3 columns with NULL values, to remove that we use the below line
```

```
df = df.dropna(axis=1)

# we are taking only the first 60 rows for developing the model

df = df.iloc[:60]

# rename the columns v1 and v2 to Output and Text respectively

df.rename(columns = {'v1':'OUTPUT', 'v2': 'TEXT'}, inplace = True)


print(df.shape)

df.head()
```

Output:

```
(5572, 5)
(60, 2)
```

	OUTPUT	TEXT
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Email Spam Classification Dataset

Step 5: Define a function to use Open AI's Embedding API

We use the Open AI's Embedding function to generate embedding vectors and use them for classification. Our API uses the "text-embedding-ada-002" model which belongs to the second generation of embedding models developed by OpenAI. The embeddings generated by this model are of length 1536.

Python3

```
# function to generate vector for a string

def get_embedding(text, model="text-embedding-ada-002"):


   return openai.Embedding.create(input = ,
model=model)['data'][0]['embedding']



# applying the above funtion to generate vectors for all 60 text pieces

df["embedding"] = df.TEXT.apply(get_embedding).apply(np.array)  # convert string to array

df.head()
```

Output:

	OUTPUT	TEXT	embedding
0	ham	Go until jurong point, crazy.. Available only ...	[-0.011956056579947472, -0.026185495778918266,...
1	ham	Ok lar... Joking wif u oni...	[-0.0024703105445951223, -0.0312176700681448, ...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	[-0.008984447456896305, 0.0006775223882868886,...
3	ham	U dun say so early hor... U c already then say...	[0.010833987966179848, -0.011291580274701118, ...
4	ham	Nah I don't think he goes to usf, he lives aro...	[0.012792329303920269, -1.7723063137964346e-05...

Email Spam Classification Dataset

Step 6: Custom Label the classes of the output variable to 1 and 0, where 1 means “spam” and 0 means “not spam”.

Python3

```
class_dict = {'spam': 1, 'ham': 0}


df['class_embeddings'] = df.OUTPUT.map(class_dict)

df.head()
```

Output:

Spam Classification dataFrame after feature engineerin

Step 7: Develop a Classification model.

We will be splitting the dataset into a training set and validation dataset using train_test_split and training a Ra

	OUTPUT	TEXT	embedding	class_embeddings
0	ham	Go until jurong point, crazy.. Available only ...	[-0.011956056579947472, -0.026185495778918266,...	0
1	ham	Ok lar... Joking wif u oni...	[-0.0024703105445951223, -0.0312176700681448, ...	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	[-0.008984447456896305, 0.0006775223882868886,...	1
3	ham	U dun say so early hor... U c already then say...	[0.010833987966179848, -0.011291580274701118, ...	0
4	ham	Nah I don't think he goes to usf, he lives aro...	[0.012792329303920269, -1.7723063137964346e-05...	0

ndom Forest Classification model.

Python3

```
# split data into train and test

X = np.array(df.embedding)


y = np.array(df.class_embeddings)


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



# train random forest classifier

clf = RandomForestClassifier(n_estimators=100)

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

preds = clf.predict(X_test.tolist())
```

```
# generate a classification report involving f1-score, recall, precision and accuracy

report = classification_report(y_test, preds)


print(report)
```

Output:

```
              precision    recall  f1-score   support

           0       0.82      1.00      0.90         9
           1       1.00      0.33      0.50         3

    accuracy                           0.83        12
   macro avg       0.91      0.67      0.70        12
weighted avg       0.86      0.83      0.80        12
```

Step 8: Calculate the accuracy of the model

Python3

```
print("accuracy: ", np.round(accuracy_score(y_test, preds)*100,2), "%")
```

Output:

```
accuracy:  83.33 %
```

Step 9: Print the confusion matrix for our classification model

Python3

```
confusion_matrix(y_test, preds)
```

Output:

```
array([[9, 0],
       [2, 1]])
```

2. Using text completion API developed by OpenAI

Step 1: Install the Openai library in the Python environment

```
!pip install -q openai
```

Step 2: Import the following libraries

Python3

```
import openai
```

Step 3: Assign your API key to the Openaithe environment

Python3

```
# replace "YOUR API KEY" with your generated API key

openai.api_key = "YOUR API KEY"
```

Step 4: Define a function using the text completion API of Openai

Python3

```
def spam_classification(message):

  response = openai.Completion.create(

    model="text-davinci-003",

    prompt=f"Classify the following message as spam or not spam:\n\n{message}\n\nAnswer:",

    temperature=0,

    max_tokens=64,

    top_p=1.0,

    frequency_penalty=0.0,

    presence_penalty=0.0
```

```
)


return response['choices'][0]['text'].strip()
```

Step 5: Try out the function with some examples

Example 1:

Python3

```
out = spam_classification("""Congratulations! You've Won a $1000
gift card from walmart.

Go to https://bit.ly to claim your reward.""")

print(out)
```

Output:

```
Spam
```

Example 2:

Python3

```
out = spam_classification("Hey Alex, just wanted to let you know
tomorrow is an off. Thank you")
```

```
print(out)
```

Output:

```
Not spam
```

Advantages:

Simplicity: The code provides a straightforward and easy-to-understand implementation of a text classification task, making it accessible for those new to machine learning and classification.

Familiarity with Weka: Weka is a popular machine learning library with a user-friendly interface, making it a good choice for educational and prototyping purposes. It's widely used in academia and industry.

Rapid Prototyping: This code allows for quick prototyping of a text classification system, making it suitable for small-scale projects or educational purposes where speed of development is crucial.

Cross-Validation: The code includes cross-validation, which helps in assessing the model's generalization performance, ensuring that the classifier can handle unseen data effectively.

Foundation for Learning: It serves as a foundational example for those interested in delving deeper into text classification and understanding the basics of supervised machine learning.

Teaching and Learning: This code is valuable for educational purposes, as it can be used to teach the fundamentals of machine learning and text classification.

Disadvantages:

Limited Real-World Applicability: The code's simplicity and use of a basic Naive Bayes classifier make it unsuitable for real-world spam classification tasks, where more advanced algorithms and extensive feature engineering are required.

Data and Dataset Limitations: The code assumes the existence of a labeled dataset in ARFF format. In practice, obtaining a high-quality and diverse dataset for spam classification can be challenging.

No Feature Engineering: The code lacks feature engineering, which is crucial for effective text classification. Real-world classifiers require the extraction of meaningful features from text data.

No Preprocessing: It does not include text preprocessing steps like tokenization, stop-word removal, or stemming, which are essential for handling natural language text effects.

No Model Tuning: The code doesn't address hyperparameter tuning or model selection, which are vital for optimizing classifier performance.

Scalability: For larger datasets, this basic implementation may not scale well, and more efficient algorithms and libraries might be necessary.

Benefits:

Simplicity: The code is easy to understand and provides a straightforward introduction to text classification for those new to machine learning.

Familiarity with Weka: Weka is a widely used machine learning library with a user- interface, making it a suitable choice for educational purposes and rapid prototyping.

Cross-Validation: The code incorporates cross-validation, allowing for a more robust evaluation of the model's performance.

Educational Tool: This code serves as a valuable educational tool for teaching the fundamentals of machine learning and text classification.

conclusion:

Developing an AI-powered spam classifier involves a comprehensive set of tools and software that collectively enable the creation of an effective and efficient system for filtering unwanted messages. This multifaceted process encompasses everything from programming languages and machine learning libraries to data preprocessing, deployment, and visualization tools.

Thank you!!!

-Prepared by harivishwa