SPAM CLASSIFIER

Build Spam Classifier Model

Building our spam classifier by:

Selecting a machine learning algorithm

Training the model

Evaluating its performance

Solution:

Building a spam classifier typically involves the following steps:

Selecting a Machine Learning Algorithm:

Choose a suitable machine learning algorithm for text classification. Common choices include Naïve Bayes, Support Vector Machines, or more advanced methods like neural networks.

Training the Model:

Collect and prepare a labeled dataset of emails, with spam and non-spam labels.

Preprocess the text data, which may include tasks like tokenization, stop word removal, and stemming.

Split the dataset into a training set and a testing/validation set.

Use the training set to train your chosen machine learning algorithm on the preprocessed data.

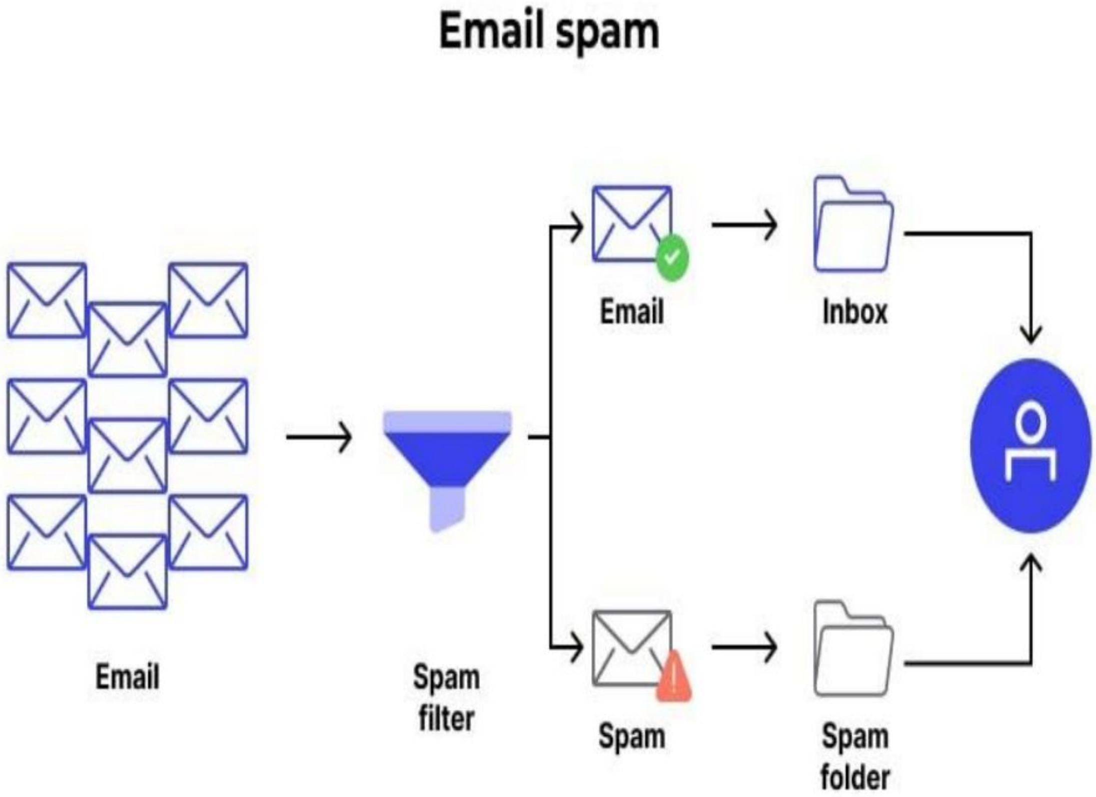
Evaluating its Performance:

Evaluate the model’s performance on the testing/validation set using appropriate metrics like accuracy, precision, recall, and F1-score.

Make adjustments to your model or feature engineering if the performance is not satisfactory.

Consider techniques like cross-validation to ensure robust performance assessment.

These steps are crucial for building an effective spam classifier. The choice of algorithm and the quality of data preprocessing can significantly impact the model's performance.



Spam classifier is a type of software or system that is designed to automatically identify and filter out unwanted or irrelevant messages, often referred to as “spam,” from a stream of incoming data, such as emails, text messages, comments, or any other form of digital communication. The primary goal of a spam classifier is to separate legitimate, wanted messages from undesirable ones. Here’s how it works:

Data Collection: The classifier first needs a dataset of messages that are labeled as either spam or not spam (ham). This dataset is used to train the classifier.

Feature Extraction: The classifier extracts relevant features from the messages, such as keywords, sender information, message structure, and more. These features are used to make decisions about whether a message is spam.

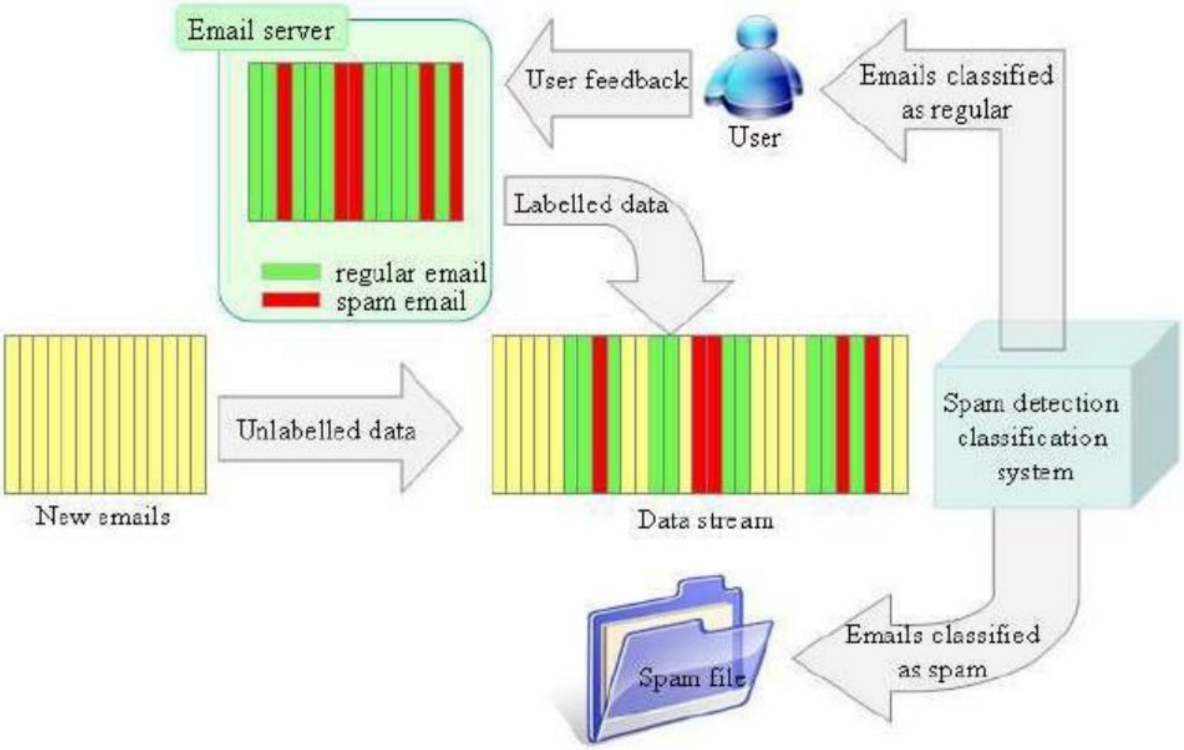
Training: Using the labeled dataset, the classifier learns patterns and relationships between the features and the spam/ham labels. Common machine learning algorithms, such as Naïve Bayes, Support Vector Machines, or deep learning models like neural networks, can be used for this training process.

Classification: Once trained, the spam classifier can be applied to incoming messages. It analyzes the features of each message and predicts whether it’s spam or not.

Threshold Setting: To control the trade-off between false positives (classifying a legitimate message as spam) and false negatives (failing to classify spam), a threshold can be set. Adjusting this threshold allows you to fine-tune the classifier’s behavior.

Filtering: Messages classified as spam can be filtered out or placed in a separate folder, while legitimate messages are delivered to the inbox or the desired location.

Spam classifiers have become an essential part of modern communication systems, as they help users manage the overwhelming amount of spam content that can inundate their inboxes or online platforms. These classifiers continually evolve to adapt to new spamming techniques and emerging threats in the digital space.



Program:

```
Import numpy as np

Import pandas as pd

Import matplotlib.pyplot as plt

Import seaborn as sns

Import tensorflow as tf

From tensorflow import keras

From tensorflow.keras import layers

From sklearn.model_selection import train_test_split

From sklearn.feature_extraction.text import TfidfVectorizer

From sklearn.naive_bayes import MultinomialNB

From sklearn.metrics import classification_report,accuracy_score

From sklearn.metrics import confusion_matrix

From tensorflow.keras.layers import TextVectorization

From sklearn.metrics import precision_score, recall_score, f1_score

Import tensorflow_hub as hub
```

Build a Spam Classifier in 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.

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_cur
```

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,

```
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] += 1


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)
```

Converting words to vectors using TF-IDF Vectorizer

```
#convert the text data into vectors
From sklearn.feature_extraction.text import TfidfVectorizer

Vectorizer = TfidfVectorizer()
Vectors = vectorizer.fit_transform(data['text'])
Vectors.shape
```

```
# OUTPUT
(5572, 9376)
```

```
#features = word_vectors
Features = vectors
```

Splitting into training and test set

```
#split the dataset into train and test set
X_train, X_test, y_train, y_test = train_test_split(features, data['label'], test_size=0.15,
random_state=111)
```

Classifying using sklearn's pre-built classifiers

In this step we will use some of the most popular classifiers out there and compare their results.

Classifiers used:

Spam classifier using logistic regression

Email spam classification using Support Vector Machine(SVM)

Spam classifier using naïve bayes

Spam classifier using decision tree

Spam classifier using K-Nearest Neighbor(KNN)

Spam classifier using Random Forest Classifier

We will make use of sklearn library. This amazing library has all of the above algorithms we just have to import them and it is as easy as that. No need to worry about all the maths and statistics behind it.

```
#import sklearn packages for building classifiers
```

```
From sklearn.linear_model import LogisticRegression
From sklearn.svm import SVC
From sklearn.naive_bayes import MultinomialNB
From sklearn.tree import DecisionTreeClassifier
From sklearn.neighbors import KNeighborsClassifier
From sklearn.ensemble import RandomForestClassifier
From sklearn.metrics import accuracy_score
#initialize multiple classification models
Svc = SVC(kernel=’sigmoid’, gamma=1.0)
Knc = KNeighborsClassifier(n_neighbors=49)
Mnb = MultinomialNB(alpha=0.2)
Dtc = DecisionTreeClassifier(min_samples_split=7, random_state=111)
Lrc = LogisticRegression(solver=’liblinear’, penalty=’l1’)
Rfc = RandomForestClassifier(n_estimators=31, random_state=111)
#create a dictionary of variables and models
Clfs = {‘SVC’ : svc,’KN’ : knc, ‘NB’: mnb, ‘DT’: dtc, ‘LR’: lrc, ‘RF’: rfc}
#fit the data onto the models
Def train(clf, features, targets):
   Clf.fit(features, targets)


Def predict(clf, features):
   Return (clf.predict(features))
Pred_scores_word_vectors = []
For k,v in clfs.items():
   Train(v, X_train, y_train)
   Pred = predict(v, X_test)
   Pred_scores_word_vectors.append((k, [accuracy_score(y_test , pred)]))
```

Predictions using TFIDF Vectorizer algorithm

```
Pred_scores_word_vectors
```

```
# OUTPUT

[(‘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
```

Checking Classification Results with Confusion Matrix

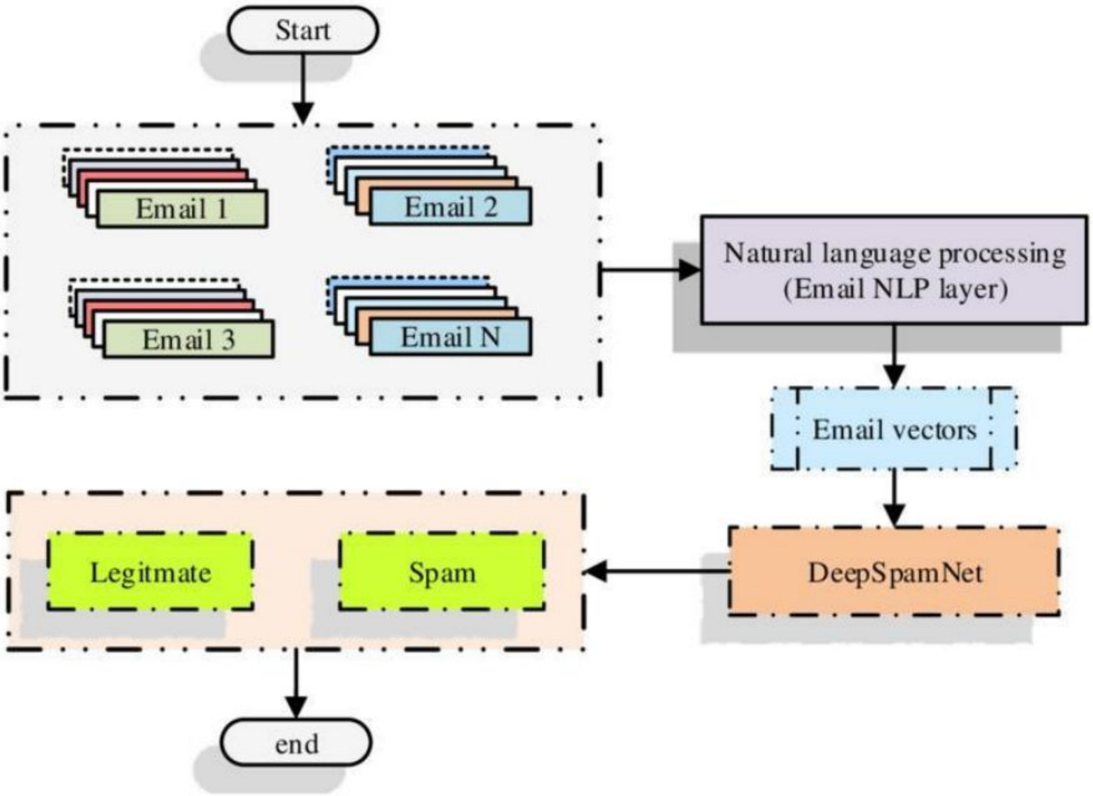
If you are confused about the confusion matrix, read this small article before proceeding – The ultimate guide to confusion matrix in machine learning

```
From sklearn.metrics import confusion_matrix

Import seaborn as sns

# Naïve Bayes
```

```
Y_pred_nb = mnb.predict(X_test)

Y_true_nb = y_test

Cm = confusion_matrix(y_true_nb, y_pred_nb)

F, ax = plt.subplots(figsize =(5,5))

Sns.heatmap(cm,annot = True,linewidths=0.5,linecolor=”red”,fmt = “.0f”,ax=ax)

Plt.xlabel(“y_pred_nb”)

Plt.ylabel(“y_true_nb”)

Plt.show()
```

Confusion matrix

From the confusion matrix, we can see that the Naïve Bayes model is balanced. That’s it !! we have successfully created a spam classifier. 🎉🎉


Start
Email 1
Email 2
Email 3
Email N
Natural language processing
(Email NLP layer)
Email vectors
DeepSpamNet
Legitmate
Spam
end