**SPAM CLASSIFIER**

**Phase 5 : Project Documentation**

**Problem definitions:**

The primary goal of the project is to build a system capable of classifying messages into two categories:

* Spam messages:

Any kind of unwanted, unsolicited digital communication that gets sent out in bulk.

* Non-spam messages:

These are legitimate messages.

The system should be designed in such a way that it can classify the messages in Emails and Text messages.

The system should be designed using Natural Language processing(NLP) and Machine learning.

* Natural Language Processing:

This technique will process and analyze the textual content of messages.

* Machine learning:

This is the fundamental component of a spam classifier, as it enables the system to learn and identify patterns and characteristics of spam and non spam messages.

**Design thinking:**

1. Create an empty list that you will use to store the preprocessed messages.
2. Create a loop to process every message in the MESSAGE column of the dataset.
3. Remove all the non-alphanumeric characters.
4. Convert the message to lowercase.
5. Split the text into words.
6. Remove the stopwords and lemmatize the words.
7. Convert the words back into sentences.
8. Append the preprocessed message into the corpus list.

**Feature engineering using the TF-IDF technique:**

* Feature engineering is the process of converting raw data features into new features suited for machine learning models.
* The Term Frequency-Inverse Document Frequency(TF-IDF) works by assigning weights to words in a document based on how frequently they appear. TF-IDF gives words that appear frequently in a document but are rare in the corpus higher weight. This allows machine learning algorithms to better understand the meaning of the text.

3. Creating and Training the Model:

1. Start by creating and initializing a naive bayes model using the scikit-learn multinomial NB class.
2. Fit the training data, allowing the model to train on the training set
3. Then make predictions on the training and testing sets using the predict method.
4. These predictions will help you evaluate your model.

4. Model evaluation:

1. Evaluate the performance of the model using the classification report function from scikit-learn.
2. Pass the training set prediction and the actual training set labels as input.
3. Do the same for the test set.

**Dataset used:**

A large number of messages containing both spam and non spam messages are created and the dataset file is uploaded with it.

**Data preprocessing Steps:**

The required packages such as matplotlib, CSV, sklearn are imported to the code. Then convert all the characters in the dataset to either upper or lower case, removing numbers, punctuation and end words.

**Feature extraction technique:**

The algorithm always expects the input to be integers/floats, so we need to have some feature extraction layer in the middle to convert the words to integers/floats. So we used the count vectorizer technique.

we need to input all the training data into CountVectorizer and the CountVectorizer will keep a dictionary of every word and its respective id and this id will relate to the word count of this word inside this whole training set.

**Machine learnings algorithms:**

SVM, the support vector machine algorithm, is a linear model for classification and regression. The idea of SVM is simple, the algorithm creates a line, or a hyperplane, which separates the data into classes. SVM can solve both linear and non-linear problems.

**Model training**:

We use a train-test split method to train our email spam detector to recognize and categorize spam emails. The train-test split is a technique for evaluating the performance of a machine learning algorithm. We can use it for either classification or regression of any supervised learning algorithm.

The procedure involves taking a dataset and dividing it into two separate datasets. The first dataset is used to fit the model and is referred to as the training dataset. For the second dataset, the test dataset, we provide the input element to the model. Finally, we make predictions, comparing them against the actual output.

**Evaluation metrics:**

We used the word cloud and confusion Matrix to evaluate the results from the design created.

Word clouds - Word clouds or tag clouds are graphical representations of word frequency that give greater prominence to words that appear more frequently in a source text. The larger the word in the visual the more common the word was in the document(s).

Confusion Matrix - A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the total number of target classes. The matrix compares the actual target values with those predicted by the machine learning model.

**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

Loading the Dataset

data = pd.read\_csv('dataset/spam.csv', encoding='latin-1')

data.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]

data['label'].value\_counts()

# OUTPUT

ham 4825

spam 747

Name: label, dtype: int64

**Preprocessing and Exploring the Dataset**

# 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 = ''

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()

#Creating Ham wordcloud

plt.figure( figsize=(10,8), facecolor='g')

plt.imshow(ham\_wordcloud)

plt.axis("off")

plt.tight\_layout(pad=0)

plt.show()

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)

**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 the 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()

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 Countvectorizer 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.

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**

#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

**Evaluating Classification Results with Confusion Matrix**

from sklearn.metrics import confusion\_matrix

import seaborn as sns

# Naive 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()

**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*

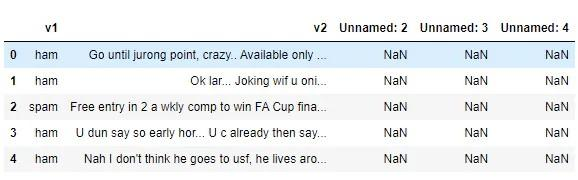
*from sklearn.tree import DecisionTreeClassifier*

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

**Loading the Dataset**

*data = pd.read\_csv('dataset/spam.csv', encoding='latin-1')*

*data.head()*



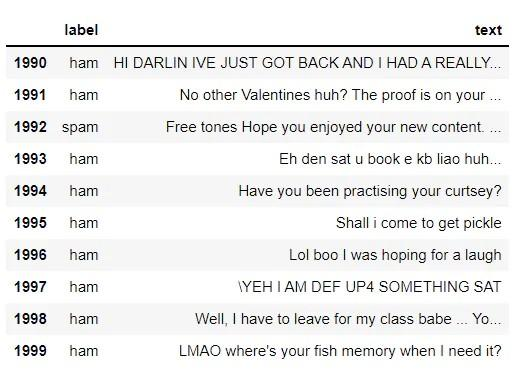
**Removing unwanted columns**

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]*



*data['label'].value\_counts()*

# OUTPUT

ham 4825

spam 747

Name: label, dtype: int64

**Preprocessing and Exploring the Dataset**

*# 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 = ''*

*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()*

*#Creating Ham wordcloud*

*plt.figure( figsize=(10,8), facecolor='g')*

*plt.imshow(ham\_wordcloud)*

*plt.axis("off")*

*plt.tight\_layout(pad=0)*

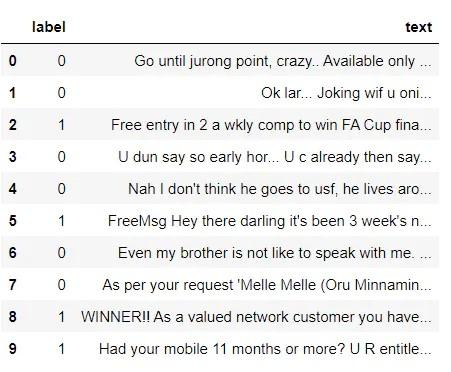
*plt.show()*

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)*



**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 the NLTK library we can easily do it.

*import nltk*

*nltk.download('stopwords')*

*#remove the punctuations and stopwords*

*import string*

*def text\_process(text):*

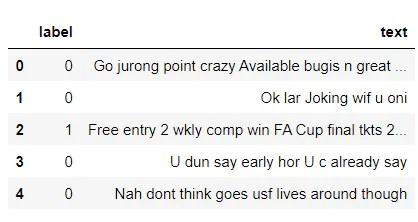
*text = text.translate(str.maketrans('', '', string.punctuation))*

*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()*



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 Countvectorizer 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.

**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**

*#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*

**Evaluating Classification Results with Confusion Matrix**

*from sklearn.metrics import confusion\_matrix*

*import seaborn as sns*

*# Naive Bayes*

*y\_pred\_nb = mnb.predict(X\_test)*

*y\_true\_nb = y\_test*

*cm = confusion\_matrix(y\_true\_nb, y\_pred\_nb)*

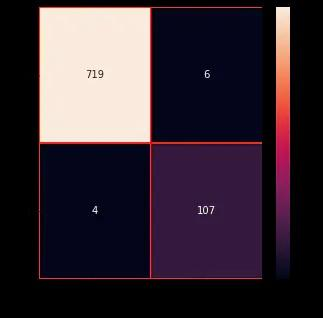
*f, ax = plt.subplots(figsize =(5,5))*

*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()*



from the confusion matrix, we can see that the Naive Bayes model is balanced.