SPAM EMAIL DETECTION

HARSHA SATHISH
AM.EN.U4CSE19123

TABLE OF CONTENTS:

Spam Email Detection

Submitted by:

Harsha Sathish

AM.EN.U4CSE19123

……

❖ Problem Definition

In today's Modern world, E-mail has become the preferred medium for communicating official information. Emails provide efficient and effective ways to transmit all kinds of electronic data. With the rapid increase in email usage, there has also been an increase in the Spam emails. These Spam emails are unsolicited and unwanted junk text, send out in bulk to an indiscriminate recipient list. These emails prevent the user from creating full and sensible use of your time, storage capability and network information measure. It is estimated that spam cost businesses on the order of $100 billion per year.

In this project, we will train the model using the datasets containing spam and non-spam email, and use various Machine Learning Algorithms to enable us to classify the emails.

❖ Datasets

- The datasets used in the Project consists of emails received by the Senior Management of the Enron Corporation which contain

the words and phrases which are commonly used in Spam Emails.

- The Data is Unorganized. Therefore, we need to pre-process the data to obtain the useful information before we apply ML techniques.
- Features:
  - There are approximately 8000 entries in each dataset.
  - There are 4 attributes in the data set:
    - Text
      - The Content of email from various sources
      - Along with content, the data, time and the subject is also present
    - Spam
      - This attribute represents if the email is a spam or ham email.
      - 1 - spam
      - 0 – ham
    - Length
      - This attribute contains the count of characters present in the email
    - Clean Text
      - This attribute contains the “text” after pre-processing i.e. removing punctuation, stop words and stemming

❖ **Text Pre-processing**

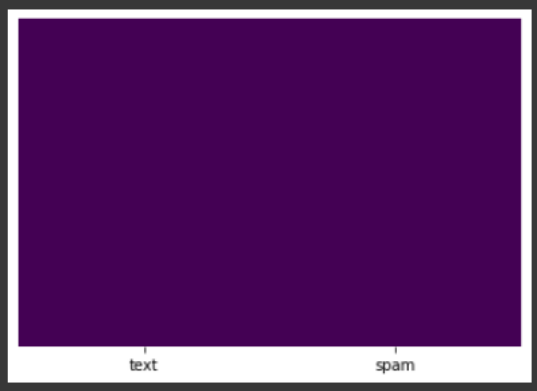
The Email text in the learning data are in plain text format. Therefore, the text should be converted into features that can represent the email. A number of pre-processing steps are performed before an algorithm can be applied.

- Finding columns with null values and removing them
  - Using Heatmaps

    Heatmaps are used to visualize the data and represent it in a 2-dimensional format in form of coloured maps.

    The colour variation shows the client the magnitude of numeric values. Heatmaps also describe about the density and intensity of variables, visualize patterns, variance and anomalies.

```
# To Check missing value

# Import Seaborn
import seaborn as sn

# Heat Map Visualization
sn.heatmap(data.isnull(), cbar=False, yticklabels=False, cmap='viridis');
```

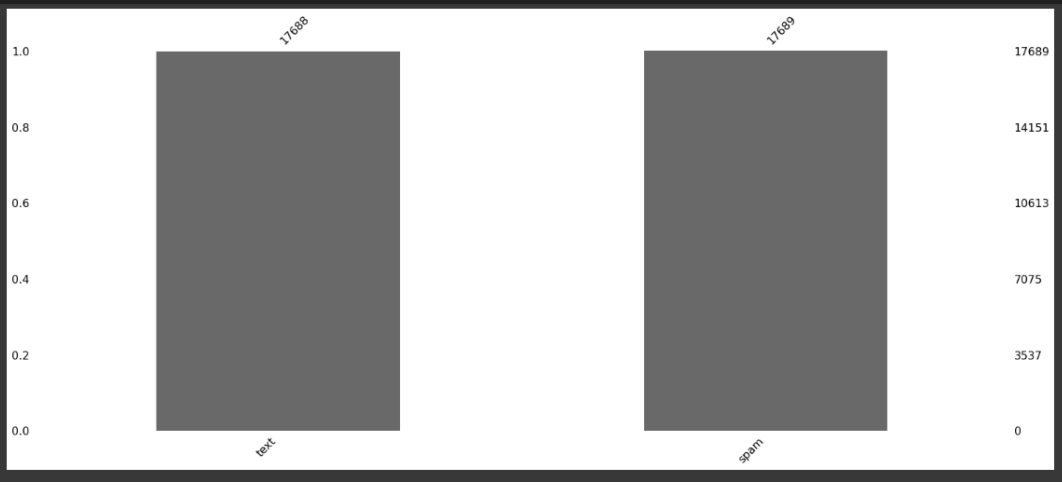


  - Using Missingno Library

    The Missingno Library provides the distribution of missing values in the dataset by Informative Visualization.

    Using the graph, we can check where the missing values are located in each column and check if there is a correlation between missing values of different columns.

```
# Import missingno Library
import missingno as msno

# Plot the Bar Graph
msno.bar(data);
```



- *Plotting the Heat Map*

```
# Plot the Heat Map
msno.heatmap(data);
```

- *Find the number of rows with NaN value*

```
[ ]  # Find the Number of Rows that has Nan Value in it

     data.isnull().sum()

     text    1
     spam    0
     dtype: int64
```

- *Find the number of non-NaN cells for each column or row*

```
     # Find the No of Non NA cells for each column or row

     data.count()

     text    17688
     spam    17689
     dtype: int64
```

- *Check if any column contains only NULL values*

```
# Find the Number of Rows that has Nan Value in it

Null_Data = data.isnull().sum()

# List for storing the Null Column Names

Null_Columns = []


for i in range(len(Null_Data)):

  # If the number of Null Values in the Row is equal to the total number of Records, then it means that the whole column contains NUll value in it.

  if Null_Data[i] == Rows - 1 or Null_Data[i] == Rows:

    Null_Columns.append(Column_Names[i])

# Print all Columns which has only NULL values

print(Null_Columns)

[]
```

It is evident that there is no column in the dataset which contains only null values

- *Drop the columns containing only NULL values*

```
[ ]  # Delete all NULL Columns which has only NULL values

     for i in Null_Columns:

       del data[i]
```

- *Check if any rows contain Null values*

```
[ ]  data.isnull().any()

     text     True
     spam    False
     dtype: bool
```

```
[ ]  data.isnull().sum()

     text    1
     spam    0
     dtype: int64
```

```
[ ]  # Display the Rows which has one or more NULL values in it

     data[data.isnull().any(axis=1)]
```

	text	spam
5112	NaN	1

- *Drop the row which contain Null value*

```
[ ]  data.dropna(inplace=True)
```

```
[ ]  data.isnull().any()
```

```
text    False
spam    False
dtype: bool
```

```
[ ]  print(data.isnull().sum())
```

```
text    0
spam    0
dtype: int64
```

The cell 5112 in text contained NaN value and it is removed.

- Add the “Text Length” column for each records

```
[ ]  # Store the Length of the messages in the New Column with respective to each of the records

     data['Length'] = data['text'].apply(len)

     data['Length'].max()
```

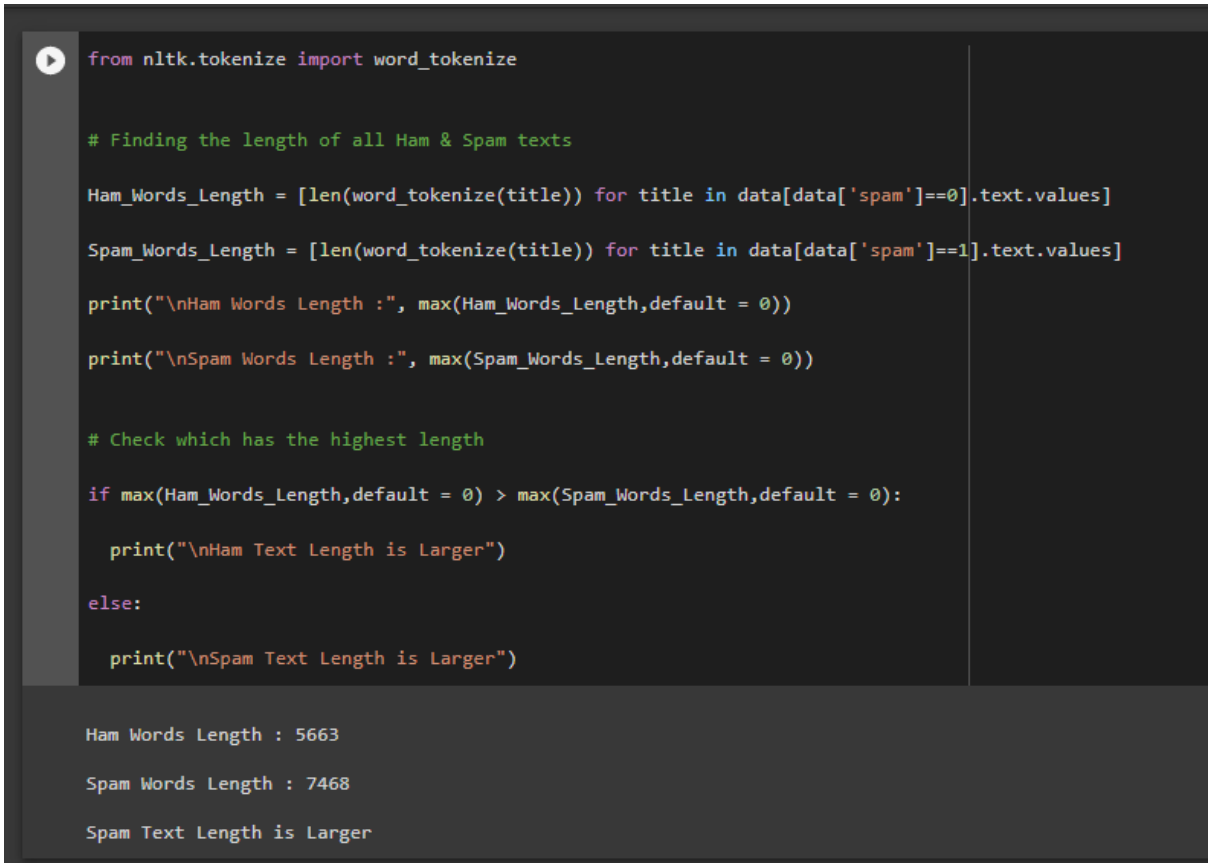
```
31636
```

```
[ ]  data.describe()
```

	spam	Length
count	15552.000000	15552.000000
mean	0.386831	2183.796232
std	0.487040	2791.350372
min	0.000000	1.000000
25%	0.000000	148.000000
50%	0.000000	1645.500000
75%	1.000000	3215.000000
max	1.000000	31636.000000

- Word Tokenization

Tokenization is the splitting of data into smaller units called tokens.



```
from nltk.tokenize import word_tokenize


# Finding the length of all Ham & Spam texts

Ham_Words_Length = [len(word_tokenize(title)) for title in data[data['spam']==0].text.values]

Spam_Words_Length = [len(word_tokenize(title)) for title in data[data['spam']==1].text.values]

print("\nHam Words Length :", max(Ham_Words_Length,default = 0))

print("\nSpam Words Length :", max(Spam_Words_Length,default = 0))


# Check which has the highest length

if max(Ham_Words_Length,default = 0) > max(Spam_Words_Length,default = 0):

  print("\nHam Text Length is Larger")

else:

  print("\nSpam Text Length is Larger")
```

```
Ham Words Length : 5663

Spam Words Length : 7468

Spam Text Length is Larger
```

For a ham email, the maximum number of ham words used is 5663. For a spam email, the maximum number of spam words used is 7468.

It is evident that the spam emails have less words as compared to ham emails.

- Data Cleaning

```
[48] import string
     from nltk.stem.porter import PorterStemmer
     ps = PorterStemmer()

     class Data_Clean():

         def __init__(self):

             pass



         def transform_text(self,text):

             text = text.lower()
             text = nltk.word_tokenize(text)

             y = []
             for i in text:
                 if i.isalnum():
                     y.append(i)

             text = y[:]
             y.clear()

             for i in text:
                 if i not in stopwords.words('english') and i not in string.punctuation:
                     y.append(i)

             text = y[:]
             y.clear()

             for i in text:
                 y.append(ps.stem(i))

             return " ".join(y)


         def Clean(self, U_data):

           C_Data = U_data.apply(self.transform_text)

           return C_Data
```

```
[49] Cleaned_Data = Data_Clean()

     data['Cleaned Text'] = Cleaned_Data.Clean(data['text'])
```

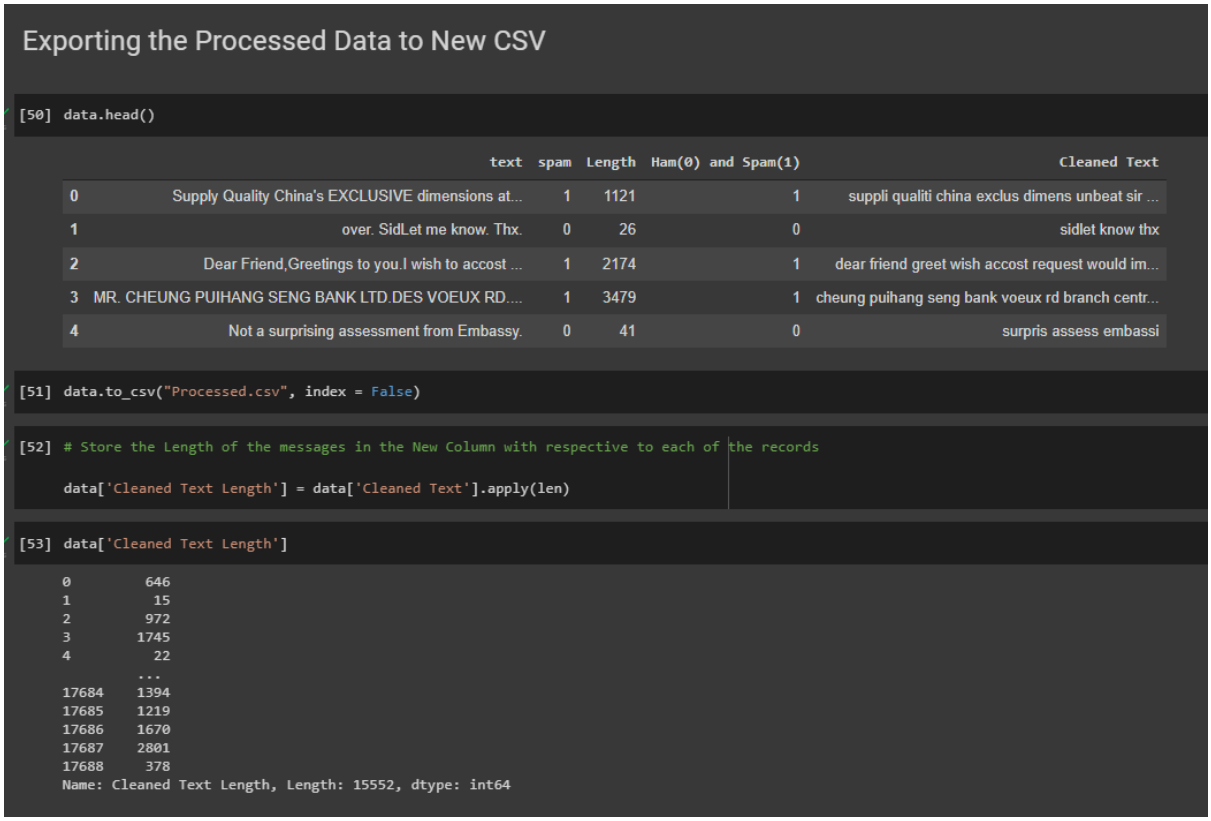
A new column “Cleaned Text” is created which contains the email text, after the removal of punctuation and stopwords.

In the function, first the text is filtered from punctuation marks. Then, the text is checked for presence of stopwords, which are removed.

After data cleaning, the dataset is subjected to Stemming using PorterStemmer.Stemming is the process of obtaining the root word from its inflected form.

The processed text is stored in “Clean Text”

- Exporting the processed data to new CSV



Exporting the Processed Data to New CSV

```
[50] data.head()
```

	text	spam	Length	Ham(0) and Spam(1)	Cleaned Text
0	Supply Quality China's EXCLUSIVE dimensions at...	1	1121	1	suppli qualiti china exclus dimens unbeat sir ...
1	over. SidLet me know. Thx.	0	26	0	sidlet know thx
2	Dear Friend,Greetings to you.I wish to accost ...	1	2174	1	dear friend greet wish accost request would im...
3	MR. CHEUNG PUIHANG SENG BANK LTD.DES VOEUX RD....	1	3479	1	cheung puihang seng bank voeux rd branch centr...
4	Not a surprising assessment from Embassy.	0	41	0	surpris assess embassi

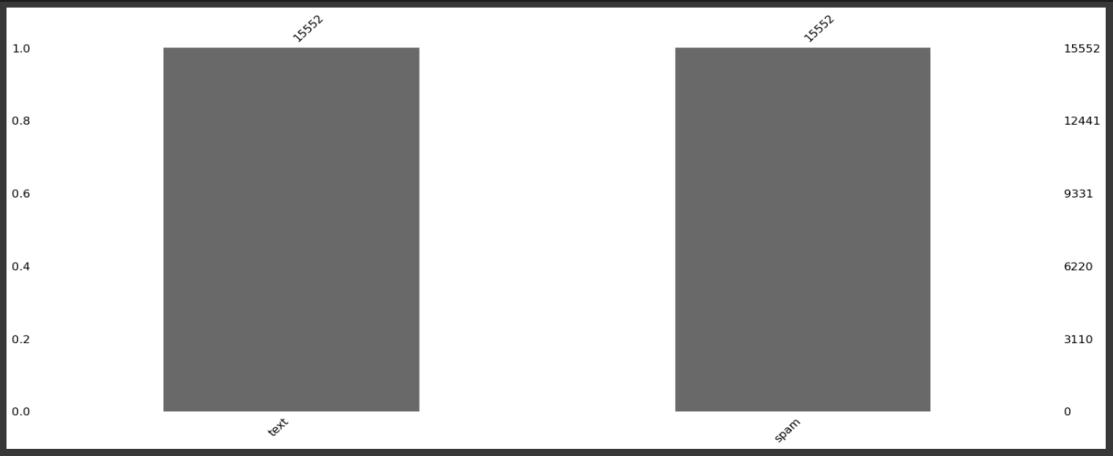
```
[51] data.to_csv("Processed.csv", index = False)

[52] # Store the Length of the messages in the New Column with respective to each of the records

     data['Cleaned Text Length'] = data['Cleaned Text'].apply(len)

[53] data['Cleaned Text Length']

0          646
1           15
2          972
3         1745
4           22
         ...
17684     1394
17685     1219
17686     1670
17687     2801
17688      378
Name: Cleaned Text Length, Length: 15552, dtype: int64
```

The Processed dataset is now converted into a new CSV file namely Processed.csv, which can be downloaded from the runtime.

Data.head() displays the first 5 elements in the dataset

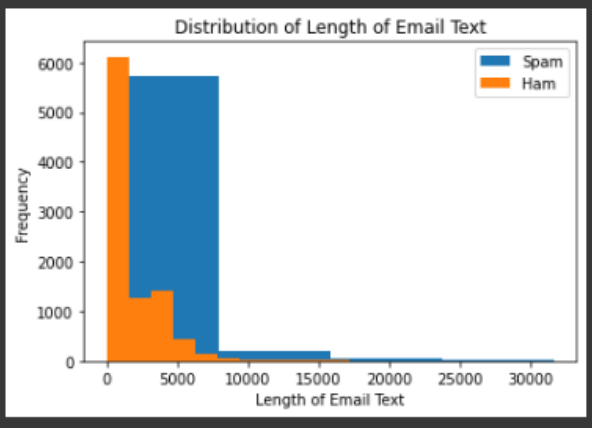
- Data Visualization
  - MSNO Bar Graph to find number of NULL values in dataset

```
[29] # Plot the MSNO Bar Graph to find the number of NULL values
     msno.bar(data);
```



- *Plotting the Length of Spam and Ham Texts Using matplotlib library*

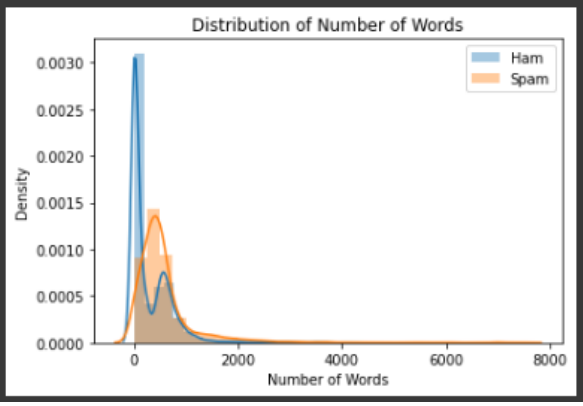
```
[34] # Import Matplotlib Library

     import matplotlib.pyplot as plt


     # Split the Spam & Ham Records

     Spam_Length =  data[data['spam']==1]

     Ham_Length =  data[data['spam']==0]


     # Plot the Length of Spam & Ham Messages

     Spam_Length['Length'].plot(bins=4, kind='hist',label = 'Spam')

     Ham_Length['Length'].plot(bins=20, kind='hist',label = 'Ham')

     plt.title('Distribution of Length of Email Text')

     plt.xlabel('Length of Email Text')

     plt.legend();
```



It is evident that length of characters of ham emails is more as compared to that of spam emails

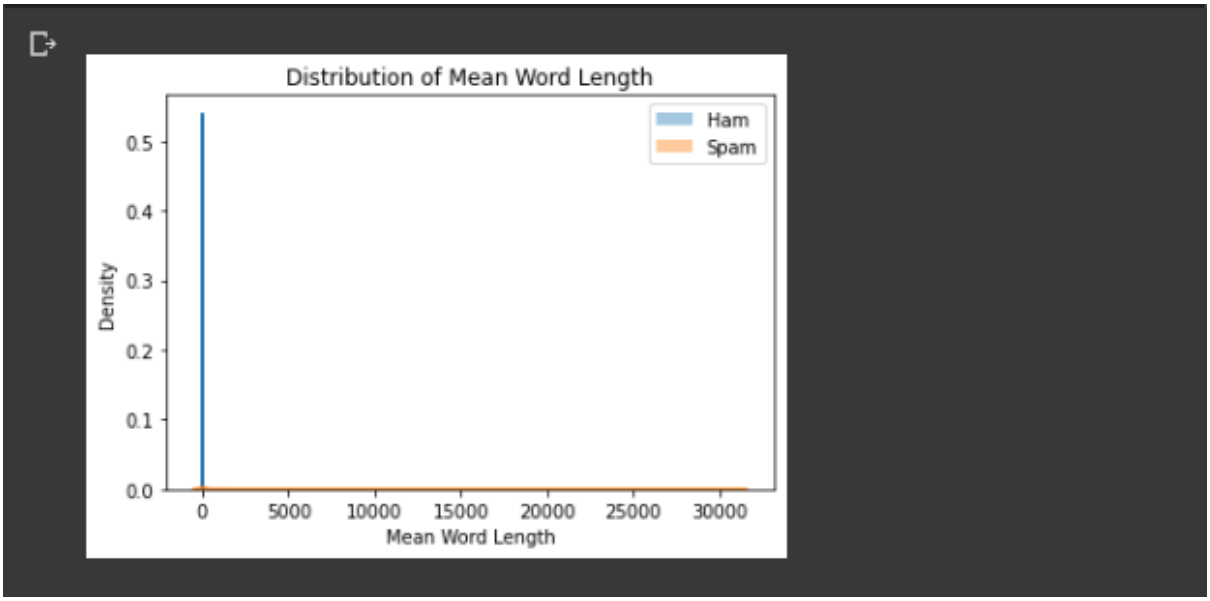
- *Plot the Spam and Ham record length after tokenization using word_tokenize function*

```
[37] # Plot the Spam & Ham record's length after tonkenizing it using word_tokenize function

     ax = sn.distplot(Ham_Words_Length, norm_hist = True, bins = 30, label = 'Ham')

     ax = sn.distplot(Spam_Words_Length, norm_hist = True, bins = 30, label = 'Spam')

     print()

     plt.title('Distribution of Number of Words')

     plt.xlabel('Number of Words')

     plt.legend()

     plt.show();
```



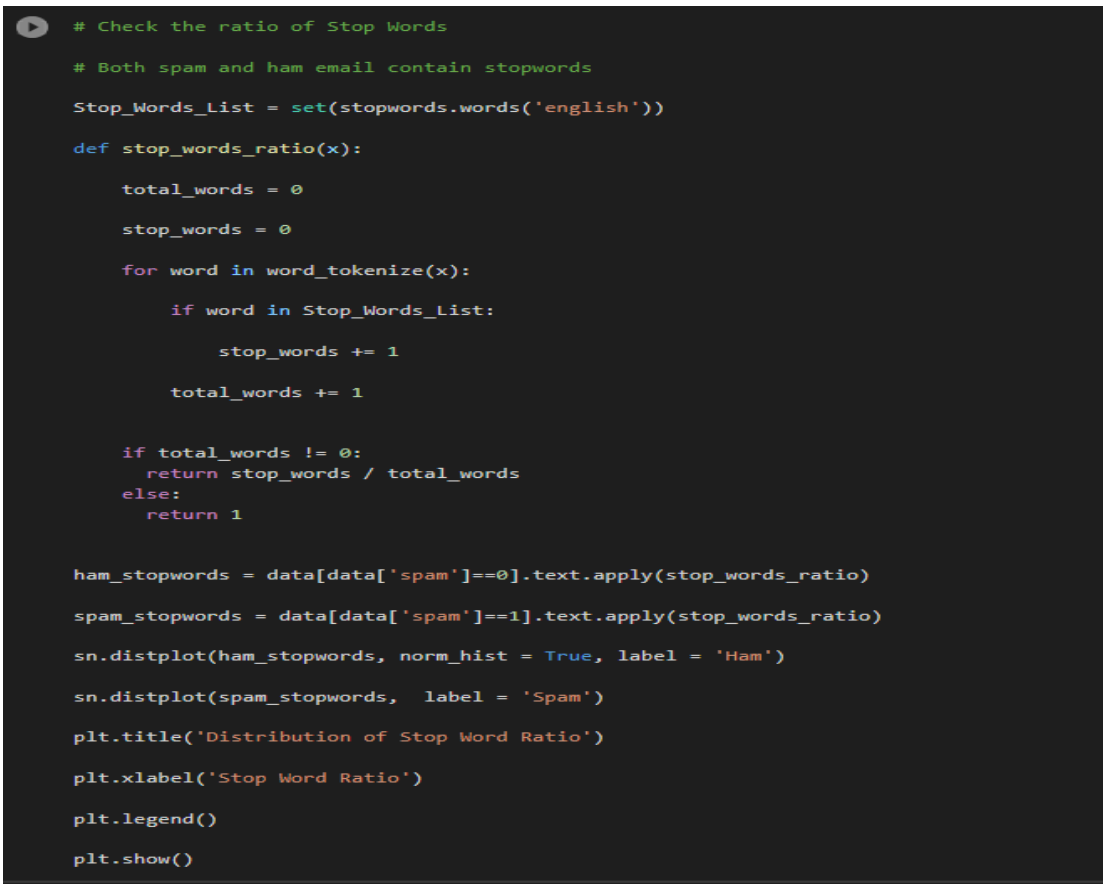
- *Plot the graph of distribution of the mean word length*

```
[ ]  import numpy as np

     # Function to find the Mean Word Length
     def Mean_Word_Length(x):
         length = np.array([])
         for word in word_tokenize(x):
             length = np.append(length, len(word))
         return length.mean()

     Ham_Meanword_Length = data[data['spam']==0].text.apply(Mean_Word_Length)
     Spam_Meanword_Length = data[data['spam']==1].text.apply(Mean_Word_Length)

     # Plot the Graph of Distribution of the Mean Word Length
     sn.distplot(Ham_Meanword_Length, norm_hist = True, bins = 30, label = 'Ham')
     sn.distplot(Spam_Meanword_Length , norm_hist = True, bins = 30, label = 'Spam')
     print()
     plt.title('Distribution of Mean Word Length')
     plt.xlabel('Mean Word Length')
     plt.legend()
     plt.show()
```



This graph plots the distribution of the mean word length of ham and spam emails. X – axis : Spam, Y-axis : Ham

- *Plot the distribution of Stop Words*



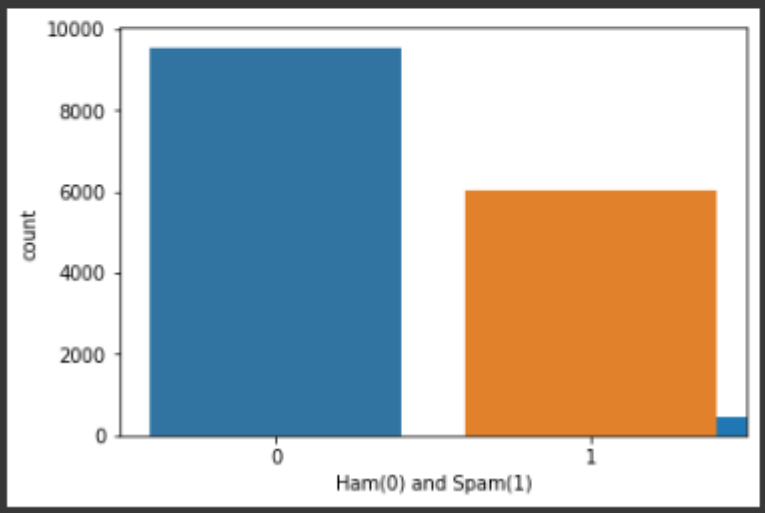
```
# Check the ratio of Stop Words

# Both spam and ham email contain stopwords

Stop_Words_List = set(stopwords.words('english'))

def stop_words_ratio(x):

    total_words = 0

    stop_words = 0

    for word in word_tokenize(x):

        if word in Stop_Words_List:

            stop_words += 1

        total_words += 1


    if total_words != 0:
      return stop_words / total_words
    else:
      return 1


ham_stopwords = data[data['spam']==0].text.apply(stop_words_ratio)

spam_stopwords = data[data['spam']==1].text.apply(stop_words_ratio)

sn.distplot(ham_stopwords, norm_hist = True, label = 'Ham')

sn.distplot(spam_stopwords,  label = 'Spam')

plt.title('Distribution of Stop Word Ratio')

plt.xlabel('Stop Word Ratio')

plt.legend()

plt.show()
```

Spam emails contain stop words with a mean of 0.274

Ham emails contain stop words with a mean of 0.256

- *Plot the count of spam and ham emails*

```
[ ]  # Divide the messages into spam and ham

     ham = data.loc[data['spam']==0]

     spam = data.loc[data['spam']==1]

     spam['Length'].plot(bins=60, kind='hist')

     data['Ham(0) and Spam(1)'] = data['spam']

     sn.countplot(data['Ham(0) and Spam(1)'], label = "Count");
```



To construct a histogram, first we have to “bin” the range of values, that is divide the entire range of values into a series of intervals.

- *Construct a word cloud visualization for the dataset*

```
class Word_Cloud():

    def __init__(self):

        pass

    def variance_column(self, data):

        return variance(data)

    def word_cloud(self, data_frame_column, output_image_file):

        text = " ".join(review for review in data_frame_column)

        stopwords = set(STOPWORDS)

        stopwords.update(["subject"])

        wordcloud = WordCloud(width = 1200, height = 800, stopwords=stopwords, margin=0, max_words = 1000).generate(text)

        plt.imshow(wordcloud, interpolation='bilinear')

        plt.axis("off")

        plt.show()

        wordcloud.to_file(output_image_file)

        return
```

```
[ ] from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

    from PIL import Image

    word_cloud = Word_Cloud()

    word_cloud.word_cloud(ham["text"], "Ham.png")

    word_cloud.word_cloud(spam["text"], "Spam.png")
```

Word cloud is a technique for visualising frequent words in a text where the size of the words represent their frequency. The word cloud provides a visual representation of text data. We use the WordCloud() Library to implement.

- Vectorize the data

```
[ ] from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
    cv = CountVectorizer()
    tfidf = TfidfVectorizer(max_features=3000)

[ ] X = tfidf.fit_transform(data['Cleaned Text'].values.astype('U')).toarray()

[ ] X.shape

    (15552, 3000)

[ ] Y = data['spam'].values

    Y.shape

    (15552,)
```

Word Vectorization is the process to map words or phrases from vocabulary to a corresponding vector or real numbers which is used to find word predictions/similarities.

Here, we are vectorizing the data using the Tf-idf vectorizer or the Term Frequency Inverse Document Frequency. It captures the importance of given word related to other words, that is it transforms text to feature vectors that can be used as an input to estimator.

X is the cleaned data, after tf-ifd vectorization.

- **Python Packages**
  - NumPy

    Numpy stands for Numerical Python, It is a library, which support large, multi dimensional arrays and matrices, along with a large collection of high level mathematical functions to operate on numpy arrays.
  - SciPy

    It is a FOSS library used for scientific computing and technical computing. It contains modules for optimization, linear algebra and special functions.
  - Scikit-learn

    It is a Machine Learning library in Python. It has different classification, regression and clustering algorithms.
  - Pandas

    It is a library which provides various data structures and easy to use data analysis and manipulation tool.
  - NLTK (Natural Langauge Toolkit)

    It is a platform used for building programs that works on human language data for applying in statistical natural language

processing(NLP). It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning.

- Matplotlib

  It is a plotting library in Python. It is cross-platform, data visualization and graphic plotting library. It offers visible open source alternative to MATLAB.

❖ Machine Learning Algorithms

➢ Kth Nearest Neighbour (KNN) [From Scratch]

KNN is a supervised learning algorithm, which is used for both classification and prediction. It is Data Driven, that is little knowledge about distribution of data is required. The complexity of the model also increases with number of training data.
KNN is termed as a Lazy Learning Algorithm. New instances are compared with instances stored in the memory during training, instead of explicit generalization.

Basic KNN Algorithm for Classification:
Input : Training Set D, Test Datapoint d, Parameter k

- Computer distance between test datapoint d and every datapoint in training set D
- Choose k datapoints in training set D that are nearest to the sample d, denote P
- Assign test datapoint d to majority class.

Output: Class Label of test datapoint d

<u>Implementation and Coding Phase of KNN</u>

- Split the dataset into X and Y

```
[ ]  X = tfidf.fit_transform(data['Cleaned Text'].values.astype('U')).toarray()

[ ]  X.shape

     (15552, 3000)

[ ]  Y = data['spam'].values

     Y.shape

     (15552,)
```

- *Split the data into training and testing partitions*

```
[ ]  from sklearn.model_selection import train_test_split

     X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size = 0.2, random_state = 2 )
```

Here, the training set is 20% and testing set is 80%.

- *KNN algorithm from Scratch*

```
[ ]  # K Nearest Neighbors Classification

     from scipy.stats import mode

     class KNN() :

       def __init__( self, K ) :

         self.K = K


       def fit( self, X_train, Y_train ) : # Function to store training set

         self.X_train, self.Y_train = X_train, Y_train

         self.m, self.n = X_train.shape # No of Rows & Columns in Training Data Set


       def predict( self, X_test ) : # Function for prediction

         self.X_test = X_test

         self.m_test, self.n = X_test.shape # No of Rows & Columns in Test Data Set

         Y_predict = np.zeros( self.m_test )

         for i in range( self.m_test ) :

           neighbors = self.find_neighbors( self.X_test[i] ) # Find the K nearest neighbors from current test example

           # most frequent class in K neighbors

           Y_predict[i] = mode( neighbors )[0][0]

         return Y_predict
```

```
def find_neighbors( self, x ) : # Function to find the K nearest neighbors to current test example
  # Calculate all the Euclidean distances between current test example x and training set X_train
  euclidean_distances = np.zeros( self.m )
  for i in range( self.m ) :
    euclidean_distances[i] = self.euclidean( x, self.X_train[i] )
    Y_train_sorted = self.Y_train[euclidean_distances.argsort()] # Sort Y_train according to euclidean_distance_array and store it in Y_train_sorted
  return Y_train_sorted[:self.K]

def euclidean( self, x, x_train ) : # Function to calculate euclidean distance
  return np.sqrt( np.sum( np.square( x - x_train ) ) )
```

- Train the model using the training data partition and predict on test set

```
[ ]  # Training the Model on the Train Data Set

     model = KNN( K = 5 )

     model.fit( X_train, Y_train )

     # Prediction on test set

     Y_pred = model.predict( X_test )
```

Here, value of K is given as 5.

- *Obtain the confusion matrix*

```
[ ]  from sklearn.metrics import confusion_matrix

     print(confusion_matrix(Y_test,Y_pred))

     [[1849   38]
      [ 394  830]]
```

- *Calculate the F1-Score*

```
[ ]  from sklearn.metrics import f1_score

     print(f1_score(Y_test,Y_pred))

     0.7934990439770554
```

F1_score is a measure of model's accuracy on a dataset. It is a way of combining the precision and recall of the medal, and is the harmonic mean of the model's precision and recall.

The f1_score for this model is 79.35%.

- *Calculate the accuracy of the model*

```
[ ]  from sklearn.metrics import accuracy_score

     print(accuracy_score(Y_test,Y_pred))

     0.8611378977820636
```

The accuracy for this model using KNN is 86.12%.

➢ Logistic Regression

Logistic regression is a supervised learning algorithm that uses a logistic funtion to predict the probability of target variable. It gives values which lie between 0 and 1.

Logistic regression is used for solving the classification problems.

In logistic regression, an "S" shaped logistic function is fitted, which predicts the maximum values. The curve indicated the likelihood of

something such as whether the person has diabetes, the cancer is malignant etc.

Logistic regression is used to classify the observations using different types of data and can easily determine the most effective variables for the classification.

The sigmoid function used is a mathematical function which maps any real value into another value within 0 and 1.

Assumptions:

- The dependent variable must be categorical in nature.
- The independent variable should not have multi-collinearity.

Implementation and Coding Phase of Logistic Regression

- Splitting the dataset into X, Y and splitting data into training and testing partitions

```
[58] from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer

     cv = CountVectorizer()

     tfidf = TfidfVectorizer(max_features=3000)

     X = tfidf.fit_transform(data['Cleaned Text'].values.astype('U')).toarray()
     Y = data['spam'].values

     from sklearn.model_selection import train_test_split

     X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size = 0.2, random_state = 2 )

     from sklearn.preprocessing import StandardScaler

     scaler = StandardScaler()

     scaler.fit(X_train)

     X_train = scaler.transform(X_train)

     X_test = scaler.transform(X_test)
```

- *Logistic Regression Implementation*

```
class LogisticRegression:

    def __init__(self, learning_rate=0.01, n_iters=1000):

        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None

    def fit(self, X, y):

        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_iters):

            linear_model = np.dot(X, self.weights) + self.bias
            y_predicted = self._sigmoid(linear_model)
            dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y))
            db = (1 / n_samples) * np.sum(y_predicted - y)
            self.weights -= self.lr * dw
            self.bias -= self.lr * db
```

```
    def predict(self, X):

        linear_model = np.dot(X, self.weights) + self.bias
        y_predicted = self._sigmoid(linear_model)
        y_predicted_cls = [1 if i > 0.5 else 0 for i in y_predicted]
        return np.array(y_predicted_cls)

    def _sigmoid(self, x):

        return 1 / (1 + np.exp(-x))

lr = LogisticRegression()

lr.fit(X_train, Y_train)

Y_pred = lr.predict(X_test)
```

- *Calculate the accuracy of the model*

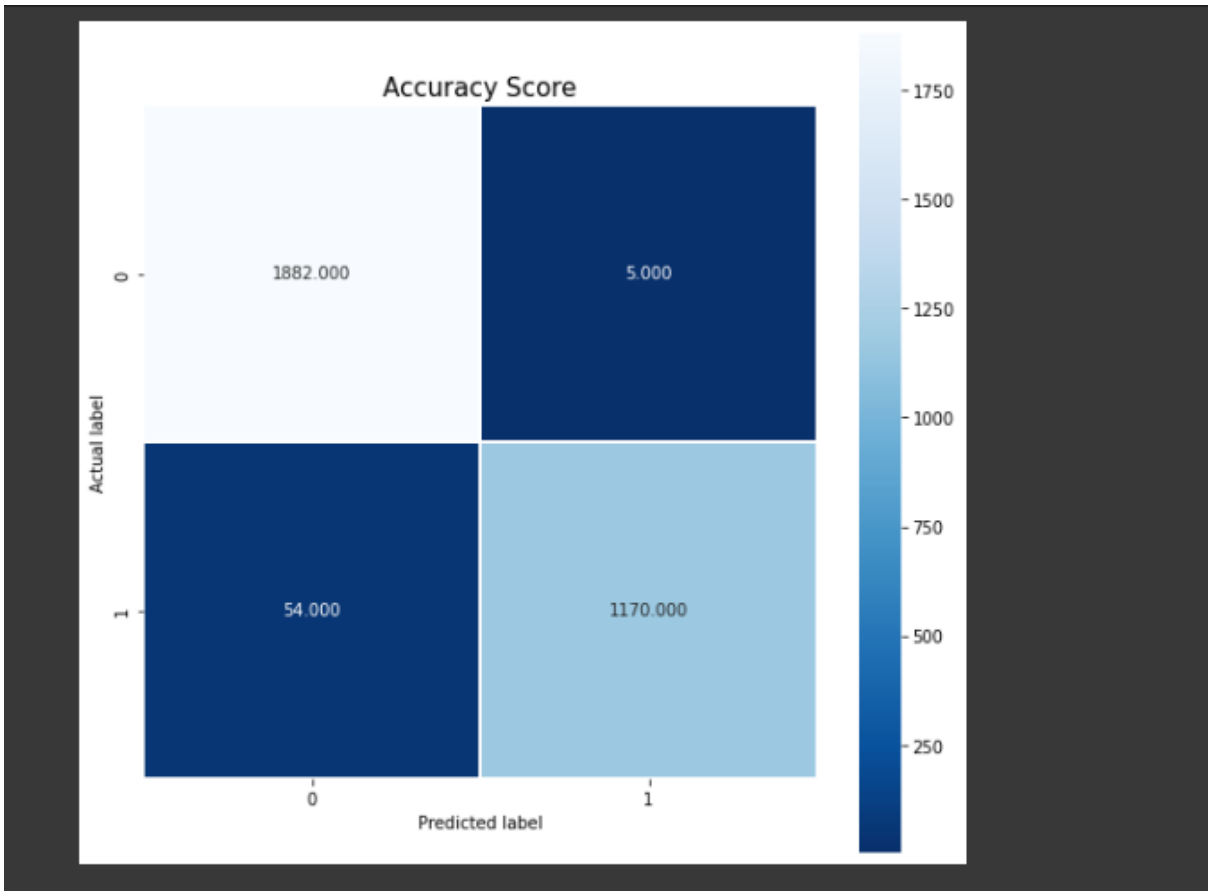
```
[59] import sklearn.metrics as metrics

     print("Accuracy:",metrics.accuracy_score(Y_test, Y_pred))

     Accuracy: 0.9810350369656059
```

- *Compute the confusion matrix*

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test,Y_pred)
print(confusion_matrix(Y_test,Y_pred))

[[1882    5]
 [  54 1170]]
```

- *Plot the heatmap of Predicted Label vs Actual Label (Confusion Matrix)*

```
[66] import seaborn as sns
     plt.figure(figsize=(9,9))

     sns.heatmap(cm,annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');

     plt.ylabel('Actual label');

     plt.xlabel('Predicted label');

     all_sample_title = 'Accuracy Score'

     plt.title(all_sample_title, size = 15);
```



- **<u>SVM</u> (Support Vector Machine)**

  SVM is one of the most popular supervised learning algorithms, which can be used for both classification and regression problems.

  The idea of SVM is to create a line or decision boundary that can segregate the n-dimensional space into classes so that the new data points can be easily put in the correct category.

  There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but the best decision boundary is to be selected, which is calle the hyperplane of SVM.

  There are 2 types of SVM:

  - **Linear SVM**
    It is used for linearly separable data, that is for datasets that can be classified into 2 classes with a single straight line. The data here is termed as linearly separable data.
  - **Non-linear SVM**
    It is used for non-linearly separable data, that is if a straight line cannot separate the data into 2 classes. The data here is termed as non-linear data.

    SVM is a very good algorithm for doing classification. It offers good accuracy and perform faster predictions compared to Naïve Bayes algorithm.

Implementation and Coding Phase of SVM Algorithm

- *Import the Vectorization packages CountVectorizer and Tf-idf Vectorizer*

```
[ ]  from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
     cv = CountVectorizer()
     tfidf = TfidfVectorizer(max_features=3000)
```

- *Assign values to X and Y predictor variables*

```
X = tfidf.fit_transform(data['Cleaned Text'].values.astype('U')).toarray()
Y = data['spam'].values
```

- *Implement SVM using inbuilt scikit library*

```
[ ]  from sklearn.model_selection import train_test_split
     from sklearn import svm

     X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size = 0.2, random_state = 2 )

     clf = svm.SVC(kernel = 'linear')

     clf.fit(X_train, Y_train)

     Y_pred = clf.predict(X_test)
```

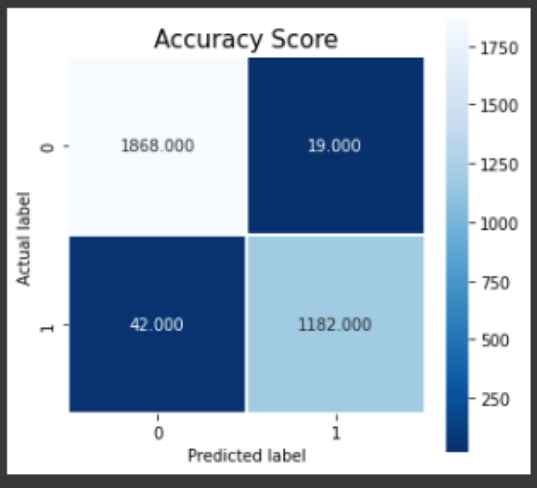
- *Calculate the accuracy, precision and recall of model using sklearn*

```
[ ]  from sklearn import metrics

     print("Accuracy:", metrics.accuracy_score(Y_test, Y_pred))

     print("Precision:",metrics.precision_score(Y_test, Y_pred))

     print("Recall:",metrics.recall_score(Y_test, Y_pred))

     Accuracy: 0.9803921568627451
     Precision: 0.984179850124896
     Recall: 0.9656862745098039
```

The model has an accuracy of 98.04% and a precision of 98.42%.

- *Calculate the confusion matrix and plot the accuracy score*

```
from sklearn.metrics import confusion_matrix

import seaborn as sns

cm = confusion_matrix(Y_test, Y_pred)

plt.figure(figsize=(5, 5))

sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');

plt.ylabel('Actual label');

plt.xlabel('Predicted label');

all_sample_title = 'Accuracy Score'

plt.title(all_sample_title, size = 15);
```



- **Random Forest Classification**

Random Forest is a supervised Machine Learning algorithm, which is used for both classification and regression problems.

Random Forest classifier contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.

Higher the number of trees in the forest, higher is the accuracy and also prevents the problem of overfitting.

Random Forest uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

Random Forests is great with high dimensional data since we are working with subsets of data. It is faster to train than decision tree because we are working only on a subset of features in this model, so we can easily work with hundreds of features.

Implementation and Coding Phase of Random Forest:

- *Import the necessary vectorizers*

```
[ ]  from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
     cv = CountVectorizer()
     tfidf = TfidfVectorizer(max_features=3000)
```

- *Give values to training variables X and Y*

```
[ ]  X = tfidf.fit_transform(data['Cleaned Text'].values.astype('U')).toarray()

[ ]  X.shape

     (15552, 3000)

[ ]  Y = data['spam'].values

     Y.shape

     (15552,)
```

- *Split the data into training and testing partition (20% test, 80% training)*

```
[ ]  from sklearn.model_selection import train_test_split

     X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size = 0.2, random_state = 2 )
```

- *Implement Random Forest using scikit-learn*

```
[ ] from sklearn.model_selection import train_test_split

    X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size = 0.2, random_state = 2 )
```

```
[ ] from sklearn.ensemble import RandomForestClassifier
    classifier = RandomForestClassifier(n_estimators=20, random_state=5)
    classifier
```

```
    RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                           criterion='gini', max_depth=None, max_features='auto',
                           max_leaf_nodes=None, max_samples=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=20,
                           n_jobs=None, oob_score=False, random_state=5, verbose=0,
                           warm_start=False)
```

```
[ ] #Import Random Forest Model
    from sklearn.ensemble import RandomForestClassifier

    #Create a Gaussian Classifier
    clf=RandomForestClassifier(n_estimators=100)

    #Train the model using the training sets y_pred=clf.predict(X_test)
    clf.fit(X_train,Y_train)

    Y_pred=clf.predict(X_test)
```

- *Compute the confusion matrix*

```
[ ] from sklearn.metrics import confusion_matrix

    print(confusion_matrix(Y_test,Y_pred))
```

```
    [[1876   11]
     [  56 1168]]
```

- *Compute the F1 Score of the model*

```
[ ] from sklearn.metrics import f1_score

    print(f1_score(Y_test,Y_pred))

    0.9721181856013317
```

- *Calculate the accuracy of the model*

```
from sklearn.metrics import accuracy_score

print(accuracy_score(Y_test,Y_pred))

0.9784635165541626
```

The model gives an accuracy of 97.84% using Random Forest Classification.

- **Predictions on Validation Dataset**

  For KNN algorithm, the accuracy was only 86.12%. Therefore, in order to increase the accuracy of the algorithm, we are using K Fold Cross Validation.

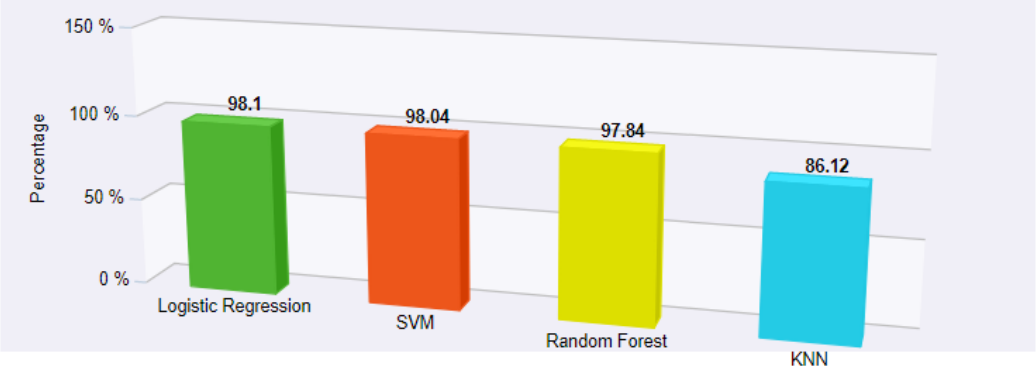
  K fold cross validation is an alternative to data partitioning, when number of records is small/

  For example if k = 5, the data is randomly partitioned into 20 equal parts, where each fold has 5% of the data.

  - Algorithm:
    - Randomly split dataset of n instances into K folds
    - For each k fold, built models on k-1 fold of dataset. Test the model to check for effectiveness for kth fold
    - Record the error on each prediction
    - Repeat the process until each k fold has become the test set
    - The average of k recorded errors is the cross validation error and will form the performance metric

```python
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

accuracy1 = []

kf = KFold(n_splits=5, random_state=None)

for train_index, test_index in kf.split(X):

      #print("Train:", train_index, "\nValidation:",test_index)

      X_train, X_test = X[train_index], X[test_index]
      Y_train, Y_test = Y[train_index], Y[test_index]

      # Standardization
      scaler = StandardScaler()
      X_train = scaler.fit_transform(X_train)
      X_test = scaler.transform(X_test)

      # Training the Model
      model = KNeighborsClassifier( n_neighbors = 5 )
      model.fit( X_train, Y_train.ravel() )

      # Predicting Test Data Set
      Y_pred = model.predict( X_test )

    # Confusion Matrix
from sklearn.metrics import confusion_matrix
print("\n\nConfusion Matrix\n\n", confusion_matrix(Y_test,Y_pred), end = "\n")

    # F1 Score
from sklearn.metrics import f1_score
print("\nF1 Score : ", f1_score(Y_test,Y_pred), end = "\n")

    # Accuracy Score
from sklearn.metrics import accuracy_score

accuracy1.append(accuracy_score(Y_test, Y_pred))
print("\nAccuracy Score : ", accuracy_score(Y_test,Y_pred))
```

❖ Accuracy Comparison of various ML Algorithms



❖ Future Scope

Spam Email Detection is essential in a Business Enterprise as it can ensure justice for sellers and retain the trust of the buyers in Online Shopping. These algorithms which have been used still have not eliminated the need for manual checking of emails. There is still scope for complete automation of spam detection systems which have higher efficiency.

The spammers also are getting smarter and spam reviews are getting untraceable. Therefore, it is necessary to identify different algorithms to identify the spam emails.

This algorithm can further be extended to cellphones, which received both SMS texts and other messaging applications.

❖ Conclusion

In this study, we have reviewed various Machine Learning approaches and their applications for spam email filtering. The paper surveyed a number of public accessible datasets and performance metrics. The challenges of these Machine Learning algorithms in expeditiously handling the menace of spam was found out and comparative studies of the machine learning technics accessible in literature was done.

Given a set of words, we used feature selection to obtain words which allowed us to distinguish between spam and ham emails. Therefore, we compared the accuracy of various Machine Learning algorithms for predicting the class attribute.

It is observed that Logistic Regression gives the highest classification accuracy of 98.10% in our trials, followed by SVM and Random Forest Algorithms.

KNN gives the least accuracy of 86.12%, when K is given a value of 5.

References

https://wiki.eecs.yorku.ca/course_archive/2013
14/F/4403/_media/project_report.pdf
https://www.malayajournal.org/articles/MJM0S0101.pdf
https://turkjphysiotherrehabil.org/pub/pdf/321/32-1-242.pdf
https://www.javatpoint.com/
https://www.geeksforgeeks.org/
https://towardsdatascience.com/
https://www.analyticsvidhya.com/blog/2021/06/automated-spam-e-
mail-detection-modelusing-common-nlp-tasks/
https://bdtechtalks.com/2020/11/30/machine-learning-spam-detection/
https://scikit-learn.org/stable/
https://www.nltk.org/