**NAANMUDHALVAN-IBM SKILL**

**ARTIFICIAL INTELLIGENCE**

**GROUP PROJECT**

**Project Title\* Build a smarter AI-powered spam classifier**

**Phase -V. Submission**

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Group Members  Name | **Naan Mudhalvan** ID | Email ID |
| 1 | AROORAN B | au820321106007 | Arooranbalachandar16@gmail.com |
| 2 | SAKTHI GANESH P | au820321106032 | Sakthiganesh343@gmail.com |
| 3 | YOGESHWARAN | au820321106041 | yogeshmaddy692@gmail.com |
| 4 | MANO BALAN B | au820321106025 | balathandayutam26@gmail.com |

**Build a smarter AI spam classifier using python**

**Introduction**

The upsurge in the volume of unwanted emails called spam has created an intense need for the development of more dependable and robust antispam filters. Any promotional messages or advertisements that end up in our inbox can be categorised as spam as they don’t provide any value and often irritates us.

Overview of the Dataset used

We will make use of the SMS spam classification data.

The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged according to being ham (legitimate) or spam.

The data was obtained from UCI’s Machine Learning Repository, alternatively, I have also uploaded the dataset and completed Jupiter notebook onto my GitHub repo.

**Data processing:**

* Import the required packages
* Loading the Dataset
* Remove the unwanted data columns
* Preprocessing and Exploring the Dataset
* Build word cloud to see which message is spam and which is not.
* Remove the stop words and punctuations
* Convert the text data into vectors

**Building a sms spam classification model:**

* Split the data into train and test sets
* Use Sklearn built-in classifiers to build the models
* Train the data on the model
* Make predictions on new data

**PROGRAM:**

Import the required packages

%matplotlib inline

Import matplotlib.pyplot as plt

Import csv

Import sklearn

Import pickle

From wordcloud import WordCloud

Import pandas as pd

Import numpy as np

Import nltk

From nltk.corpus import stopwords

From sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer

From sklearn.tree import DecisionTreeClassifier

From sklearn.model\_selection import GridSearchCV,train\_test\_split,StratifiedKFold,cross\_val\_score,learning\_curve

Data = pd.read\_csv(‘dataset/spam.csv’, encoding=’latin-1’)

Data.head()

Data = data.drop([“Unnamed: 2”, “Unnamed: 3”, “Unnamed: 4”], axis=1)

Data = data.rename(columns={“v2” : “text”, “v1”:”label”})

Data[1990:2000]

now that the data is looking pretty, let’s move on.

Data[‘label’].value\_counts()

**OUTPUT**

Ham 4825

Spam 747

Name: label, dtype: int64

Preprocessing and Exploring the Dataset

If you are completely new to NLTK and Natural Language Processing(NLP) I would recommend checking out this short article before continuing. Introduction to Word Frequencies in NLP

# Import nltk packages and Punkt Tokenizer Models

Import nltk

Nltk.download(“punkt”)

Import warnings

Warnings.filterwarnings(‘ignore’)

Build word cloud to see which message is spam and which is not

Ham words are the opposite of spam in this dataset, yeah I also don’t have any clue why it is so.

Ham\_words = ‘’

Spam\_words = ‘’

# Creating a corpus of spam messages

For val in data[data[‘label’] == ‘spam’].text:

Text = val.lower()

Tokens = nltk.word\_tokenize(text)

For words in tokens:

Spam\_words = spam\_words + words + ‘ ‘

# **Creating a corpus of ham messages**

For val in data[data[‘label’] == ‘ham’].text:

Text = text.lower()

Tokens = nltk.word\_tokenize(text)

For words in tokens:

Ham\_words = ham\_words + words + ‘ ‘

Let’s use the above functions to create Spam word cloud and ham word cloud.

Spam\_wordcloud = WordCloud(width=500, height=300).generate(spam\_words)

Ham\_wordcloud = WordCloud(width=500, height=300).generate(ham\_words)

#Spam Word cloud

Plt.figure( figsize=(10,8), facecolor=’w’)

Plt.imshow(spam\_wordcloud)

Plt.axis(“off”)

Plt.tight\_layout(pad=0)

Plt.show()

#Spam Word cloud

Plt.figure( figsize=(10,8), facecolor=’w’)

Plt.imshow(spam\_wordcloud)

Plt.axis(“off”)

Plt.tight\_layout(pad=0)

How to build a Spam Classifier in python and sklearn

**Build a Spam Classifier in python banner**

Introduction

The upsurge in the volume of unwanted emails called spam has created an intense need for the development of more dependable and robust antispam filters. Any promotional messages or advertisements that end up in our inbox can be categorised as spam as they don’t provide any value and often irritates us.

Overview of the Dataset used

We will make use of the SMS spam classification data.

The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged according to being ham (legitimate) or spam.

The data was obtained from UCI’s Machine Learning Repository, alternatively, I have also uploaded the dataset and completed Jupiter notebook onto my GitHub repo.

In this article, we’ll discuss:

Data processing

Import the required packages

Loading the Dataset

Remove the unwanted data columns

Preprocessing and Exploring the Dataset

Build word cloud to see which message is spam and which is not.

Remove the stop words and punctuations

Convert the text data into vectors

Building a sms spam classification model

Split the data into train and test sets

Use Sklearn built-in classifiers to build the models

Train the data on the model

Make predictions on new data

Import the required packages

%matplotlib inline

Import matplotlib.pyplot as plt

Import csv

Import sklearn

Import pickle

From wordcloud import WordCloud

Import pandas as pd

Import numpy as np

Import nltk

From nltk.corpus import stopwords

From sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer

From sklearn.tree import DecisionTreeClassifier

From sklearn.model\_selection import GridSearchCV,train\_test\_split,StratifiedKFold,cross\_val\_score,learning\_curve

Please note! You might find that I have reimported some of these packages again later in the article, it is just for ease of use if I ever have to use those code blocks again in future projects, you may omit those.

Loading the Dataset

Data = pd.read\_csv(‘dataset/spam.csv’, encoding=’latin-1’)

Data.head()

Df-head

Removing unwanted columns

From the above figure, we can see that there are some unnamed columns and the label and text column name is not intuitive so let’s fix those in this step.

Data = data.drop([“Unnamed: 2”, “Unnamed: 3”, “Unnamed: 4”], axis=1)

Data = data.rename(columns={“v2” : “text”, “v1”:”label”})

Data[1990:2000]

Pretty-df

Now that the data is looking pretty, let’s move on.

Data[‘label’].value\_counts()

**OUTPUT**

Ham 4825

Spam 747

Name: label, dtype: int64

Preprocessing and Exploring the Dataset

If you are completely new to NLTK and Natural Language Processing(NLP) I would recommend checking out this short article before continuing. Introduction to Word Frequencies in NLP

**Import nltk packages and Punkt Tokenizer Models**

Import nltk

Nltk.download(“punkt”)

Import warnings

Warnings.filterwarnings(‘ignore’)

Build word cloud to see which message is spam and which is not

Ham words are the opposite of spam in this dataset, yeah I also don’t have any clue why it is so.

Ham\_words = ‘’

Spam\_words = ‘’

# Creating a corpus of spam messages

For val in data[data[‘label’] == ‘spam’].text:

Text = val.lower()

Tokens = nltk.word\_tokenize(text)

For words in tokens:

Spam\_words = spam\_words + words + ‘ ‘

**Creating a corpus of ham messages**

For val in data[data[‘label’] == ‘ham’].text:

Text = text.lower()

Tokens = nltk.word\_tokenize(text)

For words in tokens:

Ham\_words = ham\_words + words + ‘ ‘

Let’s use the above functions to create Spam word cloud and ham word cloud.

Spam\_wordcloud = WordCloud(width=500, height=300).generate(spam\_words)

Ham\_wordcloud = WordCloud(width=500, height=300).generate(ham\_words)

#Spam Word cloud

Plt.figure( figsize=(10,8), facecolor=’w’)

Plt.imshow(spam\_wordcloud)

Plt.axis(“off”)

Plt.tight\_layout(pad=0)

Plt.show()

Spam-word-cloud

#Creating Ham wordcloud

Plt.figure( figsize=(10,8), facecolor=’g’)

Plt.imshow(ham\_wordcloud)

Plt.axis(“off”)

Plt.tight\_layout(pad=0)

Plt.show()

Ham-word-cloud

From the spam word cloud, we can see that “free” is most often used in spam.

Now, we can convert the spam and ham into 0 and 1 respectively so that the machine can understand.

Data = data.replace([‘ham’,’spam’],[0, 1])

Data.head(10)

Label-head

Removing punctuation and stopwords from the messages

Punctuation and stop words do not contribute anything to our model, so we have to remove them. Using NLTK library we can easily do it.

Import nltk

Nltk.download(‘stopwords’)

**Remove the punctuations and stopwords**

Import string

Def text\_process(text):

Text = text.translate(str.maketrans(‘’, ‘’, string.punctuation))

Text = [word for word in text.split() if word.lower() not in stopwords.words(‘english’)]

Return “ “.join(text)

Data[‘text’] = data[‘text’].apply(text\_process)

Data.head()

Removed-stopwords

Now, create a data frame from the processed data before moving to the next step.

Text = pd.DataFrame(data[‘text’])

Label = pd.DataFrame(data[‘label’])

Converting words to vectors

We can convert words to vectors using either Count Vectorizer or by using TF-IDF Vectorizer.

TF-IDF is better than Count Vectorizers because it not only focuses on the frequency of words present in the corpus but also provides the importance of the words. We can then remove the words that are less important for analysis, hence making the model building less complex by reducing the input dimensions.

I have included both methods for your reference.

Converting words to vectors using Count Vectorizer

## Counting how many times a word appears in the dataset

From collections import Counter

Total\_counts = Counter()

For I in range(len(text)):

For word in text.values[i][0].split(“ “):

Total\_counts[word] +=

Print(“Total words in data set: “, len(total\_counts))

**OUTPUT**

Total words in data set: 11305

# Sorting in decreasing order (Word with highest frequency appears first)

Vocab = sorted(total\_counts, key=total\_counts.get, reverse=True)

Print(vocab[:60])

**OUTPUT**

[‘u’, ‘2’, ‘call’, ‘U’, ‘get’, ‘Im’, ‘ur’, ‘4’, ‘ltgt’, ‘know’, ‘go’, ‘like’, ‘don’t’, ‘come’, ‘got’, ‘time’, ‘day’, ‘want’, ‘Ill’, ‘lor’, ‘Call’, ‘home’, ‘send’, ‘going’, ‘one’, ‘need’, ‘Ok’, ‘good’, ‘love’, ‘back’, ‘n’, ‘still’, ‘text’, ‘im’, ‘later’, ‘see’, ‘da’, ‘ok’, ‘think’, ‘Ì’, ‘free’, ‘FREE’, ‘r’, ‘today’, ‘Sorry’, ‘week’, ‘phone’, ‘mobile’, ‘cant’, ‘tell’, ‘take’, ‘much’, ‘night’, ‘way’, ‘Hey’, ‘reply’, ‘work’, ‘make’, ‘give’, ‘new’]

# Mapping from words to index

Vocab\_size = len(vocab)

Word2idx = {}

#print vocab\_size

For I, word in enumerate(vocab):

Word2idx[word] = I

# Text to Vector

Def text\_to\_vector(text):

Word\_vector = np.zeros(vocab\_size)

For word in text.split(“ “):

If word2idx.get(word) is None:

Continue

Else:

Word\_vector[word2idx.get(word)] += 1

Return np.array(word\_vector)

# Convert all titles to vectors

Word\_vectors = np.zeros((len(text), len(vocab)), dtype=np.int\_)

For I, (\_, text\_) in enumerate(text.iterrows()):

Word\_vectors[i] = text\_to\_vector(text\_[0])

Word\_vectors.shape

**OUTPUT**

(5572, 11305)

Predictions using TFIDF Vectorizer algorithm

Pred\_scores\_word\_vectors

**PROGRAM**

[(‘SVC’, [0.9784688995215312]),

(‘KN’, [0.9330143540669856]),

(‘NB’, [0.9880382775119617]),

(‘DT’, [0.9605263157894737]),

(‘LR’, [0.9533492822966507]),

(‘RF’, [0.9796650717703349])]

Model predictions

#write functions to detect if the message is spam or not

Def find(x):

If x == 1:

Print (“Message is SPAM”)

Else:

Print (“Message is NOT Spam”)

Newtext = [“Free entry”]

Integers = vectorizer.transform(newtext)

X = mnb.predict(integers)

Find(x)

**OUTPUT**

Message is SPAM

**Conclusion**:

In conclusion, a robust and effective spam-based classifier is a crucial tool in today's digital landscape to filter out unwanted and potentially harmful content. By implementing advanced machine learning techniques and continually updating the classifier with new data, we can significantly enhance its accuracy and performance. However, it's essential to balance accuracy with false positives to ensure that legitimate messages aren't mistakenly classified as spam. Regular monitoring and fine-tuning of the classifier are key to maintaining its effectiveness in an ever-evolving online environment.