

2 Sentiment Analysis

2.2 Movie Review Data

Let us first start by looking at the data provided with the exercise. We have positive and negative movie reviews labeled by human readers, all positive and negative reviews are in the ‘pos’ and ‘neg’ folders respectively. If you look inside a sample file, you will see that these review messages have been ‘tokenized’, where all words are separated from punctuations. There are approximately 1000 files in each category with files names starting with cv000, cv001, cv002 and so on. You will split the dataset into training set and testing set.

1. Write some code to load the data from text files.

```
In [1]: import numpy as np
import pandas as pd
import sklearn
from scipy import stats, integrate
import matplotlib.pyplot as plt
import seaborn as sns
import glob
import os
Freelist = []
list_neg = []
list_pos = []
File_txt = 0
Freelist = []
```

```
In [2]: File_txt = glob.glob('review_polarity/txt_sentoken/**/*.txt')

for i in File_txt:
    REad_File= open(i,'r')
    Freelist.append(REad_File.read())
    REad_File.close
print(len(Freelist))
# Freelist
# Freelist
```

2000

```
In [42]: # we read all file in directory ('review_polarity/txt_sentoken/**/*.txt')
# we will get neg first and pos second and then we read txt file with open(Fil
ename, 'r')
# we append word to Freelist and we Get All data neg and pos in Freelist
```

```
In [1]: # Freelist
```

2.3 TF-IDF

From a raw text review, you want to create a vector, whose elements indicate the number of each word in each document. The frequency of all words within the documents are the ‘features’ of this machine learning problem.

A popular method for transforming a text to a vector is called tf-idf, short for term frequencyinverse document frequency.

1. Conduct a research about tf-idf and explain how it works.
2. Scikit-learn provides a module for calculating this, this is called TfidfVectorizer. You can study how this function is used here:

```
http://scikit-learn.org/stable/modules/generated/sklearn.feature\_extraction.text.TfidfVectorizer.html
```

Write code to transform your text to tf-idf vector.

```
In [43]: from sklearn.feature_extraction.text import TfidfVectorizer

data_n = TfidfVectorizer()
TF_IDF_DATA = data_n.fit_transform(Freelist)
TF_IDF_DATA

#2.3.1 TF-IDF เป็นการนับจำนวนคำข้าวที่มีอยู่ใน txt ไฟล์แล้วหารด้วยคำทั้งหมดที่มี โดยจะมี TF term ค่อยเก็บ คำ และ IDF term ที่ค่อยเก็บ คำ weight ของแต่ละคำ โดยค่านวนจากคำที่มีคำข้าวมาก
#ที่จะ weight ค่าน้อย เช่น number one, number two ,number three คำว่า number ก็จะ weight ค่าน้อยเนื่องจาก มีการใช้คำนี้很多
#ทำให้การใช้ TF-IDF สามารถลดคำของที่มีความสำคัญน้อย ในข้อมูลของเราลงได้ โดยการคูณ ระหว่าง TF term และ IDF term
# fit transform and mean fit() and transform() on the same data
```

```
Out[43]: <2000x39659 sparse matrix of type '<class 'numpy.float64'>' with 666842 stored elements in Compressed Sparse Row format>
```

2.4 Classification

Use 4 different models to classify each movie into positive or negative category.

1. K-Nearestneighbormodel,using module sklearn.neighbors.KNeighborsClassifier
2. RandomForest, using module sklearn.ensemble.RandomForestClassifier
3. SVM, using module sklearn.svm.SVC
4. Neural network, using sklearn.neural_network.MLPClassifier

You may pick other models you would like to try. Just present results for at least 4 models. Please provide your code for model fitting and cross validation. Calculate your classification accuracy, precision, and recall.

```
In [5]: from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score

from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_predict
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
```

```
In [6]: y_label = ([0]*1000+[1]*1000)
# Y_Label เป็น Label ที่เราจำลองมาจากการอ่านค่า ของไฟล์ข้างบนซึ่งเราเอาค่า ของไฟล์ Neg มา
# วน เลยให้ Label ตัวแรกเป็น 0 1000ตัว ตาม Len ของList neg
# และ เป็น 1 อีก1000 ตัวตาม List ของ pos โดยเราจำลอง LABEL NEG = 0 POS = 1
```

```
In [7]: Classification = {"KNearest":KNeighborsClassifier() , "RandomForest":RandomForestClassifier(), "SVM":SVC(), "NeuralNetwork":MLPClassifier()}

for name,classi in Classification.items():
    Model =cross_val_predict(classi , TF_IDF_DATA, y_label)
    print("Classi = ",name)
    print("ACCURACY "+ name +" = " ,accuracy_score(y_label,Model))

    print(classification_report(y_label,Model))

#การใช้ sklearn.model_selection.cross_val_predict(estimator,x,y) คือจะมี3ตัวแปรคือ
#estimator เป็น object ที่ fit กับ กับ predict model
#x จะเป็นค่า Arrayที่เอาไว้ fit เช่น list หรือ arrayที่มีขั้นต่ำ 2 มิติขึ้นไป
#y จะเป็นค่าที่จะไว้ พยายาม จะ predict ค่าใหม่ ใน case ของ supervised Learning
#การใช้ sklearn.metrics.classification_report(y_true, y_pred, labels=None, target_names=None,
#sample_weight=None, digits=2)
#จะมี 2 ตัวแปรหลักๆที่ใช้ คือ y_trueกับ ypred
# โดย y_true คือ ค่า correct Value
#ส่วน ypred จะเป็นค่าที่ Estimate มาจาก target เพื่อนำมาเทียบกัน และหาค่าต่างๆ อย่างเช่น Precision recall f1-score
#ซึ่งค่าต่างๆ ของแต่ละ ค่าตอบ อยู่ตามข้างล่างนี้
```

```

Classi = KNearest
ACCURACY KNearest = 0.572
      precision    recall   f1-score   support
      0          0.80     0.19     0.31      1000
      1          0.54     0.95     0.69      1000
avg / total       0.67     0.57     0.50      2000

Classi = RandomForest
ACCURACY RandomForest = 0.6565
      precision    recall   f1-score   support
      0          0.63     0.77     0.69      1000
      1          0.70     0.55     0.61      1000
avg / total       0.66     0.66     0.65      2000

Classi = SVM
ACCURACY SVM = 0.7585
      precision    recall   f1-score   support
      0          0.80     0.70     0.74      1000
      1          0.73     0.82     0.77      1000
avg / total       0.76     0.76     0.76      2000

Classi = NeuralNetwork
ACCURACY NeuralNetwork = 0.824
      precision    recall   f1-score   support
      0          0.83     0.81     0.82      1000
      1          0.82     0.84     0.83      1000
avg / total       0.82     0.82     0.82      2000

```

2.5 Model Tuning

Can you try to beat the simple model you created above? Here are some things you may try:

- When creating TfidfVectorizer object, you may tweak sublinear_tf parameter which use the tf with logarithmic scale instead of the usual tf.
- You may also exclude words that are too frequent or too rare, by adjusting max_df and min_df.
- Adjusting parameters available in the model, like neural network structure or number of trees in the forest.

Design at least 3 experiments using these techniques. Show your experimental results.

```
In [8]: Set_Sublin_Set_Tresh = TfidfVectorizer(sublinear_tf = True ,max_df = 0.6, min_
df = 0.03)

TF_IDF_DATA_NEW = Set_Sublin_Set_Tresh.fit_transform(Freelist)
```

```
In [ ]: #sublinear_tf = superlinear rates of convergence ของ tf โดย replace จาก tf เป็น  
tf = 1+log(tf) ทำให้ค่าrate มากขึ้น  
#max_df = ปรับค่า Threshold จาก 1.0(default) เป็น0.6 ตัดค่าที่มี ค่าซ้ำมากเกินไป ออกจาก  
document  
#           0.6means "ignore terms that appear in more than 60% of the docu  
ments".  
#min_df = ปรับค่า Threshold จาก 1(default) เป็น0.03 ตัดค่าที่มีน้อยเกินไป จน ทำให้ไม่มีผล  
ต่อ document  
#           0.03 ignore terms that appear in less than 3% of the documents  
#
```

```
In [10]: Classification = {"KNearest":KNeighborsClassifier(n_neighbors=70),"RF":RandomForestClassifier(n_estimators = 150,random_state=0),
                         "Neural":MLPClassifier(solver='lbfgs',hidden_layer_sizes=(20,20), random_state=1)}
for name,classi in Classification.items():
    if name == "Neural":
        Model =cross_val_predict(classi , TF_IDF_DATA, y_label)
        print("Classi = ",name)
        print("ACCURACY "+ name ,accuracy_score(y_label,Model))
        print(classification_report(y_label,Model))
    else:
        Model =cross_val_predict(classi , TF_IDF_DATA_NEW, y_label)
        print("Classi = ",name)
        print("ACCURACY "+ name ,accuracy_score(y_label,Model))
        print(classification_report(y_label,Model))

#การใช้ sklearn.model_selection.cross_val_predict(estimator,x,y) คือจะมี 3 ตัวแปรคือ
#estimator เป็น object ที่ fit กับ predict model
#x จะเป็นค่า Array ที่เอาไว้ fit เช่น List หรือ array ที่มีขั้นต่ำ 2 มิติขึ้นไป
#y จะเป็นค่าที่จะไว้พยากรณ์ จะ predict ค่าใหม่ ใน case ของ supervised Learning
#การใช้ sklearn.metrics.classification_report(y_true, y_pred, labels=None, target_names=None,
#sample_weight=None, digits=2)
#จะมี 2 ตัวแปรหลักๆ ที่ใช้ คือ y_true กับ ypred
#โดย y_true คือ ค่า correct Value
#ส่วน ypred จะเป็นค่าที่ Estimate มาจาก target เพื่อนำมาเทียบกัน และหาค่าต่างๆ อย่างเช่น
#Precision recall f1-score
#ชึ่งค่าต่างๆ ของแต่ละ คำตอบ อยู่ตามข้างล่างนี้

#เหมือนข้อข้างบนแต่มี จุดเปลี่ยนอยู่หลายจุด เช่น
# -----KNeighborsClassifier -----
#ปรับค่า n_neighbors จาก 5(default) เป็น 70 โดยสนใจตัวออดข้างมากขึ้น ทำให้ค่า ที่ระหว่าง 0 กับ 1 ในช่วง 70 ตัว จะมีผลต่อค่า Data มากขึ้น และถ้าหาก
#เป็น ค่า 0 1 โดยการ random แล้ว n_neighbors จะมีผลมากๆ และบางครั้งอาจจะทำให้ข้อมูล overfit โดยต้องเลือกการใช้ค่านี้ดีๆ และในกรณีนี้ เป็น 0 กับ 1 ที่เป็นค่าเรียงกัน
#จึงมีผลกับ 0 1 ในช่วง 70 ตัว
# -----RandomForestClassifier -----
#n_estimators จาก 10(default) เป็น 150 คือจาก สูมแค่เทียบ 10 tree
#เปลี่ยนเป็น 150 เพื่อหา อันที่ดีมากยิ่งขึ้น ทำให้เพิ่มประสิทธิภาพการทำงาน
#และ set random_state จาก None (default) เป็น 0
# เพราะ ตอนแรก หากเป็น None จะ random ไปเรื่อยๆ ตาม numpy.random เราเลย พิคค่าไว้
#เพื่อเพิ่มประสิทธิภาพของการทำงาน
# -----Neural Network -----
#ปรับ solver จาก adam(default) เป็น lbfgs
# adam จะ ทำงานได้ดีใน Data ที่มีน้อยมากๆ (thousands of training sample หรือมากกว่านั้น) จะดี
# lbfgs ทำงานได้ดีใน small dataset faster and perform better **** เร็วกว่ามากๆ
# hidden_layer_sizes : tuple, length = n_layers - 2, default (100, )
# เปลี่ยนจาก 100,Length เป็น (20,20) ทั้งหมดเพื่อเพิ่มค่า ความแม่นยำ จากการทดลองหลากๆแบบ
```

```

Classi = KNearest
ACCURACY KNearest 0.7675
      precision    recall   f1-score   support
      0          0.84     0.66     0.74      1000
      1          0.72     0.88     0.79      1000
avg / total       0.78     0.77     0.76      2000

Classi = RF
ACCURACY RF 0.8055
      precision    recall   f1-score   support
      0          0.78     0.85     0.81      1000
      1          0.84     0.76     0.80      1000
avg / total       0.81     0.81     0.81      2000

Classi = Neural
ACCURACY Neural 0.838
      precision    recall   f1-score   support
      0          0.84     0.84     0.84      1000
      1          0.84     0.84     0.84      1000
avg / total       0.84     0.84     0.84      2000

```

3 Text Clustering

We have heard about Google News clustering. In this exercise, we are going to implement it with Python.

3.1 Data Preprocessing

Let's switch up and use another dataset called 20newsgroup data, which is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The data is collected from a university's mailing list, where students exchange opinions in everything from motorcycles to middle east politics.

1. Import data using `sklearn.datasets.fetch_20newsgroups`
2. Transform data to vector with `TfidfVectorizer`

```
In [13]: from sklearn.datasets import fetch_20newsgroups
```

```
SetTrain = fetch_20newsgroups(subset='train')
TFidVectorizer = TfidfVectorizer(max_df=0.6, min_df=0.03)
Data = TFidVectorizer.fit_transform(SetTrain.data)
```

```
In [14]: #download dataset from sklearn.datasets.fetch_20newsgroups
#ใช้ TfidfVectorizer จากการอธิบายในข้อ 2.3 และเปลี่ยนมันเป็น vector โดยใช้ .fit_transform
```

3.2 Clustering

We are going to use the simplest clustering model, k-means clustering, to do this task. Our hope is that this simple algorithm will result in meaningful news categories, without using labels.

1. Fit K-Means clustering model to the text vector. What is the value of K you should pick? Why?
2. Use Silhouette score to evaluate your clusters. Try to evaluate the model for different values of k to see which k fits best for the dataset.

```
In [15]: from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import numpy as np
```

```
In [16]: #จาก การเลือกค่า K เราควรจะเลือก ที่20 โดยดูจากData setที่ให้มา มี 20 category ดังนั้น ค่า
kเงื่อนดั้นควรเป็นที่20
```

```
In [19]: kmeans = KMeans(n_clusters=20).fit(Data)
```

```
In [20]: Value_score= silhouette_score(Data,kmeans.labels_)
Value_score
```

```
Out[20]: 0.014781858389155434
```

```
In [21]: k_Data = [int(i) for i in range(2,38)]
max_df_Data = np.arange(0.5,1.0,0.1)
min_df_Data = np.arange(0.01,0.1,0.01)
round_loop = 0
print ("K DATA = ",k_Data)
print ("MAXDF DATA = ",max_df_Data)
print ("MINDF DATA = ",min_df_Data)
```

```
K DATA =  [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37]
MAXDF DATA =  [0.5 0.6 0.7 0.8 0.9]
MINDF DATA =  [0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09]
```

```
In [27]: Maxscore = [0,0,0,0]
round_loop =0
for Df_max in max_df_Data:
    for Df_min in min_df_Data:
        for K_value in k_Data:
            #
            #TFidVectorizer = TfidfVectorizer(max_df=Df_max, min_df=Df_min)
            #kmeans = KMeans(n_clusters=K_value).fit(Data)
            #score = silhouette_score(Data,kmeans.labels_)
            #print (score)
            #if Maxscore[0] < score:
            #    Maxscore[0] = score
            #    Maxscore[1] =Df_max
            #    Maxscore[2] =Df_min
            #    Maxscore[3] =K_value
            #    print (Maxscore)
            #    print (Maxscore)
            round_loop += 1
print (round_loop)
```

1620

```
In [28]: #K Data คือค่าใน n_clusters ,MAXDF DATAและ MINDF DATA คือค่าที่อยู่ใน TfidfVectorizer
#ในการตัดคำออกต่างๆ
# จากการลูปทั้งหมด 1620 รอบ อาจจะ ошибค่าจริงๆไม่ได้ เพียง เพราะ มันคงจะมี output เยอะมากจนคือ
# คล้ายเลือกจากที่คิดว่าน่าจะเป็นไปได้
```

```
In [35]: round_loop = 0
for K_value in k_Data:
    TFidVectorizer = TfidfVectorizer(max_df=0.6, min_df=0.03)
    Data = TFidVectorizer.fit_transform(SetTrain.data)
    kmeans = KMeans(n_clusters=K_value).fit(Data)
    score = silhouette_score(Data,kmeans.labels_)
    print (score)
    if Maxscore[0] < score:
        Maxscore[0] = score
        Maxscore[1] = 0.6
        Maxscore[2] = 0.3
        Maxscore[3] = K_value
        print (Maxscore)
    print (" K = ",K_value)
    print ("BEST SCORE  [Value ,Maxdf,Mindf ,K]" ,Maxscore)
    round_loop += 1
print (round_loop)
```

```
0.013330683853750608
K = 2
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.010842512096460402
K = 3
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.0027604108517145553
K = 4
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.004262302721368848
K = 5
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.00576620249091172
K = 6
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.0012661166798503527
K = 7
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.008901928086184771
K = 8
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.007311682133817594
K = 9
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.007865702185601564
K = 10
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.004153211055832921
K = 11
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.009553389392312785
K = 12
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.009748716955176776
K = 13
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.01102962682578062
K = 14
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.012441859047931027
K = 15
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.011393510632826587
K = 16
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.01002632756717323
K = 17
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.0126662402027510311
K = 18
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013341151453348974, 0.6, 0.3, 2]
0.013697266579315666
[0.013697266579315666, 0.6, 0.3, 19]
K = 19
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.013697266579315666, 0.6, 0.3, 19]
0.01506498841518331
[0.01506498841518331, 0.6, 0.3, 20]
```

```
K = 20
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.01506498841518331, 0.6, 0.3, 20]
0.014018745071716908
K = 21
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.01506498841518331, 0.6, 0.3, 20]
0.013048793578776953
K = 22
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.01506498841518331, 0.6, 0.3, 20]
0.014825059633922682
K = 23
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.01506498841518331, 0.6, 0.3, 20]
0.016621490864018484
[0.016621490864018484, 0.6, 0.3, 24]
K = 24
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.016621490864018484, 0.6, 0.3, 24]
0.017818939633817466
[0.017818939633817466, 0.6, 0.3, 25]
K = 25
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.017818939633817466, 0.6, 0.3, 25]
0.016828680565491756
K = 26
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.017818939633817466, 0.6, 0.3, 25]
0.01707398758888227
K = 27
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.017818939633817466, 0.6, 0.3, 25]
0.0196453786431104
[0.0196453786431104, 0.6, 0.3, 28]
K = 28
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.0196453786431104, 0.6, 0.3, 28]
0.017855754389593765
K = 29
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.0196453786431104, 0.6, 0.3, 28]
0.018772045326412506
K = 30
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.0196453786431104, 0.6, 0.3, 28]
0.018850252515179654
K = 31
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.0196453786431104, 0.6, 0.3, 28]
0.02103178845252741
[0.02103178845252741, 0.6, 0.3, 32]
K = 32
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.02103178845252741, 0.6, 0.3, 32]
0.01910364248139966
K = 33
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.02103178845252741, 0.6, 0.3, 32]
0.02118464123938383
[0.02118464123938383, 0.6, 0.3, 34]
K = 34
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.02118464123938383, 0.6, 0.3, 34]
0.022683834554311805
[0.022683834554311805, 0.6, 0.3, 35]
K = 35
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.022683834554311805, 0.6, 0.3, 35]
0.020648792749358304
K = 36
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.022683834554311805, 0.6, 0.3, 35]
0.017589984219730338
```

```
K = 37
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.022683834554311805, 0.6, 0.3, 35]
36
```

In [38]: `k_Data_2 = [int(i) for i in range(37,41)]`

In [39]: `for K_value in k_Data_2:
 TFidVectorizer = TfidfVectorizer(max_df=0.6, min_df=0.03)
 Data = TFidVectorizer.fit_transform(SetTrain.data)
 kmeans = KMeans(n_clusters=K_value).fit(Data)
 score = silhouette_score(Data,kmeans.labels_)
 print (score)
 if Maxscore[0] < score:
 Maxscore[0] = score
 Maxscore[1] = 0.6
 Maxscore[2] = 0.3
 Maxscore[3] = K_value
 print (Maxscore)
 print (" K = ",K_value)
 print ("BEST SCORE [Value ,Maxdf,Mindf ,K]" ,Maxscore)
 round_loop += 1
print (round_loop)`

```
0.01999320490146477
K = 37
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.022683834554311805, 0.6, 0.3, 35]
0.02226172687844214
K = 38
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.022683834554311805, 0.6, 0.3, 35]
0.01968501412113009
K = 39
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.022683834554311805, 0.6, 0.3, 35]
0.017715456285546622
K = 40
BEST SCORE [Value ,Maxdf,Mindf ,K] [0.022683834554311805, 0.6, 0.3, 35]
40
```

In [41]: `TFidVectorizer = TfidfVectorizer(max_df=0.6, min_df=0.03)
Data = TFidVectorizer.fit_transform(SetTrain.data)
kmeans = KMeans(n_clusters=Maxscore[3]).fit(Data)
score = silhouette_score(Data,kmeans.labels_)

print (" BEST K in this Data set is ",Maxscore[3])
print ("Best Silhouette score is ",score)`

```
BEST K in this Data set is 35
Best Silhouette score is 0.02122747632251899
```

3.3 Topic Terms

We want to explore each cluster to understand what news articles are in the cluster, what terms are associated with the cluster. This will require a bit of hacking.

1. Use TfidfVectorizer.get_feature_names to extract words associated with each dimension of the text vector.
2. Extract cluster's centroids using kmeans.cluster_centers_ .
3. For each centroid, print the top 15 words that have the highest frequency.

```
In [53]: Allname_list = TFidVectorizer.get_feature_names()  
#use get_feature_names to get feature name to extract word we will get array for  
#use only index to find word
```

```
In [54]: Centroid = kmeans.cluster_centers_  
Centroid  
  
#Extract cluster's centroids with kmean cluseter centers  
  
#cluster_centers_ : array, [n_clusters, n_features]  
#Coordinates of cluster centers
```

```
Out[54]: array([[0.03908232, 0.01043577, 0.0393467 , ..., 0.00312986, 0.03230858,  
    0.01605957],  
    [0.00449062, 0.00483878, 0.01453639, ..., 0.00490331, 0.03234945,  
    0.01441724],  
    [0.00797177, 0.00520058, 0.01690675, ..., 0.00237157, 0.04876936,  
    0.0215497 ],  
    ...,  
    [0.00226075, 0.00426246, 0.01007843, ..., 0.00521206, 0.05057354,  
    0.02125985],  
    [0.00379712, 0.00106012, 0.00653402, ..., 0.00218854, 0.05215283,  
    0.02045172],  
    [0.00328108, 0.00524984, 0.01946668, ..., 0.00342005, 0.03418815,  
    0.01828043]])
```

```
In [56]: ListName = []
for RoundCount,Data in enumerate(order_centroids):
    Top = Data.argsort()[-15:][::-1]
    print ("Centroid ",RoundCount)
    for index in Top:
        ListName.append(name_all[index])
    print (" -----HIHGEST FREQUENCY WORD -----")
    print (ListName)
    print (" -----")
    ListName =[]
#enumerate is we can for i in data and use another variable to count round
#Listname is List word 15 words Highest frequency
#use argsort to sort valiable to change to index example [0.5,0.0,0.7]
# argsort in convert to [0.0,0.5,0.7] and then get index --> 0.0 =0  0.5=1
# 0.7 =2
# argsort [0.5, 0.0 ,0.7] --> [1,0,2]
# and we use[-15] [::-1] to inverst array and get only 15 Last data it mean
# Highest Frequency
```

```
Centroid 0
-----HIHGEST FREQUENCY WORD -----
['sale', 'offer', 'new', 'distribution', 'price', 'or', 'university', 'bes
t', 'email', 'com', 'usa', '10', 'with', 'sell', '00']
-----
Centroid 1
-----HIHGEST FREQUENCY WORD -----
['university', 'posting', 'host', 'nntp', 'thanks', 'any', 'have', 'de', 'o
r', 'you', 'with', 'be', 'me', 'mail', 'if']
-----
Centroid 2
-----HIHGEST FREQUENCY WORD -----
['hp', 'com', 'you', 'have', 'tin', 'with', 'newsreader', 'are', 'version',
'host', 'nntp', 'posting', 'my', 'article', 'or']
-----
Centroid 3
-----HIHGEST FREQUENCY WORD -----
['steve', 'com', 'you', 'not', 'writes', 'was', 'article', 'are', 'can', 't
hey', 'if', 'as', 'ca', 'university', 'be']
-----
Centroid 4
-----HIHGEST FREQUENCY WORD -----
['god', 'jesus', 'not', 'you', 'we', 'are', 'he', 'be', 'as', 'his', 'but',
'have', 'who', 'with', 'do']
-----
Centroid 5
-----HIHGEST FREQUENCY WORD -----
['car', 'you', 'my', 'com', 'have', 'with', 'was', 'but', 'be', 'or', 'the
y', 'are', 'if', 'out', 'me']
-----
Centroid 6
-----HIHGEST FREQUENCY WORD -----
['org', 'you', 'writes', 'article', 'be', 'not', 'com', 'as', 'will', 'if',
'are', 'have', 'with', 'or', 'can']
-----
Centroid 7
-----HIHGEST FREQUENCY WORD -----
['window', 'windows', 'you', 'problem', 'com', 'with', 'my', 'when', 'be',
'if', 'can', 'or', 'not', 'an', 'have']
-----
Centroid 8
-----HIHGEST FREQUENCY WORD -----
['mit', 'internet', 'com', 'you', 'host', 'nntp', 'posting', 'with', 'my',
'be', 'are', 'technology', 'an', 'have', 'if']
-----
Centroid 9
-----HIHGEST FREQUENCY WORD -----
['card', 'video', 'with', 'windows', 'have', 'an', 'can', 'you', 'at', 'doe
s', 'university', 'my', 'anyone', 'or', 'graphics']
-----
Centroid 10
-----HIHGEST FREQUENCY WORD -----
['netcom', 'com', 'services', 'you', 'be', 'writes', 'david', 'have', 'no
t', 'article', 'are', 'with', 'line', 'or', 'if']
-----
Centroid 11
-----HIHGEST FREQUENCY WORD -----
```

```
['be', 'you', 'with', 'have', 'are', 'my', 'or', 'as', 'not', 'but', 'if',
'can', 'at', 'one', 'they']
-----
Centroid 12
-----HIGHGEST FREQUENCY WORD -----
['ca', 'canada', 'university', 'you', 'article', 'have', 'writes', 'was',
'my', 'posting', 'can', 'host', 'nntp', 'are', 'with']
-----
Centroid 13
-----HIGHGEST FREQUENCY WORD -----
['ibm', 'com', 'you', 'not', 'pc', 'are', 'or', 'article', 'disclaimer', 'h
ave', 'writes', 'be', 'posting', 'can', 'if']
-----
Centroid 14
-----HIGHGEST FREQUENCY WORD -----
['cs', 'science', 'computer', 'dept', 'article', 'you', 'writes', 'universi
ty', 'with', 'be', 'univ', 'are', 'can', 'but', 'have']
-----
Centroid 15
-----HIGHGEST FREQUENCY WORD -----
['uk', 'ac', 'co', 'you', 'with', 'are', 'university', 'be', 'as', 'or', 'c
an', 'have', 'not', 'if', 'writes']
-----
Centroid 16
-----HIGHGEST FREQUENCY WORD -----
['was', 'were', 'they', 'by', 'as', 'not', 'had', 'their', 'with', 'have',
'who', 'you', 'are', 'no', 'people']
-----
Centroid 17
-----HIGHGEST FREQUENCY WORD -----
['drive', 'hard', 'disk', 'with', 'my', 'have', 'you', 'or', 'can', 'if',
'mac', 'system', 'not', 'computer', 'be']
-----
Centroid 18
-----HIGHGEST FREQUENCY WORD -----
['washington', 'university', 'host', 'nntp', 'posting', 'you', 'article',
'if', 'distribution', 'writes', 'with', 'have', 'can', 'be', 'me']
-----
Centroid 19
-----HIGHGEST FREQUENCY WORD -----
['nasa', 'gov', 'space', 'article', 'writes', 'was', 'you', 'center', 'rese
arch', 'not', 'with', 'host', 'com', 'have', 'at']
-----
Centroid 20
-----HIGHGEST FREQUENCY WORD -----
['cc', 'university', 'you', 'posting', 'nntp', 'host', 'have', 'are', 'wit
h', 'au', 'article', 'at', 'not', 'writes', 'or']
-----
Centroid 21
-----HIGHGEST FREQUENCY WORD -----
['state', 'university', 'posting', 'you', 'host', 'nntp', 'article', 'new',
'have', 'are', 'or', 'my', 'be', 'writes', 'at']
-----
Centroid 22
-----HIGHGEST FREQUENCY WORD -----
['gun', 'you', 'are', 'have', 'as', 'be', 'not', 'control', 'they', 'woul
d', 'by', 'if', 'com', 'my', 'with']
```

```
-----  
Centroid 23  
-----HIHGEST FREQUENCY WORD -----  
['space', 'nasa', 'be', 'by', 'as', 'earth', 'not', 'at', 'was', 'with', 'will', 'you', 'would', 'university', 'have']  
-----  
Centroid 24  
-----HIHGEST FREQUENCY WORD -----  
['00', '10', '34', '11', '14', '12', '15', '25', '18', '17', '20', '30', '24', '19', '35']  
-----  
Centroid 25  
-----HIHGEST FREQUENCY WORD -----  
['he', 'his', 'was', 'him', 'be', 'as', 'but', 'not', 'you', 'with', 'have', 'at', 'year', 'has', 'who']  
-----  
Centroid 26  
-----HIHGEST FREQUENCY WORD -----  
['key', 'chip', 'be', 'will', 'they', 'can', 'as', 'public', 'system', 'are', 'with', 'you', 'bit', 'if', 'or']  
-----  
Centroid 27  
-----HIHGEST FREQUENCY WORD -----  
['andrew', 'pittsburgh', 'you', 'host', 'nntp', 'posting', 'com', 'are', 'or', 'reply', 'be', 'what', 'engineering', 'have', 'your']  
-----  
Centroid 28  
-----HIHGEST FREQUENCY WORD -----  
['you', 'your', 'are', 'if', 'not', 'have', 'be', 'do', 'com', 'what', 'as', 'or', 'can', 'with', 'they']  
-----  
Centroid 29  
-----HIHGEST FREQUENCY WORD -----  
['are', 'we', 'not', 'be', 'as', 'you', 'they', 'people', 'have', 'or', 'if', 'with', 'by', 'there', 'but']  
-----  
Centroid 30  
-----HIHGEST FREQUENCY WORD -----  
['she', 'her', 'was', 'my', 'you', 'be', 'not', 'with', 'com', 'has', 'or', 'so', 'as', 'at', 'what']  
-----  
Centroid 31  
-----HIHGEST FREQUENCY WORD -----  
['team', 'game', 'they', 'games', 'was', 'will', 'win', 'but', 'he', 'have', 'you', 'are', 'ca', 'be', 'year']  
-----  
Centroid 32  
-----HIHGEST FREQUENCY WORD -----  
['com', 'inc', 'you', 'article', 'writes', 'posting', 'sun', 'nntp', 'host', 'my', 'have', 'or', 'are', 'not', 'with']  
-----  
Centroid 33  
-----HIHGEST FREQUENCY WORD -----  
['windows', 'dos', 'files', 'file', 'have', 'with', 'you', 'can', 'program', 'use', 'or', 'com', 'my', 'any', 'run']  
-----  
Centroid 34
```

```
-----HIHGEST FREQUENCY WORD -----  
['access', 'net', 'communications', 'com', 'usa', 'nntp', 'posting', 'hos  
t', 'be', 'steve', 'unix', 'you', 'or', 'are', 'distribution']  
-----
```