**ASSIGNMENT NO. 5**

**Name: Prajakta Mane**

**Class: BE-A**

**Roll No.: B1821016**

**PRN: 71901723J**

**Title:** Text classification for Sentiment analysis using KNN

# **Objectives:**

1. To handle Twitter Data for performing computing.
2. To analyze data using R programming tools.

# **Theory:**

Sentiment analysis refers to the use of natural language processing, text analysis, and computational linguistics to systematically identify, extract, quantify, and study effective states and subjective information. Sentiment analysis is widely applied to customer materials such as reviews and survey responses. The most common type of sentiment analysis is ‘polarity detection’ and involves classifying customer materials/reviews as positive, negative or neutral.

**Text Processing**

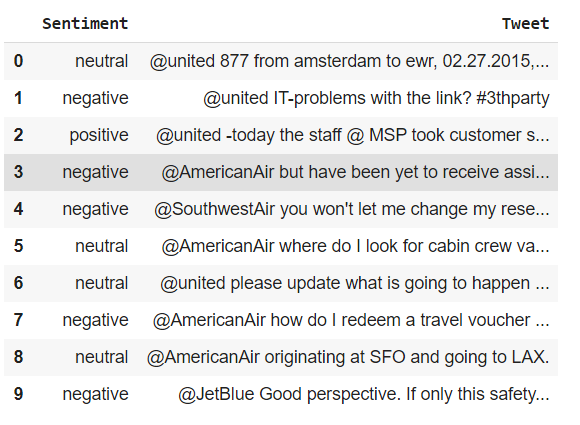
With the increasing importance of computational text analysis in research, many researchers face the challenge of learning how to use advanced software that enables this text analysis. Text processing has a direct application to Natural Language Processing, also known as NLP. NLP is aimed at processing the languages spoken or written by humans when they communicate with one another. This is different from the communication between a computer and a human where the communication is either a computer program written by a human or some gesture by a human like clicking the mouse at some position. NLP tries to understand the natural language spoken by humans and classify it, analyze it as well if required to respond to it. Python has a rich set of libraries which cater to the needs of NLP. The Natural Language ToolKit (NLTK) is a suite of such libraries which provides the functionalities required for NLP..

## **Twitter Data**

Twitter is an online microblogging tool that disseminates more than 400 million messages per day, including vast amounts of information about almost all industries from entertainment to sports, health to business etc. One of the best things about Twitter — indeed, perhaps its greatest appeal - is in its accessibility. It’s easy to use both for sharing information and for collecting it.Twitter provides unprecedented access to our lawmakers and to our celebrities, as well as to news as it’s happening. Twitter represents an important data source for the business models of huge companies as well. All the above characteristics make twitter a best place to collect real time and latest data to analyse and do any sought of research for real life situations.

## **DATASET DESCRIPTION**

We are given a [Twitter US Airline Sentiment](https://www.kaggle.com/crowdflower/twitter-airline-sentiment) dataset that contains around 14,601 tweets about each major U.S. airline. The tweets are labelled as positive, negative, or neutral based on the nature of the respective Twitter user’s feedback regarding the airline. The dataset is further segregated into training and test sets in a stratified fashion. Train set contains 11,680 tweets whereas the test set contains 2,921 tweets.Our task is to develop and train a k-nearest neighbors classifier on the training set and use it to predict sentiment classes of the tweets present in the test set. Here is a sneak-peek into the training dataset that we have got at our hands:



**Pre-Processing**

## Raw tweets scraped from twitter generally result in a noisy dataset. This is due to the casual nature of people’s usage of social media. Tweets have certain special characteristics such as retweets, emoticons, user mentions, etc. which have to be suitably extracted. Therefore, raw twitter data has to be normalized to create a dataset which can be easily learned by various classifiers. We have applied an extensive number of pre-processing steps to standardize the dataset and reduce its size. We first do some general pre-processing on tweets which is as follows.

## • Convert the tweet to lower case.

## • Replace 2 or more dots (.) with space.

## • Strip spaces and quotes (” and ’) from the ends of tweet.

## • Replace 2 or more spaces with a single space.

## Special twitter features as follows.

## **URL:**

## Users often share hyperlinks to other webpages in their tweets. Any particular URL is not important for text classification as it would lead to very sparse features. Therefore, we replace all the URLs in tweets with the word URL. The regular expression used to match URLs is ((www\.[\S]+)|(https?://[\S]+)).

## **User Mention**

## Every twitter user has a handle associated with them. Users often mention other users in their tweets by @handle. It replaces all user mentions with the word USER\_MENTION. The regular expression used to match user mention is @[\S]+.

## **K-Nearest Neighbours**

## K-Nearest Neighbours is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection.It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as [GMM](https://en.wikipedia.org/wiki/Mixture_model), which assume a Gaussian distribution of the given data).

KNN algorithm is used to classify by finding the K nearest matches in training data and then using the label of closest matches to predict. Traditionally, distance such as euclidean is used to find the closest match.KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

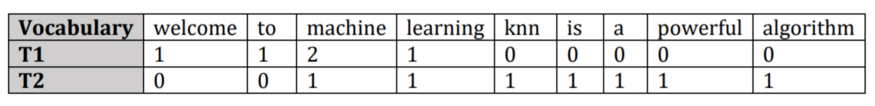
**Feature Extraction**

## In the feature extraction step, we will need to represent each tweet as a bag-of-words (BoW), i.e. an unordered set of words with their positions ignored and all of the emphasis placed on the respective frequencies of each word. For example, consider these two tweets:

## T1 = Welcome to machine learning, machine!

## T2 = kNN is a powerful machine learning algorithm.

## The bag-of-words representation (ignoring case and punctuation) for the above two tweets are:

In order to create this bag-of-words representation, we would first need to extract out the unique words from all of our tweets in the training dataset.

## 

## 

## 

## **Code for Twitter sentiment analysis**

**Importing Librarie**s

import pandas as pd

import re

import numpy as np

import scipy

import itertools

import matplotlib

import matplotlib.pyplot as plt

from scipy.spatial.distance import cdist

from collections import Counter

from random import choice

# Storing the training and test datasets into their respective dataframes

trained = pd.read\_csv('train.csv')

test = pd.read\_csv('test.csv')

# **Preprocessing**

trained['Tweet'] = trained['Tweet'].str.lower() # Ensuring all words in the Tweet column of training data are lowercased

test['Tweet'] = test['Tweet'].str.lower() # Ensuring all words in the Tweet column of test data are lowercased

# Parsing the stop\_words.txt file and storing all the words in a list.

stopwords = []

with open('stop\_words.txt','r') as file:

for line in file:

for word in line.split():

stopwords.append(word)

# Removing all stopwords from all the tweets in training data.

trained["Tweet"] = trained["Tweet"].apply(lambda func: ' '.join(sw

for sw in func.split()

if sw not in stopwords))

# Removing all stopwords from all the tweets in test data.

test["Tweet"] = test["Tweet"].apply(lambda func: ' '.join(sw

for sw in func.split()

if sw not in stopwords))

trained.head()

**Output:-**

|  | **Sentiment** | **Tweet** |
| --- | --- | --- |
| **0** | neutral | @united 877 amsterdam ewr, 02.27.2015, 737-300. |
| **1** | negative | @united it-problems link? #3thparty |
| **2** | positive | @united -today staff @ msp took customer servi... |
| **3** | negative | @americanair yet receive assistance one agents... |
| **4** | negative | @southwestair let change reservation online wa.. |

#List of all special characters that are to be removed.

special\_chars = ["!",'"',"%","&","amp","'","(",")", "\*","+",",","-",".",

"/",":",";","<","=",">","?","[","\\","]","^","\_",

"`”,"{","|","}","~","–","@","#","$"]

#Training Data

trained['Tweet'] = trained['Tweet'].str.replace(r'http?://[^\s<>"]+|www\.[^\s<>"]+', '') # Removing hyperlinks from all the tweets. They are not needed for classification.

trained['Tweet'] = trained['Tweet'].str.replace('@[A-Za-z0-9]+', '') # Removing usernames from all the tweets.

trained['Tweet'] = trained['Tweet'].str.replace(r'\B#\w\*[a-zA-Z]+\w\*', '') # Removing hashtags, including the text, from all the tweets. Hashtags are useless since their words cannot be splitted with spaces.

trained['Tweet'] = trained['Tweet'].str.replace('\d+', '')

# Removing numbers from all the tweets. They will not assist in any way to improve the classification process.

for c in special\_chars:

trained['Tweet'] = trained['Tweet'].str.replace(c,'')

# Removing all special characters from all the tweets

#Test Data

test['Tweet'] = test['Tweet'].str.replace(r'http?://[^\s<>"]+|www\.[^\s<>"]+', '') # Removing hyperlinks from all the tweets

test['Tweet'] = test['Tweet'].str.replace('@[A-Za-z0-9]+', '') # Removing usernames from all the tweets.

test['Tweet'] = test['Tweet'].str.replace(r'\B#\w\*[a-zA-Z]+\w\*', '') # Removing hashtags, including the text, from all the tweets

test['Tweet'] = test['Tweet'].str.replace('\d+', '') # Removing numbers from all the tweets

for c in special\_chars:

test['Tweet'] = test['Tweet'].str.replace(c,'') # Removing all special characters from all the tweets

trained.head()

**Output:-**

|  | **Sentiment** | **Tweet** |
| --- | --- | --- |
| **0** | neutral | amsterdam ewr |
| **1** | negative | itproblems link |
| **2** | positive | today staff msp took customer service to a new le... |
| **3** | negative | yet receive assistance one agents securing ne... |
| **4** | negative | let change reservation online wasting time |

test.head()

**Output:-**







|  | **Sentiment** | **Tweet** |
| --- | --- | --- |
| **0** | neutral | jump dallasaustin market news. |
| **1** | positive | chicago seen seat a aa far great ride pdx |
| **2** | negative | need bag bouncer get together |
| **3** | negative | hey jetblue stranded entire plane supposed go... |
| **4** | negative | big fail curbside baggage pittsburgh charge .. |



#Training Data

train\_unique = (list(set(trained['Tweet'].str.findall("\w+").sum()))) # Finding all the unique words in training data's Tweet column

train\_unique\_words = len(train\_unique)

#Test Data

test\_unique = (list(set(test['Tweet'].str.findall("\w+").sum()))) # Finding all the unique words in test data's Tweet column

test\_unique\_words = len(test\_unique)

print("Unique words in Training Data: {}".format(train\_unique\_words))

print("Unique words in Test Data: {}".format(test\_unique\_words))

**Output:-**

Unique words in Training Data: 10033

Unique words in Test Data: 4839

## 

## **Feature Extraction**

#Training Data

train\_matrix = [] # Forming a 2D matrix to store all training feature vectors

#Test Data

test\_matrix = [] # Forming a 2D matrix to store all test feature vectors

#Training Data: Extracting features and storing them into the training feature matrix

for sentence in trained['Tweet']:

train\_featurevec = []

word = sentence.split()

for w in train\_unique:

train\_featurevec.append(word.count(w))

train\_matrix.append(train\_featurevec)

#Test Data: Extracting features and storing them into the test feature matrix

for sentence in test['Tweet']:

test\_featurevec = []

word = sentence.split()

for w in train\_unique:

test\_featurevec.append(word.count(w))

test\_matrix.append(test\_featurevec)

print("Shape of Training Matrix: ({0} , {1})".format(len(train\_matrix),len(train\_matrix[0])))

print("Shape of Test Matrix: ({0} , {1})".format(len(test\_matrix),len(test\_matrix[0])))

**Output:-**

Shape of Training Matrix: (11680 , 10033)

Shape of Test Matrix: (2921 , 10033)

#Calculating distances between every test instance with all the train instances. This returns a 2D distances vector.

dists = cdist(test\_matrix,train\_matrix,'euclidean')

#Making an empty column in our test data for predicted labels.

test['Predicted Label'] = ''

dists.shape

**Output:**

(2921, 11680)

#Function that takes a list and returns the mode of the list. If there are more than one modes, it randomly selects any of them.

def get\_mode(l):

counting = Counter(l)

max\_count = max(counting.values())

return choice([ks for ks in counting if counting[ks] == max\_count])

## **K Nearest Neighbors & Performance Measures**

# Making a general structure of our confusion matrix

cmatrix = pd.DataFrame({'Gold Positive': '', 'Gold Neutral': '', 'Gold Negative': ''},

index = ['Predicted Positive','Predicted Neutral','Predicted Negative'])

# Lists that will later store respective values for plotting

accuracy\_list = []

recall\_list = []

precision\_list = []

F1\_list = []

def cmatrix\_measures(k,dists,test,cmatrix):

row\_count = 0

first\_max = 0

second\_max = 0

check\_tie = False

for ls in dists:

sorted\_distances\_indices = np.argsort(ls) #Getting a sorted list of indices of all distances in ls with the smallest distance's index at 0th position

knn\_indices = []

knn\_indices = list(itertools.islice(sorted\_distances\_indices,k)) #Extracting the indices of the k-smallest distances

knn\_labels = []

for i in knn\_indices:

label = trained['Sentiment'][i] #Extracting the label of the instance by indexing it through the DataFrame.

knn\_labels.append(label) #Appending the label to our labels list.

max\_class = get\_mode(knn\_labels)

first\_max = max\_class

second\_max = max(knn\_labels)

if first\_max == second\_max:

check\_tie = True

predicted\_label = max\_class

test['Predicted Label'][row\_count] = predicted\_label

row\_count += 1

#Creating a frequency DataFrame that will store value counts for each tuple of instances. E.g (positive,positive = 309) and so on for all other seven instances.

testfreqdf = test.groupby(["Sentiment", "Predicted Label"]).size().reset\_index(name="Frequency")

testfreqdf

#Extracting values from the Frequency DataFrame and assigning to specific cells in the confusion matrix.

cmatrix['Gold Positive']['Predicted Positive'] = testfreqdf['Frequency'][8]

cmatrix['Gold Neutral']['Predicted Positive'] = testfreqdf['Frequency'][5]

cmatrix['Gold Negative']['Predicted Positive'] = testfreqdf['Frequency'][2]

cmatrix['Gold Positive']['Predicted Neutral'] = testfreqdf['Frequency'][7]

cmatrix['Gold Neutral']['Predicted Neutral'] = testfreqdf['Frequency'][4]

cmatrix['Gold Negative']['Predicted Neutral'] = testfreqdf['Frequency'][1]

cmatrix['Gold Positive']['Predicted Negative'] = testfreqdf['Frequency'][6]

cmatrix['Gold Neutral']['Predicted Negative'] = testfreqdf['Frequency'][3]

cmatrix['Gold Negative']['Predicted Negative'] = testfreqdf['Frequency'][0]

#Extracting all three True Positives from the matrix to measure accuracy.

TP = cmatrix['Gold Positive']['Predicted Positive']

TNT = cmatrix['Gold Neutral']['Predicted Neutral']

TN = cmatrix['Gold Negative']['Predicted Negative']

total = testfreqdf['Frequency'].sum()

accuracy = ((TP+TNT+TN)/total)\*100

accuracy = round(accuracy,2)

accuracy\_list.append(accuracy)

#Extracting all recalls from the matrix to measure macroaveraged recall.

recall\_pos = cmatrix['Gold Positive']['Predicted Positive']/cmatrix['Gold Positive'].sum()

recall\_neut = cmatrix['Gold Neutral']['Predicted Neutral']/cmatrix['Gold Neutral'].sum()

recall\_neg = cmatrix['Gold Negative']['Predicted Negative']/cmatrix['Gold Negative'].sum()

macroaveraged\_recall = ((recall\_pos+recall\_neut+recall\_neg)/3)\*100

macroaveraged\_recall = round(macroaveraged\_recall,2)

recall\_list.append(macroaveraged\_recall)

#Extracting all precisions from the matrix to measure macroaveraged precision.

precision\_pos = cmatrix['Gold Positive']['Predicted Positive']/(cmatrix.iloc[0,0:3].sum())

precision\_neut = cmatrix['Gold Neutral']['Predicted Neutral']/(cmatrix.iloc[1,0:3].sum())

precision\_neg = cmatrix['Gold Negative']['Predicted Negative']/(cmatrix.iloc[2,0:3].sum())

macroaveraged\_precision = ((precision\_pos+precision\_neut+precision\_neg)/3)\*100

macroaveraged\_precision = round(macroaveraged\_precision,2)

precision\_list.append(macroaveraged\_precision)

#Extracting all F1\_scores from the matrix to measure macroaveraged F1\_score.

F1\_pos = (2\*precision\_pos\*recall\_pos)/(precision\_pos+recall\_pos)

F1\_neut = (2\*precision\_neut\*recall\_neut)/(precision\_neut+recall\_neut)

F1\_neg = (2\*precision\_neg\*recall\_neg)/(precision\_neg+recall\_neg)

F1\_score = ((F1\_pos + F1\_neut + F1\_neg)/3)\*100

F1\_score = round(F1\_score,2)

F1\_list.append(F1\_score)

print("\n\nConfusion Matrix with k = {}:\n".format(k))

print(cmatrix)

print("\nAccuracy with k = {0}: {1}%".format(k,accuracy))

print("Macroaveraged Precision with k = {0}: {1}%".format(k,macroaveraged\_precision))

print("Macroaveraged Recall with k = {0}: {1}%".format(k,macroaveraged\_recall))

print("Macroaveraged F1-score with k = {0}: {1}%".format(k,F1\_score))

#Calling the function for each individual k

cmatrix\_measures(1,dists,test,cmatrix)

cmatrix\_measures(3,dists,test,cmatrix)

cmatrix\_measures(5,dists,test,cmatrix)

cmatrix\_measures(7,dists,test,cmatrix)

cmatrix\_measures(10,dists,test,cmatrix)

**Output:**

Confusion Matrix with k = 1:

Gold Positive Gold Neutral Gold Negative

Predicted Positive 265 90 188

Predicted Neutral 152 409 801

Predicted Negative 55 116 845

Accuracy with k = 1: 52.0%

Macroaveraged Precision with k = 1: 54.0%

Macroaveraged Recall with k = 1: 56.24%

Macroaveraged F1-score with k = 1: 50.96%

Confusion Matrix with k = 3:

Gold Positive Gold Neutral Gold Negative

Predicted Positive 280 101 179

Predicted Neutral 143 421 862

Predicted Negative 49 93 793

Accuracy with k = 3: 51.15%

Macroaveraged Precision with k = 3: 54.78%

Macroaveraged Recall with k = 3: 57.01%

Macroaveraged F1-score with k = 3: 50.93%

Confusion Matrix with k = 5:

Gold Positive Gold Neutral Gold Negative

Predicted Positive 284 109 185

Predicted Neutral 144 424 869

Predicted Negative 44 82 780

Accuracy with k = 5: 50.94%

Macroaveraged Precision with k = 5: 54.91%

Macroaveraged Recall with k = 5: 57.21%

Macroaveraged F1-score with k = 5: 50.79%

Confusion Matrix with k = 7:

Gold Positive Gold Neutral Gold Negative

Predicted Positive 295 108 173

Predicted Neutral 142 445 926

Predicted Negative 35 62 735

Accuracy with k = 7: 50.5%

Macroaveraged Precision with k = 7: 56.32%

Macroaveraged Recall with k = 7: 58.31%

Macroaveraged F1-score with k = 7: 51.09%

Confusion Matrix with k = 10:

Gold Positive Gold Neutral Gold Negative

Predicted Positive 298 103 189

Predicted Neutral 136 446 937

Predicted Negative 38 66 708

Accuracy with k = 10: 49.71%

Macroaveraged Precision with k = 10: 55.69%

Macroaveraged Recall with k = 10: 58.09%

Macroaveraged F1-score with k = 10: 50.48%

**Plotting (Part-1)**

k\_list = [1,3,5,7,10]

fig = plt.figure(figsize=(12,8))

plt.subplot(2,2,1)

plt.plot(k\_list,accuracy\_list)

plt.title("Accuracy as KNN increase",fontsize='x-large')

plt.xlabel("k Nearest Neighbors",fontsize='large')

plt.ylabel("Accuracy",fontsize='large')

plt.subplot(2,2,2)

plt.plot(k\_list,recall\_list)

plt.title("Recall as KNN increase",fontsize='x-large')

plt.xlabel("k Nearest Neighbors",fontsize='large')

plt.ylabel("Recall",fontsize='large')

plt.subplot(2,2,3)

plt.plot(k\_list,precision\_list)

plt.title("Precision as KNN increase",fontsize='x-large')

plt.xlabel("k Nearest Neighbors",fontsize='large')

plt.ylabel("Precision",fontsize='large')

plt.subplot(2,2,4)

plt.plot(k\_list,F1\_list)

plt.title("F1 as KNN increase",fontsize='x-large')

plt.xlabel("k Nearest Neighbors",fontsize='large')

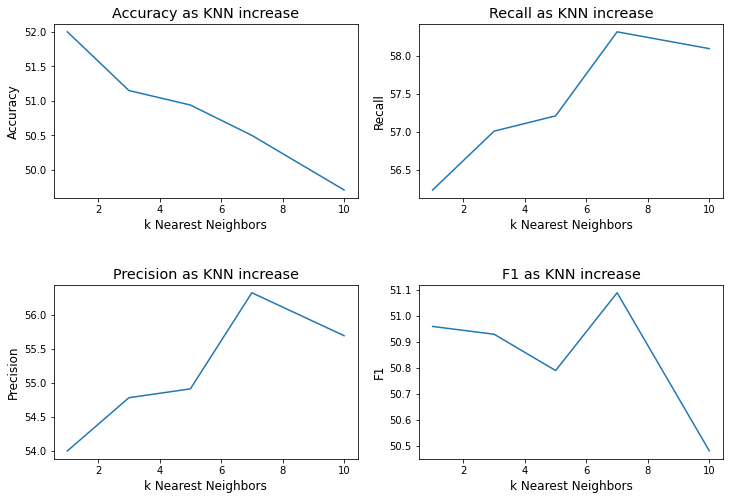
plt.ylabel("F1",fontsize='large')

fig.subplots\_adjust(hspace=.5)

plt.show

**Output:**

<function matplotlib.pyplot.show(close=None, block=None)>



# 

# **Part 2**

# from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn import metrics

from sklearn.metrics import classification\_report, confusion\_matrix

#Initializing lists to be used in plotting later.

acc\_list = []

rec\_list = []

prec\_list = []

f1\_list = []

# **KNN & Performance Measures with SKLearn**

def classifying(trainfeatures,testfeatures):

X\_train = trainfeatures #train\_matrix - 2D feature vector

X\_test = testfeatures #test\_matrix - 2D feature vector

y\_train = trained.iloc[:,0].values #trained['Sentiment]

y\_test = test.iloc[:,0].values #test['Sentiment]

for k in [1,3,5,7,10]:

classifier = KNeighborsClassifier(n\_neighbors=k,algorithm='brute') #Using brute-force algorithm for quicker computation.

classifier.fit(X\_train, y\_train) #Fitting the built-in sklearn classifier on our training data

predicted\_label = classifier.predict(X\_test) #Making the classifier to predict on the previously unseen test data.

accuracy\_score = (metrics.accuracy\_score(y\_test,predicted\_label))

accuracy\_score = (round(accuracy\_score,2))\*100

acc\_list.append(accuracy\_score)

confusion\_mat = confusion\_matrix(y\_test, predicted\_label)

class\_report = classification\_report(y\_test, predicted\_label)

macro\_precision = (metrics.precision\_score(y\_test, predicted\_label, average='macro'))

macro\_precision = (round(macro\_precision,2))\*100

prec\_list.append(macro\_precision)

macro\_recall = (metrics.recall\_score(y\_test, predicted\_label, average='macro'))

macro\_recall = (round(macro\_recall,2))\*100

rec\_list.append(macro\_recall)

macro\_f1 = (metrics.f1\_score(y\_test, predicted\_label, average='macro'))

macro\_f1 = (round(macro\_f1,2))\*100

f1\_list.append(macro\_f1)

print("\n\nConfusion Matrix for k = {} is:\n".format(k))

print(confusion\_mat)

print("\nClassification Report for k = {} is:\n".format(k))

print(class\_report)

print("Accuracy Score for k = {0} is: {1}%".format(k,accuracy\_score))

print("Macroaveraged Recall for k = {0} is: {1}%".format(k,macro\_recall))

print("Macroaveraged Precision for k = {0} is: {1}%".format(k,macro\_precision))

print("Macroaveraged F1-score for k = {0} is: {1}%".format(k,macro\_f1))

classifying(train\_matrix,test\_matrix)

**Output:**

Confusion Matrix for k = 1 is:

[[873 773 188]

[122 416 77]

[ 69 180 223]]

Classification Report for k = 1 is:

precision recall f1-score support

negative 0.82 0.48 0.60 1834

neutral 0.30 0.68 0.42 615

positive 0.46 0.47 0.46 472

accuracy 0.52 2921

macro avg 0.53 0.54 0.50 2921

weighted avg 0.65 0.52 0.54 2921

Accuracy Score for k = 1 is: 52.0%

Macroaveraged Recall for k = 1 is: 54.0%

Macroaveraged Precision for k = 1 is: 53.0%

Macroaveraged F1-score for k = 1 is: 50.0%

Confusion Matrix for k = 3 is:

[[905 801 128]

[126 425 64]

[ 71 168 233]]

Classification Report for k = 3 is:

precision recall f1-score support

negative 0.82 0.49 0.62 1834

neutral 0.30 0.69 0.42 615

positive 0.55 0.49 0.52 472

accuracy 0.54 2921

macro avg 0.56 0.56 0.52 2921

weighted avg 0.67 0.54 0.56 2921

Accuracy Score for k = 3 is: 54.0%

Macroaveraged Recall for k = 3 is: 56.00000000000001%

Macroaveraged Precision for k = 3 is: 56.00000000000001%

Macroaveraged F1-score for k = 3 is: 52.0%

Confusion Matrix for k = 5 is:

[[838 868 128]

[106 432 77]

[ 66 142 264]]

Classification Report for k = 5 is:

precision recall f1-score support

negative 0.83 0.46 0.59 1834

neutral 0.30 0.70 0.42 615

positive 0.56 0.56 0.56 472

accuracy 0.53 2921

macro avg 0.56 0.57 0.52 2921

weighted avg 0.67 0.53 0.55 2921

Accuracy Score for k = 5 is: 53.0%

Macroaveraged Recall for k = 5 is: 56.99999999999999%

Macroaveraged Precision for k = 5 is: 56.00000000000001%

Macroaveraged F1-score for k = 5 is: 52.0%

Confusion Matrix for k = 7 is:

[[771 902 161]

[ 86 441 88]

[ 44 147 281]]

Classification Report for k = 7 is:

precision recall f1-score support

negative 0.86 0.42 0.56 1834

neutral 0.30 0.72 0.42 615

positive 0.53 0.60 0.56 472

accuracy 0.51 2921

macro avg 0.56 0.58 0.51 2921

weighted avg 0.69 0.51 0.53 2921

Accuracy Score for k = 7 is: 51.0%

Macroaveraged Recall for k = 7 is: 57.99999999999999%

Macroaveraged Precision for k = 7 is: 56.00000000000001%

Macroaveraged F1-score for k = 7 is: 51.0%

Confusion Matrix for k = 10 is:

[[755 901 178]

[ 71 452 92]

[ 43 143 286]]

Classification Report for k = 10 is:

precision recall f1-score support

negative 0.87 0.41 0.56 1834

neutral 0.30 0.73 0.43 615

positive 0.51 0.61 0.56 472

accuracy 0.51 2921

macro avg 0.56 0.58 0.51 2921

weighted avg 0.69 0.51 0.53 2921

Accuracy Score for k = 10 is: 51.0%

Macroaveraged Recall for k = 10 is: 57.99999999999999%

Macroaveraged Precision for k = 10 is: 56.00000000000001%

Macroaveraged F1-score for k = 10 is: 51.0%

# 

# 

# **Plotting (Part-2)**

k\_ls = [1,3,5,7,10]

fig = plt.figure(figsize=(12,8))

plt.subplot(2,2,1)

plt.plot(k\_ls,acc\_list)

plt.title("Accuracy as KNN increase",fontsize='x-large')

plt.xlabel("k Nearest Neighbors",fontsize='large')

plt.ylabel("Accuracy",fontsize='large')

plt.subplot(2,2,2)

plt.plot(k\_ls,rec\_list)

plt.title("Recall as KNN increase",fontsize='x-large')

plt.xlabel("k Nearest Neighbors",fontsize='large')

plt.ylabel("Recall",fontsize='large')

plt.subplot(2,2,3)

plt.plot(k\_ls,prec\_list)

plt.title("Precision as KNN increase",fontsize='x-large')

plt.xlabel("k Nearest Neighbors",fontsize='large')

plt.ylabel("Precision",fontsize='large')

plt.subplot(2,2,4)

plt.plot(k\_ls,f1\_list)

plt.title("F1 as KNN increase",fontsize='x-large')

plt.xlabel("k Nearest Neighbors",fontsize='large')

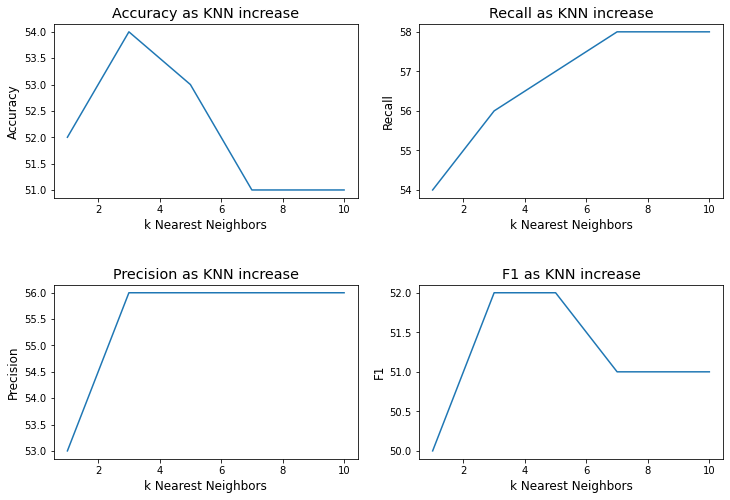
plt.ylabel("F1",fontsize='large')

fig.subplots\_adjust(hspace=.5)

Plt.show

**Output:**

<function matplotlib.pyplot.show(close=None, block=None)>



**Conclusion:**

Hence, we studi ed On Twitter Data performs computing using Business Intelligence analytical tools electively.