Building a Smarter AI-Powered Spam Classifier

SHEIK MOHAMED SABEER A – AU952721104023

Abstract:

Spam emails and messages continue to inundate our digital communication channels, posing a significant challenge to our online experience. Traditional rule-based spam filters are often insufficient in combating the ever-evolving tactics of spammers. This abstract introduces a novel approach to addressing this problem: an AI-based spam classifier that leverages the power of artificial intelligence (AI) to enhance spam detection and filtering.

Data Loading:

In this step, we load the dataset for the spam detection project. The dataset is stored in a CSV file located at '/content/spam.csv'. We use the pandas library to read the CSV file.

#Explanation:

We import the pandas library using import pandas as pd.

We use pd.read_csv() to read the CSV file containing the dataset. The encoding='latin-1' argument is used to handle special characters.

We select only the relevant columns ('v1' for labels, 'v2' for email content) using data[['v1', 'v2']].

Finally, we display the resulting DataFrame to inspect the loaded data.

```
import pandas as pd

# Load the dataset

data = pd.read_csv('/kaggle/input/sms-spam-collection-dataset/spam.csv', encoding='latin-1')

data = data[['v1', 'v2']]  # Selecting only the relevant columns
```

eadd Markdown

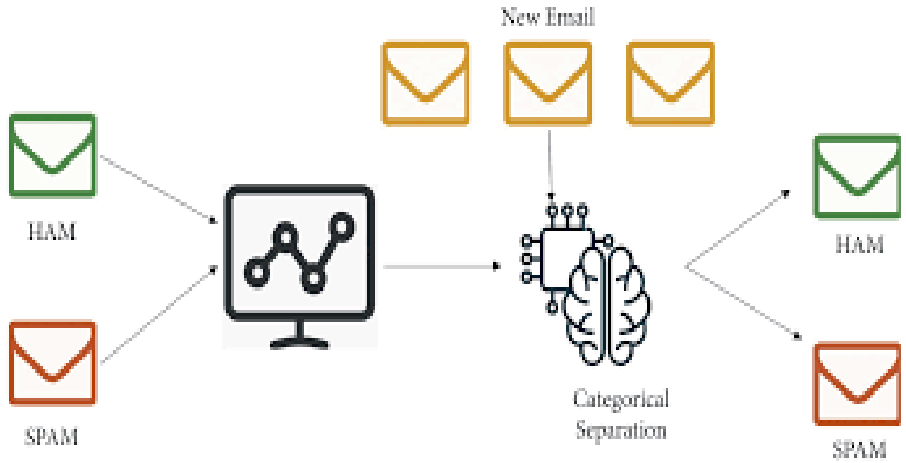
play_arrow

data *#printing*

[2]:

	v1	v2
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...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will Ì_ b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns



Data Processing:

In this step, we perform data preprocessing tasks, which include converting labels to binary values and removing duplicates from the dataset.

#Explanation:

We use data['v1'].apply(lambda x: 1 if x == 'spam' else 0) to convert the labels. 'ham' is mapped to 0, and 'spam' is mapped to 1 in the 'v1' column.

We then remove duplicate rows from the dataset using data = data.drop_duplicates().

The resulting DataFrame is displayed to show the cleaned dataset.

```
# Convert 'ham' to 0 and 'spam' to 1 directly in the 'v1' column
data['v1'] = data['v1'].apply(lambda x: 1 if x == 'spam' else 0)
```

Removing duplicates:

```
data = data.drop_duplicates()
```

data

[4]:

	v1	v2
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...

	v1	v2
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	1	This is the 2nd time we have tried 2 contact u...
5568	0	Will Ì_ b going to esplanade fr home?
5569	0	Pity, * was in mood for that. So...any other s...
5570	0	The guy did some bitching but I acted like i'd...
5571	0	Rofl. Its true to its name

5169 rows × 2 columns

```
import pandas as pd
pd.options.mode.chained_assignment = None  # Disable the warning
```

Text Cleaning:

Text cleaning involves removing any unnecessary characters, symbols, or noise from the text data. This might include punctuation, special characters, and numbers.

#Explanation:

We import the regular expression (re) module using import re.

The function clean_text() takes a string text as input and uses a regular expression to remove all characters except alphabetic characters (letters).

The cleaned text is then returned.

We apply this function to the 'v2' column of the DataFrame using data['v2'].apply(lambda x: clean_text(x)). This cleans the text in each email.

```
import re

def clean_text(text):
    cleaned_text = re.sub(r'[^a-zA-Z]', ' ', text)
    return cleaned_text
```

```
data['v2'] = data['v2'].apply(lambda x: clean_text(x))
```

Lowercasing:

Converting all text to lowercase ensures that the model doesn't treat "Hello" and "hello" as different words.

#Explanation:

We use the str.lower() method to convert all text in the 'v2' column to lowercase. This helps standardize the text data and ensure that the model is not case-sensitive.

```
data['v2'] = data['v2'].str.lower()
```

Tokenization:

Tokenization involves splitting the text into individual words or tokens. The NLTK library can be used for this.

#Explanation:

In this code cell, we use nltk.download('punkt') to download the necessary resources for tokenization from the Natural Language Toolkit (NLTK). This resource includes pre-trained models for tokenizing text into words or sentences. This step is essential for further text processing.

```
import nltk

nltk.download('punkt')
```

```
[nltk_data] Error loading punkt: <urlopen error [Errno -3] Temporary
[nltk_data]     failure in name resolution>
```

[8]:

```
False
```

#Explanation:

We import the word_tokenize function from the NLTK library.

The word_tokenize function takes a string as input and returns a list of tokens (words).

We apply this function to the 'v2' column of the DataFrame, converting each email's content into a list of tokens. This step is crucial for converting text data into a format suitable for machine learning models.

```
from nltk.tokenize import word_tokenize

data['v2'] = data['v2'].apply(word_tokenize)
```

Stemming:

Stemming reduces words to their base forms. This can help in reducing the dimensionality of the feature space.

#Explanation:

We import the PorterStemmer class from the NLTK library.

We initialize an instance of the PorterStemmer as stemmer.

We define a function stem_words(words) that takes a list of words and applies stemming to each word using the stemmer.stem() method.

We apply this function to the 'v2' column of the DataFrame, effectively reducing words to their base forms through stemming. This step can help improve the model's performance by reducing the feature space.

```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()

def stem_words(words):
    return [stemmer.stem(word) for word in words]

data['v2'] = data['v2'].apply(stem_words)
```

add Codeadd Markdown

play_arrow

data

[11]:

	v1	v2
0	0	[go, until, jurong, point, crazi, avail, onli,...
1	0	[ok, lar, joke, wif, u, oni]
2	1	[free, entri, in, a, wkli, comp, to, win, fa, ...
3	0	[u, dun, say, so, earli, hor, u, c, alreadi, t...
4	0	[nah, i, don, t, think, he, goe, to, usf, he, ...
...	...	...

	v1	v2
5567	1	[thi, is, the, nd, time, we, have, tri, contac...
5568	0	[will, b, go, to, esplanad, fr, home]
5569	0	[piti, wa, in, mood, for, that, so, ani, other...
5570	0	[the, guy, did, some, bitch, but, i, act, like...
5571	0	[rofl, it, true, to, it, name]

5169 rows × 2 columns

Feature Extraction

convert the tokenized words back to text and apply Count Vectorization to transform the text data into numerical format.

#Explanation:

We import the CountVectorizer class from the scikit-learn library.

We convert the tokenized words back to text using data['v2'].apply(lambda x: ' '.join(x)). This step is necessary for the Count Vectorizer to work correctly.

We initialize the Count Vectorizer with a maximum of 5000 features using CountVectorizer(max_features=5000). You can adjust this parameter based on your specific needs and computational resources.

We apply the Count Vectorizer to the 'v2' column of the DataFrame, transforming the text data into a numerical format suitable for machine learning models.

If needed, we convert the result to a dense array using features = features.toarray(). This step may be necessary depending on the specific requirements of the downstream modeling process.

```
from sklearn.feature_extraction.text import CountVectorizer


# Convert tokenized words back to text

data['v2'] = data['v2'].apply(lambda x: ' '.join(x))
```

```
# Initialize the Count Vectorizer

count_vectorizer = CountVectorizer(max_features=5000)  # You can adjust max_features as needed



# Apply the vectorizer to the 'v2' column

features = count_vectorizer.fit_transform(data['v2'])



# Convert the result to a dense array (if needed)

features = features.toarray()
```

Train-Test Split:

Split your data into training and testing sets. This allows you to evaluate the performance of your model on data it hasn't seen before.

```
#Explanation:
```

We import the train_test_split function from scikit-learn, which allows us to split the dataset into training and testing sets.

We use train_test_split to split the features (features) and labels (data['v1']) into training and testing sets. The parameter test_size=0.2 indicates that 20% of the data will be used for testing, while 80% will be used for training.

The random_state=42 ensures that the split is reproducible. The same random state will produce the same split each time the code is run.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(features, data['v1'], test_size=0.2, random_state=42)
```

Model Training:

Train your chosen model on the training data.

#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.

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

[15]:

```
MultinomialNB
```

```
MultinomialNB()
```