In [2]:
```
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
```

In [3]:
```
# Storing the training and test datasets into their respective dataframes
trained = pd.read_csv('train_clean.csv')
test = pd.read_csv('test_clean.csv')
```

In [4]:
```
trained.head()
```

Out[4]:

	Unnamed: 0	Sentiment	Tweet
0	0	neutral	amsterdam ewr
1	1	negative	ITproblems link
2	2	positive	today staff MSP took customer service new le...
3	3	negative	yet receive assistance one agents securing ne...
4	4	negative	let change reservation online Im wasting time

In [5]:
```
test.head()
```

Out[5]:

	Unnamed: 0	Sentiment	Tweet
0	0	neutral	jump DallasAustin market News
1	1	positive	Chicago seen seat A AA So far great ride On ...
2	2	negative	need bag bouncer Get together
3	3	negative	Hey Jetblue stranded entire plane supposed go...
4	4	negative	Big fail curbside baggage Pittsburgh charge ...

In [8]:
```
#Training Data
train_unique = (list(set(trained['Tweet'].str.findall("\w+").sum()))) # Finding a
train_unique_words = len(train_unique)

#Test Data
test_unique = (list(set(test['Tweet'].str.findall("\w+").sum()))) # Finding all t
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))
```

```
Unique words in Training Data: 12416
Unique words in Test Data: 5814
```

Feature Extraction

In [9]:
```
#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
```

In [10]:
```
#Training Data: Extracting features and storing them into the training feature ma
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)
```

In [11]:
```
print("Shape of Training Matrix: ({0} , {1})".format(len(train_matrix),len(train_
print("Shape of Test Matrix: ({0} , {1})".format(len(test_matrix),len(test_matrix
```

```
Shape of Training Matrix: (11680 , 12416)
Shape of Test Matrix: (2921 , 12416)
```

In [12]:
```
#Calculating distances between every test instance with all the train instances.
dists = cdist(test_matrix,train_matrix,'euclidean')
```

In [13]:
```
#Making an empty column in our test data for predicted labels.
test['Predicted Label'] = ''
dists.shape
```

Out[13]: (2921, 11680)

In [14]:
```
#Function that takes a list and returns the mode of the list. If there are more t
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

In [15]:
```
# Making a general structure of our confusion matrix
cmatrix = pd.DataFrame({'Gold Positive': '', 'Gold Neutral': '', 'Gold Negative':
                       index = ['Predicted Positive','Predicted Neutral','Predict

# Lists that will later store respective values for plotting
accuracy_list = []
recall_list = []
precision_list = []
F1_list = []
```

In [16]:
```
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 o
    knn_indices = []
    knn_indices = list(itertools.islice(sorted_distances_indices,k)) #Extracting

    knn_labels = []
    for i in knn_indices:
      label = trained['Sentiment'][i] #Extracting the label of the instance by in
      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
  testfreqdf = test.groupby(["Sentiment", "Predicted Label"]).size().reset_index(
  testfreqdf

  #Extracting values from the Frequency DataFrame and assigning to specific cells
  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 Posit
  recall_neut = cmatrix['Gold Neutral']['Predicted Neutral']/cmatrix['Gold Neutra
  recall_neg = cmatrix['Gold Negative']['Predicted Negative']/cmatrix['Gold Negat
  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,
precision_neut = cmatrix['Gold Neutral']['Predicted Neutral']/(cmatrix.iloc[1,0
precision_neg = cmatrix['Gold Negative']['Predicted Negative']/(cmatrix.iloc[2,
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_preci
print("Macroaveraged Recall with k = {0}: {1}%".format(k,macroaveraged_recall))
print("Macroaveraged F1-score with k = {0}: {1}%".format(k,F1_score))
```

In [17]:
```
#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)
```

```
C:\Users\aj240\AppData\Local\Temp/ipykernel_112/4185092973.py:24: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  test['Predicted Label'][row_count] = predicted_label



Confusion Matrix with k = 1:

                   Gold Positive Gold Neutral Gold Negative
Predicted Positive           263          107           205
Predicted Neutral            132          343           758
Predicted Negative            77          165           871

Accuracy with k = 1: 50.56%
Macroaveraged Precision with k = 1: 50.6%
Macroaveraged Recall with k = 1: 52.99%
Macroaveraged F1-score with k = 1: 48.82%

C:\Users\aj240\AppData\Local\Temp/ipykernel_112/4185092973.py:24: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  test['Predicted Label'][row_count] = predicted_label



Confusion Matrix with k = 3:

                   Gold Positive Gold Neutral Gold Negative
Predicted Positive           273          124           235
Predicted Neutral            135          363           783
Predicted Negative            64          128           816

Accuracy with k = 3: 49.71%
Macroaveraged Precision with k = 3: 50.83%
Macroaveraged Recall with k = 3: 53.79%
Macroaveraged F1-score with k = 3: 48.39%

C:\Users\aj240\AppData\Local\Temp/ipykernel_112/4185092973.py:24: SettingWith
CopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  test['Predicted Label'][row_count] = predicted_label



Confusion Matrix with k = 5:

                   Gold Positive Gold Neutral Gold Negative
Predicted Positive           272          133           250
Predicted Neutral            138          376           814
Predicted Negative            62          106           770

Accuracy with k = 5: 48.55%
Macroaveraged Precision with k = 5: 50.64%
Macroaveraged Recall with k = 5: 53.58%
Macroaveraged F1-score with k = 5: 47.51%

C:\Users\aj240\AppData\Local\Temp/ipykernel_112/4185092973.py:24: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  test['Predicted Label'][row_count] = predicted_label



Confusion Matrix with k = 7:

                   Gold Positive Gold Neutral Gold Negative
Predicted Positive           275          121           247
Predicted Neutral            153          397           871
Predicted Negative            44           97           716

Accuracy with k = 7: 47.52%
Macroaveraged Precision with k = 7: 51.42%
Macroaveraged Recall with k = 7: 53.95%
Macroaveraged F1-score with k = 7: 47.18%

C:\Users\aj240\AppData\Local\Temp/ipykernel_112/4185092973.py:24: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  test['Predicted Label'][row_count] = predicted_label
```

```
Confusion Matrix with k = 10:

                   Gold Positive Gold Neutral Gold Negative
Predicted Positive           275          112           252
Predicted Neutral            158          424           893
Predicted Negative            39           79           689

Accuracy with k = 10: 47.52%
Macroaveraged Precision with k = 10: 52.39%
Macroaveraged Recall with k = 10: 54.92%
Macroaveraged F1-score with k = 10: 47.42%
```

In [ ]: