MACHINE LEARNING PROJECT 1

People receive many emails everyday and many of them could be spam, so how could we detect the spam emails and reduce our time to check them one by one? This project is about using Naive Bayes algorithm,Logistic regression,Decision tree,Support Vector machine algorithm to build a machine learning classification model to detect the spam email. All the packages worked with was added in the beginning of this project.

In [49]:

```
import numpy as np import
pandas as pd import
matplotlib.pyplot as plt import
nltk import matplotlib import
seaborn as sns import re import
collections from collections
import Counter import csv
from nltk.corpus import stopwords from
sklearn.feature_extraction.text import TfidfVectorizer from
sklearn.naive_bayes import MultinomialNB from
sklearn.metrics import confusion_matrix from
sklearn.metrics import accuracy_score from sklearn.metrics
import recall_score from sklearn.metrics import
precision_score from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report from
sklearn.linear_model import LogisticRegression from
sklearn.svm import SVC from nltk.classify import
NaiveBayesClassifier from nltk.tokenize import
word_tokenize from nltk.tokenize import RegexpTokenizer
import os import string
from string import punctuation pd.options.mode.chained_assignment = None
from nltk.stem.snowball import SnowballStemmer from spellchecker import
SpellChecker from sklearn.model_selection import train_test_split from
collections import Counter from nltk.stem import
WordNetLemmatizer,PorterStemmer from sklearn.feature_extraction.text
import CountVectorizer from sklearn.linear_model import SGDClassifier from
sklearn import svm from sklearn.datasets import load_files from
sklearn.dummy import DummyClassifier from sklearn.model_selection import
GridSearchCV from sklearn.pipeline import Pipeline from
sklearn.feature_extraction.text import TfidfTransformer from
sklearn.metrics import precision_recall_curve,auc, accuracy_score,f1_
score from time import time
from sklearn.model selection import learning curve,ShuffleSplit,GridSearch
```

```
CV
from sklearn import tree
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
```

There are 3672 ham and 1500 spam emails in the dataset,the number of spam messages are smaller than that of ham in the dataset we are working on.The data was loaded and read.Classifying not spam messages(ham) as 0 and spam messages as 1

In [55]:

```
rootdir="C:/Users/soar/Desktop/enron1"#Setting my working directory to whe
re my data is located

for directories, subdirs, files in os.walk(rootdir):
    print(directories, subdirs, len(files))
```

```
C:/Users/soar/Desktop/enron1 ['ham', 'spam'] 0
C:/Users/soar/Desktop/enron1\ham [] 3672
C:/Users/soar/Desktop/enron1\spam [] 1500
```

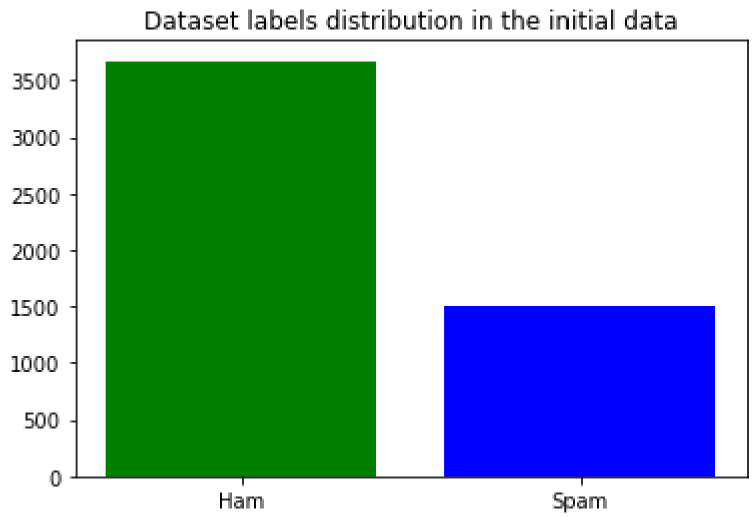
In [56]:

```
ham_list = []
spam_list = []
#reading ham email and creating dataframe(ham=0)
for directories, subdirs, files in os.walk(rootdir):
    if (os.path.split(directories)[1]  == 'ham'):
        for filename in files:
            with open(os.path.join(directories, filename), encoding="latin
-1") as f:
                data = f.read()
                ham_list.append(data)
ham=pd.DataFrame(ham_list,columns=["message"])
ham["class"] = 0

#reading spam email and creating dataframe(spam=1)
if (os.path.split(directories)[1]  == 'spam'):
        for filename in files:
            with open(os.path.join(directories, filename), encoding="latin
-1") as f:
                data = f.read()
                spam_list.append(data)
spam=pd.DataFrame(spam_list,columns=["message"])#Creating a Pandas Datafra
me of ham and spam
spam["class"] = 1
emails_df=pd.concat([spam,ham])
```

In [62]:

```
target_cnt = Counter(emails_df['class'])#Visualizing the Label distributio
n in our initial data

plt.figure(figsize=(6,4))
plt.bar(target_cnt.keys(), target_cnt.values(),tick_label =('Spam', 'Ham'
),color=['b','g'] )
plt.title("Dataset labels distribution in the initial data")
plt.show;
```

p



From the plot we can see that non-spam(ham) words are longer than the spam words basically because they contain relevant informations,for task,jobs,meetings and also from family. The ham words or non-spam also contains more replies this is why they are longer than spam words.

In [68]:

```
# Reset pandas df index.
emails_df = emails_df.sample(frac=1).reset_index(drop=True) #reshuffling th
e text to enable even distribution

print(emails_df.index)
emails_df
```

```
RangeIndex(start=0, stop=5172, step=1)
```

Out[68]:

	message	class
0	subject site sweet teen sex check odownload ho...	1
1	subject hpl nom march see attached file hplno ...	0
2	subject archived great shot california livingg...	1
3	subject operating productionset information re...	0
4	subject swjlmiqqt fountain youthajuinbol balza...	1
...	...	...
5167	subject cornhuskertenaska operating without ga...	0
5168	subject hillarious take minute call listen gre...	0
5169	subject noms forwarded ami chokshi corp enron ...	0

	j	p
5170	subject want something extra bed try revolutio...	1
5171	subject hpl meter brown common pointdaren peri...	0

5172 rows × 2 columns

In any machine learning task, cleaning or preprocessing the data is the most important part.There are different types of text preprocessing steps which we can do on text data. We need to carefully choose the preprocessing steps after examining the data and apply a particular step based on our case. The preprocessing / cleaning steps we undertook are: Lower casing,Removal of Punctuations,Removal of Stopwords,tokenizing,Stemming,Lemmatization etc. Lower casing :The idea is to convert the input text into same casing format so that 'text', 'Text' and 'TEXT' are treated the same way.This is more helpful for text featurization techniques like frequency, tfidf as it helps to combine the same words together thereby reducing the duplication and get correct counts / tfidf values. Removal of Punctuation: This is a text standardization process that will help to treat 'hurray' and 'hurray!' in the same way. The string.punctuation in python contains punctuation symbols this was what was used. Stemming:This is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form. For example, if there are two words in the corpus run and running, then stemming will stem the suffix to make them run. The stemming algorithm used is porter stemmer which is widely used. Tokenization: Tokenization is the method where the sentences within an email are broken into individual words (tokens). These tokens are saved into an array and used towards the testing data to identify the occurrence of every word in an email. This will help the algorithms in predicting whether the email should be considered as spam or ham.

In [61]:

```
stemmer=PorterStemmer() punctuation = string.punctuation def
clean_email(email):    email=str(email)    email = re.sub(r'http\S+$@',
'', email)    email = re.sub("\d+", "", email)    email =
email.replace('\n', '')    email = re.sub(r'http\S+','',email)    email =
re.sub('[0-9]+','',email)#Removal of numbers    email =
email.translate(str.maketrans("", "", punctuation))#Removal of
punctuations
    email = email.lower() #lower casing    email1 =RegexpTokenizer(r'\w+')
 tokens = email1.tokenize(email) #tokenizing    filtered_words=[w for w in
 tokens if len(w)>2 if not w in stopwords.wo
rds('english')]#Removal of stopwords
stem_words=[stemmer.stem(w)for w in filtered_words]#Stemming
return " ".join(filtered_words) emails_df['message'] =
emails_df['message'].apply(clean_email)
```

Stopwords are commonly occuring words in a language like 'the', 'a' etc.. They are removed from the text as they don't provide valuable information for analysis. The stopword lists for english language from the nltk package was combined with new stopwords observed from the data and removed.

In [7]:

```
STOPWORDS = set(stopwords.words('english'))
newstopwords={'subject','re','email','u','p','td','e','www','http','c'}
```

```
STOPWORDS =STOPWORDS.union(newstopwords)
def remove_stopwords(text):
    """custom function to remove the stopwords"""
    return " ".join([word for word in str(text).split() if word not in STO
PWORDS])

emails_df["message"] = emails_df["message"].apply(lambda text: remove_stop
words(text))
```

Lemmatization is similar to stemming in reducing inflected words to their word stem but differs in the way that it makes sure the root word (also called as lemma) belongs to the language.

In [63]:

```
lemmatizer = WordNetLemmatizer()
def lemmatize_words(text):
    return " ".join([lemmatizer.lemmatize(word) for word in text.split()])

emails_df["message"] = emails_df["message"].apply(lambda text: lemmatize_w
ords(text))
```

First is to check if there is any missing data that needs to be replaced or dropped. Since there is none, we carry on with our model building.

In [65]:

```
emails_df['message'].isnull().sum#checking for missing data
```

Out[65]:

```
<bound method Series.sum of 0       False
1       False
2       False
3       False
4       False
        ...
3667    False
3668    False
3669    False
3670    False
3671    False
Name: message, Length: 5172, dtype: bool>
```

In [10]:

```
anydup=emails_df['message'].duplicated().any()
np.where(pd.isnull(emails_df))#checking for empty emails
```

Out[10]:

```
(array([], dtype=int64), array([], dtype=int64))
```

In [11]:

```
np.where(emails_df.applymap(lambda x: x == ''))#checking for empty emails
```

Out[11]:

```
(array([ 520,  534,  563, 1126, 1146, 2035, 2204, 2477, 2502,
```

```
2902, 3442,
        3702, 3770, 3772, 3801, 3971, 3986, 4145, 4732], dtyp
e=int64),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0],
 dtype=int64))
```

In [12]:

```
# Define the independent variables as Xs.
X=emails_df['message']
# Define the target (dependent) variable as Y.
Y = emails_df['class']
# Create a train/test split using 30% test size.
x_train, x_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.3,rand
om_state=0)
# Check the split printing the shape of each set.
print(emails_df.message.shape) print(x_train.shape,
Y_train.shape) print(x_test.shape, Y_test.shape)
```

```
(5172,)
(3620,) (3620,)
(1552,) (1552,)
```

To analyze the text data, we have to turn the words into numerical numbers. We have multiple choices to accomplish this step, Binary Term Frequency, Bag of Words Frequency, L1 normalized Term Frequency, L2 Normalized TFIDF and Word2Vec. This is called feature extraction. Next we split the data into train and test data and determine the X train, y train, X test and y test. Then we use the CountVectorizer and our Machine learning model to fit and transform the train set and transform test set. The testing feature (independent) data set will be stored in X_test and the testing target (dependent) data set will be stored in y_test . The training feature (independent) data set will be stored in X_train and the training target (dependent) data set will be stored in y_train .

In [13]:

```
X = emails_df['message'].values# Define the independent variables as X.
# Define the target (dependent) variable as Y.
Y = emails_df['class'].values
# Vectorize words - Turn the text numerical feature vectors, #
using the strategy of tokenization, counting and normalization.
vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.5,
                                       stop_words='english')
X = vectorizer.fit_transform(X)
# Create a train/test split using 30% test size.
             X_train, X_test, y_train, y_test = train_test_split(X,
                                                                  Y,
                                                         test_size=0.3,
                                                          shuffle=True,
                                                        random_state=0)
# Check the split printing the shape of each set.
print(X_train.shape, y_train.shape) print(X_test.shape,
y_test.shape)
```

```
(3620, 62086) (3620,)
(1552, 62086) (1552,)
```

In [14]:

```
train_set=pd.concat([x_train,Y_train],axis=1)#Joining the train features t
o its label in a pandas Dataframe
```

In [15]:

```
test_set=pd.concat([x_test,Y_test],axis=1)#Joining the train features to i
ts label in a pandas Dataframe
```

In [16]:

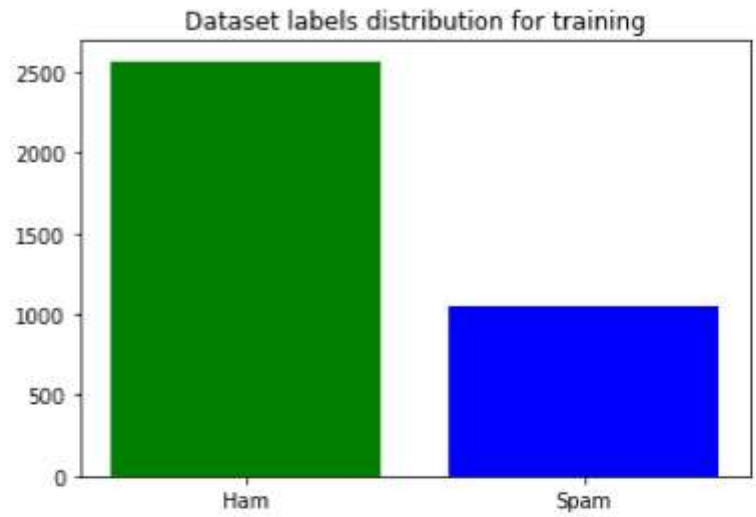
```
train_set.to_csv("train.csv",sep = "\t",encoding="utf-8")#Saving the train
set in a csv file
```

In [17]:

```
test_set.to_csv("test.csv",sep="\t",encoding="utf-8")#Saving the test file
in csv file
```

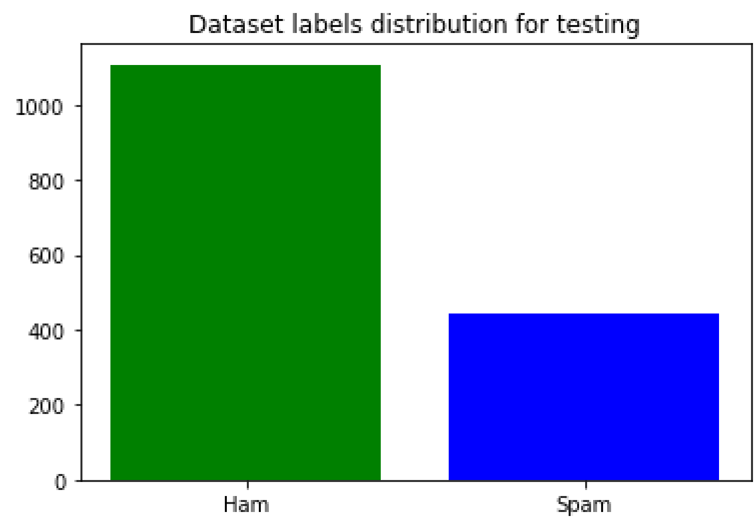
In [59]:

```
#visualising the spam and ham in the training set
target_cnt = Counter(y_train)

plt.figure(figsize=(6,4))
plt.bar(target_cnt.keys(), target_cnt.values(),tick_label =('Spam', 'Ham'
),color=['b','g'] )
plt.title("Dataset labels distribution for training")
plt.show;
```



In [60]:

```
#visualising the spam and ham in the test set
target_cnt = Counter(y_test)

plt.figure(figsize=(6,4))
plt.bar(target_cnt.keys(), target_cnt.values(),tick_label =('Ham', 'Spam'
),color=['g','b'] )
plt.title("Dataset labels distribution for testing")
plt.show;
```



From the plots both in the training and test set we can see that non-spam(ham) words are longer than the spam words basically because they contain relevant informations,for task,jobs,meetings and also from family. The ham words or non-spam also contains more replies this is why they are longer than spam words

In [20]:

```
#Count of ham and spam in train data,where 0 represents ham and 1 represen
ts spam
Y_train.value_counts()
```

Out[20]:

```
0    2565
1    1055
Name: class, dtype: int64
```

In [21]:

```
Y_test.value_counts()#Count of ham and spam in the test data
```

Out[21]:

```
0    1107
1     445
Name: class, dtype: int64
```

In [22]:

```
spamwords=' '.join(train_set[train_set['class']==1]['message'])
topspam=collections.Counter(spamwords.split()).most_common(20)
```

In [23]:

```
top20spam=pd.DataFrame(topspam,columns=['Word','Count'])
```

In [25]:

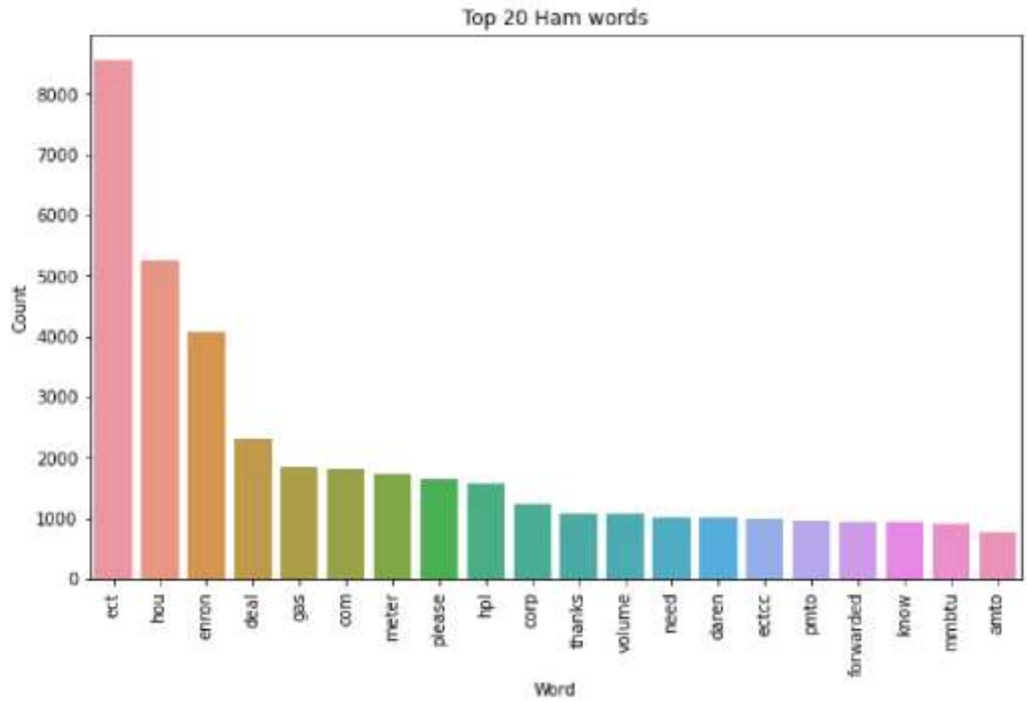
```
hamwords=' '.join(train_set[train_set['class']==0]['message'])
topham=collections.Counter(hamwords.split()).most_common(20)
```

In [26]:

```
top20ham=pd.DataFrame(topham,columns=['Word','Count'])
```

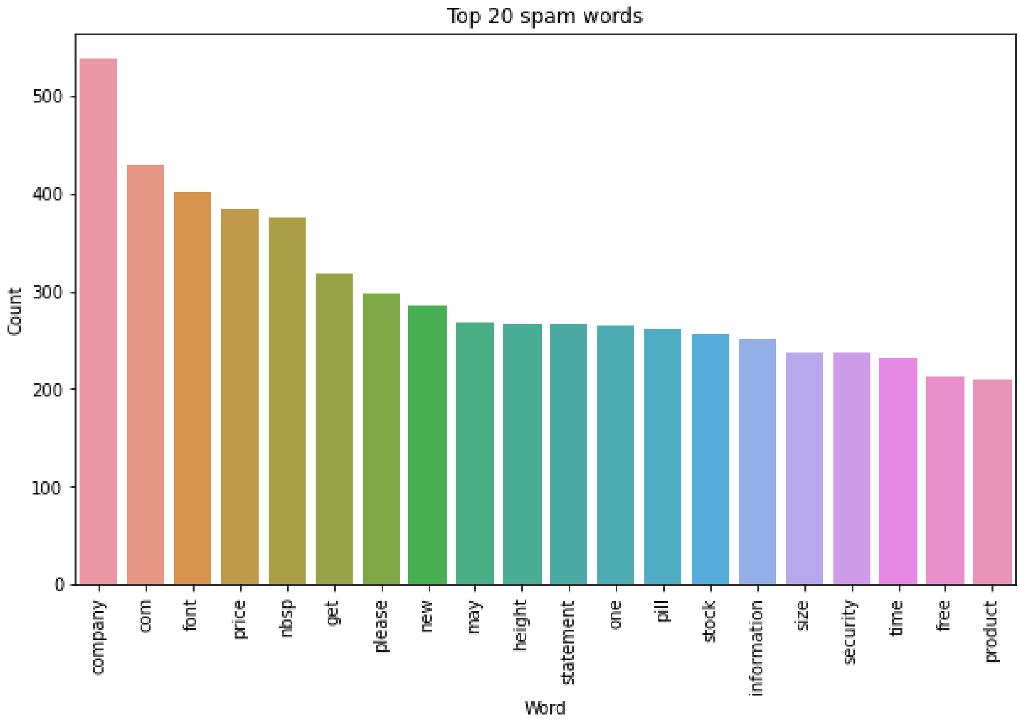
In [28]:

```
fig, ax = plt.subplots(figsize=(10, 6))# A bar plot showing the most frequ
ent Ham words
sns.barplot(x='Word', y='Count',
            data=top20ham, ax=ax)
plt.title("Top 20 Ham words")
plt.xticks(rotation='vertical');
```



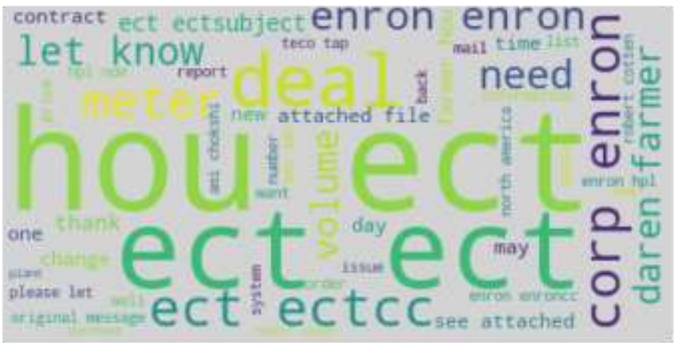
In [29]:

```
fig, ax = plt.subplots(figsize=(10, 6))#A bar plot showing the most freque
nt spam words
sns.barplot(x='Word', y='Count',
            data=top20spam, ax=ax)
plt.title("Top 20 spam words")
plt.xticks(rotation='vertical');
```



In [30]:

```
# Create and generate a word cloud image:
wordcloud = WordCloud(max_words=50, background_color="lightgrey").generate
(hamwords)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



For non-spam or ham we can see dominant words like enron,meter,gas,daren etc,

In [31]:

```
# Create and generate a word cloud image:
```

```
wordcloud = WordCloud(max_words=50, background_color="lightgrey",height=51
2,width=640).generate(spamwords)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

in spam words as expected we see dominant words like company,business,price,software-product,order,offer etc.

In [32]:

```
#Use  tf*idf features extraction
vectorizer = TfidfVectorizer(ngram_range=(1, 2), max_features = 2000,subli
near_tf=True,use_idf=True)
X_train_tfidf = vectorizer.fit_transform(x_train)
X_train_tfidf.shape
```

Out[32]:

```
(3620, 2000)
```

In [33]:

```
vectorizer = TfidfVectorizer(ngram_range=(1, 2), max_features = 2000,subli
near_tf=True,use_idf=True)
X_test_tfidf = vectorizer.fit_transform(x_test)
X_test_tfidf.shape
```

Out[33]:

```
(1552, 2000)
```

MACHINE LEARNING MODELS The sections below explains each of the Machine Learning models that will be implemented to achieve the aim of this project. Linear Classifiers This classifier is useful as a simple baseline to compare with other (real) classifiers. It shows how many mails were classified as ham and spam.

In [34]:

```
#Using Linear model
#using TFIDF
detect_model = DummyClassifier(strategy='most_frequent').fit(X_train , y_t
rain)
pred_train_LM = detect_model.predict(X_train )
pred_test_LM =  detect_model.predict(X_test)
acc_TELM = accuracy_score(y_test, pred_test_LM)
acc_TRLM = accuracy_score(y_train, pred_train_LM)
print("Test accuracy: %.2f%%" % (acc_TELM*100))#accuracy for test data/val
idation
print("Train accuracy: %.2f%%" % (acc_TRLM *100))#accuracy for train data
print()
print("Test data confusion matrix")
y_true = pd.Series(y_test, name='True')
y_pred = pd.Series(pred_test_LM , name='Predicted')
pd.crosstab(y_true, y_pred)
```

```
Test accuracy: 71.33%

Train accuracy: 70.86%

Test data confusion matrix
```

Out[34]:

Predicted	0
True	
0	1107
1	445

Let us test both extraction method on Multinomial naive bayes classifier First test for classification

CONFUSION MATRIX The detection of spam emails can be evaluated by different performance measures. Confusion Matrix is being used to visualise the detection of the emails for models. Confusion matrix can be defined as below: True Negative – Ham email predicted as ham False Negative – Ham email predicted as spam True Positive – Spam email predicted as spam False Positive – Spam email predicted as ham Where Ham represents 0 and spam represents 1

In [35]:

```
#using TFIDF
train_scores = []
test_scores = []
times = []

t = time()
detect_model = MultinomialNB()
detect_model.fit(X_train , y_train)
pred_train_MNB = detect_model.predict(X_train )
score = f1_score(y_train, pred_train_MNB)
train_scores.append(score )
print("Train f1 score: %.2f%%" % (score*100))#accuracy for test data/valid
ati
pred_test_MNB =  detect_model.predict(X_test)
```

```
times.append(time()-t)

score = f1_score(y_test, pred_test_MNB)
test_scores.append(score)

print("Test f1 score: %.2f%%" % (score *100))#accuracy for train data
print()
print("Test data confusion matrix")
y_true = pd.Series(y_test, name='True')
y_pred = pd.Series(pred_test_MNB , name='Predicted')
pd.crosstab(y_true, y_pred)
```

```
Train f1 score: 92.95%
Test f1 score: 76.86%

Test data confusion matrix
```

Out[35]:

Predicted	0	1
True		
0	1105	2
1	166	279

Logistic Regression with SGD optimizer Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes.

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as Logistic Regression.

EVALUATION

In evaluating the performance of the model,the following metrics was selected and investigated,this is to examine all error by looking at parameters that combine several approaches. Confusion or error matrix, Accuracy,Recalll or Sensitivity,Precision,F1-score,Area under Receiver Operating Curve was all used for evaluation and result was interpreted.

In [36]:

```
t = time()
detect_model = SGDClassifier(loss="log", penalty="l2", tol=0.0001)
detect_model.fit(X_train , y_train)
pred_train_LR = detect_model.predict(X_train )
score = f1_score(y_train, pred_train_LR)
train_scores.append(score )
print("Train f1 score: %.2f%%" % (score*100))#accuracy for test data/valid
ation
pred_test_LR =  detect_model.predict(X_test)
times.append(time()-t)

score = f1_score(y_test, pred_test_LR)
test_scores.append(score)

print("Test f1 score: %.2f%%" % (score *100))#accuracy for train data
```

```
print()
print("Test data confusion matrix")
y_true = pd.Series(y_test, name='True')
y_pred = pd.Series(pred_test_LR , name='Predicted')
pd.crosstab(y_true, y_pred)
```

```
Train f1 score: 99.95%
Test f1 score: 95.82%

Test data confusion matrix
```

Out[36]:

Predicted	0	1
True		
0	1078	29
1	9	436

This algorithm plots each node from a dataset within a dimensional plane and through classification technique the cluster of data is separated by a hyperplane into their respective groups . The hyperplane can be described as equation-5:H = VX + c (5) where c is a constant and V is the vector. Disadvantage of working with SVC algorithm is that it cannot handle a large dataset, whereas SGD provides efficiency and other tuning opportunities.

In [37]:

```
t = time()
detect_model = svm.SVC(kernel='linear')
detect_model.fit(X_train , y_train)
pred_train_SVM = detect_model.predict(X_train )
score = f1_score(y_train, pred_train_SVM)
train_scores.append(score )
print("Train f1 score: %.2f%%" % (score*100))#accuracy for test data/valid
ation
pred_test_SVM =  detect_model.predict(X_test)
times.append(time()-t)

score = f1_score(y_test, pred_test_SVM)
test_scores.append(score)

print("Test f1 score: %.2f%%" % (score *100))#accuracy for train data
print()
print("Test data confusion matrix")
y_true = pd.Series(y_test, name='True')
y_pred = pd.Series(pred_test_SVM , name='Predicted')
pd.crosstab(y_true, y_pred)
```

```
Train f1 score: 99.95%
Test f1 score: 96.58%

Test data confusion matrix
```

Out[37]:

Predicted	0	1
True		

0	1083	24
1	7	438

The Decision Tree model is based on the predictive method.The model creates a category which is further distributed into sub-categories and so on. The algorithm runs until the user has terminated or the program has reached its end decision.The model predicts the value of the data by learning from the provided training data.

In [38]:

```
#Using Decision tree
t = time()
detect_model = tree.DecisionTreeClassifier()
detect_model.fit(X_train , y_train)
pred_train_DT = detect_model.predict(X_train )
score = f1_score(y_train, pred_train_DT)
train_scores.append(score )
print("Train f1 score: %.2f%%" % (score*100))#accuracy for test data/valid
ation
pred_test_DT =  detect_model.predict(X_test)
times.append(time()-t)

score = f1_score(y_test, pred_test_DT)
test_scores.append(score)

print("Test f1 score: %.2f%%" % (score *100))#accuracy for train data
print()
print("Test data confusion matrix")
y_true = pd.Series(y_test, name='True')
y_pred = pd.Series(pred_test_DT , name='Predicted')
pd.crosstab(y_true, y_pred)
```

```
Train f1 score: 100.00%
Test f1 score: 91.25%

Test data confusion matrix
```

Out[38]:

Predicted	0	1
True		
0	1061	46
1	33	412

In [39]:

```
estimator = ['Naive Bayes','Logistic Regression with SGD','SVM','Decision
 Tree']
d = {'Estimator': estimator,
     'Train F1-score':train_scores,
     'Test F1-score': test_scores,
     'Time to fit and predict (secs)': times
    }
```

```
df = pd.DataFrame(data=d).set_index('Estimator')
```

Out[39]:df

	Train F1-score	Test F1-score	Time to fit and predict (secs)
Estimator			
Naive Bayes	0.929477	0.768595	0.021992
Logistic Regression with SGD	0.999526	0.958242	0.054970
SVM	0.999526	0.965821	6.059038
Decision Tree	1.000000	0.912514	1.491059

Interpreting the result above, it can be seen that: All Classifiers achieved high F1 scores in the unseen test set except Naive Bayes. ranging from (76.67%-97.91%). The Naive Bayes is the lowest classifier. Logistic Regression using SGD as optimizer is also fast and performs better than the Decision tree classifier, which is much slower. SVM has the same score with the logistic regression classification (97.53%). SVM algorithms is computationally expensive to run than Logistic Regression and Naive Bayes as it takes more time. ACCURACY This project is aimed at finding the highest accuracy for detecting the emails correctly as ham and spam. Accuracy analyses the correct number of emails classified as ‘Spam’ and ‘Ham’.This can be measured by equation below: The RECALL measurement provides the calculation of how many emails were correctly predicted as spam from the total number of spam emails that were provided. This is defined by equation, where ‘TP + FN’ are the total number of spam emails within thetext PRECISION The precision measurement is to calculate the correctly identified values, meaning how many correctly identified spam emails have been classified from the given set of positive emails. This means to calculate the total number of emails which were correctly predicted as positive from amongst the total number of emails predicted positive. This is defined by equation: Precision=TP/TP+FP ROC-AUC Curve In Machine learning,performance measurement is an essential task.When we need to check or visualize the performance of the multi-classification problem,we use the ROC(Receiver Operating Characteristics)-AUC(Area Under Curve) Curve.It is one of the most important evaluation metrics for checking any classification model's performance.

In [40]:

```
#Accuracy score
test_accscores = []
test_roauscores = [] times
= []

t  =  time()  detect_model  =  MultinomialNB()  detect_model.fit(X_train  ,
y_train)  pred_test_MNB  =   detect_model.predict(X_test)  acc_TEMNB  =
accuracy_score(y_test,pred_test_MNB)     test_accscores.append(acc_TEMNB)
roau_TEMNB=          roc_auc_score(y_test,         pred_test_MNB        )
test_roauscores.append(roau_TEMNB)   times.append(time()-t)   print("Test
accuracy: %.2f%%" % (acc_TEMNB*100))#accuracy for test data/va lidation
print("ROC-AUC SCORE: %.2f%%" % (roau_TEMNB*100))#accuracy for test data/v
alidation
print(classification report(y test, pred test MNB))
```

```
Test accuracy: 89.18%
ROC-AUC SCORE: 81.26%
              precision    recall  f1-score   support

           0       0.87      1.00      0.93      1107
           1       0.99      0.63      0.77       445

    accuracy                           0.89      1552
   macro avg       0.93      0.81      0.85      1552
weighted avg       0.90      0.89      0.88      1552
```

In [41]:

```
#For logistic regression
t = time()
detect_model = SGDClassifier(loss="log", penalty="l2", tol=0.0001)
detect_model.fit(X_train , y_train)
pred_test_LR =  detect_model.predict(X_test)
acc_TELR = accuracy_score(y_test,pred_test_LR)
test_accscores.append(acc_TELR)
roau_TELR= roc_auc_score(y_test, pred_test_LR)
test_roauscores.append(roau_TELR)
times.append(time()-t)
print("Test accuracy: %.2f%%" % (acc_TELR*100))#accuracy for test data/val
idation
print("ROC-AUC SCORE: %.2f%%" % (roau_TELR*100))#accuracy for test data/va
lidation
print(classification_report(y_test, pred_test_LR))
```

```
Test accuracy: 97.55%
ROC-AUC SCORE: 97.68%
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      1107
           1       0.94      0.98      0.96       445

    accuracy                           0.98      1552
   macro avg       0.96      0.98      0.97      1552
weighted avg       0.98      0.98      0.98      1552
```

In [42]:

```
#For Support Vector Machine
t = time()
detect_model = svm.SVC(kernel='linear')
detect_model.fit(X_train , y_train)
pred_test_SVM =  detect_model.predict(X_test)
acc_TESVM = accuracy_score(y_test,pred_test_SVM)
test_accscores.append(acc_TESVM)
roau_TESVM= roc_auc_score(y_test, pred_test_SVM)
test_roauscores.append(roau_TESVM)
times.append(time()-t)
print("Test accuracy: %.2f%%" % (acc_TESVM*100))#accuracy for test data/va
lidation
print("ROC-AUC SCORE: %.2f%%" % (roau_TESVM*100))#accuracy for test data/v
alidation
print(classification_report(y_test, pred_test_SVM))
```

```
Test accuracy: 98.00%
ROC-AUC SCORE: 98.13%
              precision    recall  f1-score   support

           0       0.99      0.98      0.99      1107
           1       0.95      0.98      0.97       445

    accuracy                           0.98      1552
   macro avg       0.97      0.98      0.98      1552
weighted avg       0.98      0.98      0.98      1552
```

In [43]:

```
#For Decision tree
t = time()
detect_model = tree.DecisionTreeClassifier()
detect_model.fit(X_train , y_train)
pred_test_DT =  detect_model.predict(X_test)
acc_TEDT = accuracy_score(y_test,pred_test_DT)
test_accscores.append(acc_TEDT)
roau_TEDT= roc_auc_score(y_test, pred_test_DT )
test_roauscores.append(roau_TEDT)
times.append(time()-t)
print("Test accuracy: %.2f%%" % (acc_TEDT*100))#accuracy for test data/val
idation
print("ROC-AUC SCORE: %.2f%%" % (roau_TEDT*100))#accuracy for test data/va
lidation
print(classification_report(y_test, pred_test_DT))
```

```
Test accuracy: 95.04%
ROC-AUC SCORE: 94.57%
              precision    recall  f1-score   support

           0       0.97      0.96      0.96      1107
           1       0.90      0.93      0.92       445

    accuracy                           0.95      1552
   macro avg       0.93      0.95      0.94      1552
weighted avg       0.95      0.95      0.95      1552
```

In [44]:

```
estimator = ['Naive Bayes','Logistic Regression with SGD','SVM','Decision
 Tree']
d = {'Estimator': estimator,
     'Test accuracy score':test_accscores,
     'Test roc-auc score': test_roauscores,
     'Time to fit and predict (secs)': times
    }

df1 = pd.DataFrame(data=d).set_index('Estimator')
df1
```

Out[44]:

	Test accuracy score	Test roc-auc score	Time to fit and predict (secs)

Estimator			
Naive Bayes	0.891753	0.812580	0.018007
Logistic Regression with SGD	0.975515	0.976789	0.021588
SVM	0.980026	0.981295	3.703729
Decision Tree	0.950387	0.945736	1.678886

The highest accuracy observed was from Support Vector Machine which is 98.00% this is slightly above that of Logistic Regression with SGD which has an accuracy of 97.55%.Taking into consideration the cost of running these models,we can observe that SVM classifier took approximately 4 seconds to run compared to Logistic Regression which is approximately 0.02 seconds.We go with Logistic Regression as our best model here.This is followed by Decision Tree model which has 95.00% accuracy,Naive Bayes model has the least accuracy of 89.17%.The order is the same for area under the Receiver operating characteristics curve.

Precision-Recall Curves The Precision-recall diagrams help us determine which model perform better by measuring the area under the curve (AUC). The larger the area under the curve (AUC) the better the system. From our plot we get the highest AUC for SVM and Logistic Regression with SGD

Tuning of Parameters For every model, certain parameters are provided with a range of possibilities. These parameters are the ones that have high impact towards detecting the emails and learning rate. This will then be implemented within algorithms. I selected Logistic Regression with SGD as the best model for this machine learning classification based on accuracy and cost in terms of time in running the model.It can be further tuned using GridSearchCV. SGD Parameters: Hyperparameter tuning the algorithm offers 3 parameters from the SGD algorithm: Alpha values, Epsilon values and Tol values for the search space. Values for all three keys ranged from 0.0001 to 1000 as a dictionary.

In [46]:

```
pipe_SGD = Pipeline([ ('bow'   , CountVectorizer()),
                      ('tfidf'   , TfidfTransformer()),
                      ('clf_SGD' , SGDClassifier(random_state=5)),
 ])

parameters_SGD = {
    #'vect__ngram_range': ((1, 1), (1, 2)),  # unigrams or bigrams
    'tfidf__use_idf': (True, False),
    #'tfidf__norm': ('l1', 'l2'),
    #'clf_SGD__max_iter': (5,10),
    'clf_SGD__alpha': (1e-05, 1e-04),
}

grid_SGD = GridSearchCV(pipe_SGD, parameters_SGD, cv=5,
                              n_jobs=-1, verbose=1)

grid_SGD.fit(X=x_train, y=y_train)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concu
rrent workers.
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed:    9.3s
finished
```

Out[46]:

```
GridSearchCV(cv=5,
 estimator=Pipeline(steps=[('bow', CountVectorize
r()),
                                        ('tfidf', TfidfTransfo
rmer()),
                                        ('clf_SGD',
                                         SGDClassifier(random_
state=5))]),
             n_jobs=-1,
 param_grid={'clf_SGD__alpha': (1e-05, 0.0001),
 'tfidf__use_idf': (True, False)},          verbose=1)
```

In [47]:
```
grid_SGD.best_params_#Best
parameter
```

Out[47]:
```
{'clf_SGD__alpha': 0.0001,
'tfidf__use_idf': True}
```

In [48]:
```
grid_SGD.best_score_#Best score/cross
validation score
```

Out[48]:

```
0.9845303867403314
```

CONCLUSION

The project successfully implemented models combined with algorithms.Approximately 5,000 emails were tested with the proposed models.Splitting was done in the ratio of 70% for training set and 30% for testing set.In the first part,I first split before converting to numerical.In the second part I converted to numerical through feature extraction then split. I assigned both train sets as X_train_tfidf and X_train respectively.I observed the second part method was doing better with my model so I used that for all the Machine learning models carried out in this project. Four different classification algorithms was used (Multinomial Naïve Bayes,Logistic Regression with SGD, Support Vector Machine and Decision Tree).These algorithms were then tested and experimented with Scikit-learns library and its modules.The results were evaluated using different error evaluation technique. I concluded on the best algorithms which is Logistic Regression with SGD taking in accuracy and cost as a measure.This was further tuned to achieve the best score using the best parameter.A score of 98.45% was achieved. The highest accuracy provided was from SVM with score of 98.00% for 70:30 train and test split set on enron1 dataset.For evaluating the performance In terms of F1-Score, precision and recall,Support Vector Machine and Logistic Regression with SGD performed better than Decision Tree and Multinomial Naïve Bayes.

REFERENCE

-S. Gibson, B. Issac, L. Zhang and S. M. Jacob, "Detecting Spam Email With Machine Learning Optimized With Bio-Inspired Metaheuristic Algorithms," in IEEE Access, vol. 8, pp. 187914-187932, 2020, doi: 10.1109/ACCESS.2020.3030751. -Varghese, Reshma, and K. A. Dhanya. “Efficient Feature Set for Spam Email Filtering,” 732–37. IEEE Computer Society, 2017. https://doi.org/10.1109/IACC.2017.0152.