**Assignment 1**

**Question 1**

1a. Please refer to assign1.py

1b. Please refer to assign1.py

1c. Using f1-scores to measure the accuracy of the trained Naïve Bayes binary classifier for each label returned the following:

|  |  |
| --- | --- |
| **Label** | **CS5304NBClassifier F1-score** |
| 4 | 0.6422 |
| 5 | 0.6103 |
| 7 | 0.6237 |
| 33 | 0.8040 |
| 59 | 0.6568 |
| 70 | 0.8198 |
| 83 | 0.4870 |
| 95 | 0.7426 |
| 98 | 0.8800 |
| 102 | 0.8296 |

The Naïve Bayes classifier shows label 83 to be the most difficult to classify.

Looking at the f1-scores for the trained K-Nearest Neighbor method returned:

|  |  |  |
| --- | --- | --- |
| **Label** | **CS5304KNNClassifier F1-score** | **k** |
| 4 | 0.7858 | 17 |
| 5 | 0.7992 | 87 |
| 7 | 0.3525 | 1 |
| 33 | 0.8290 | 19 |
| 59 | 0.5275 | 7 |
| 70 | 0.8273 | 19 |
| 83 | 0.5923 | 5 |
| 95 | 0.7262 | 21 |
| 98 | 0.8004 | 7 |
| 102 | 0.8374 | 25 |

Where the ‘k’ column is the value of k that returned the highest f1-score in the range [1, 109]. The K-Nearest Neighbor method returns label 7 to be the most difficult to classify. However, note that this is achieved when k=1, that is, when the model has been overfitted. This suggests that the validation data is very similar to the training data since simply looking at the closest neighbor to the validation data maximizes accuracy. Therefore, despite the low f1-score of label 7, I still believe that label 83 is the most difficult to classify. We can see this by increasing the number of observations in the training data to 5000:

|  |  |  |
| --- | --- | --- |
| **Label** | **CS5304KNNClassifier F1-score** | **k** |
| 4 | 0.8330 | 93 |
| 5 | 0.8242 | 53 |
| 7 | 0.5928 | 3 |
| 33 | 0.8680 | 53 |
| 59 | 0.6822 | 31 |
| 70 | 0.8323 | 11 |
| 83 | 0.5959 | 15 |
| 95 | 0.7683 | 35 |
| 98 | 0.9034 | 17 |
| 102 | 0.8900 | 79 |

Note that the f1-score for label 7 has increased while it has remained similar for label 83. Increasing the number of observations in the training data to 7500:

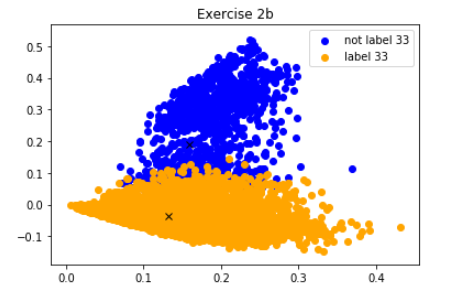
|  |  |  |
| --- | --- | --- |
| **Label** | **CS5304KNNClassifier F1-score** | **k** |
| 4 | 0.8369 | 99 |
| 5 | 0.8281 | 81 |
| 7 | 0.6086 | 5 |
| 33 | 0.8706 | 81 |
| 59 | 0.6891 | 9 |
| 70 | 0.8395 | 5 |
| 83 | 0.5894 | 45 |
| 95 | 0.7685 | 105 |
| 98 | 0.9056 | 27 |
| 102 | 0.8885 | 69 |

Again we see that the f1-score of label 7 has improved from the case of 5000 observations in the training data, but the f1-score of label 83 has not improved despite the increase in training data observations.

**Question 2**

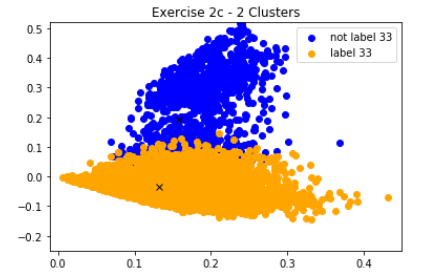
2a. Please refer to assign1.py

2b.



2c. For the K-Means clustering algorithm for 2, 3 and 4 clusters and random centroid initialization we find the following:

**2 clusters**



**3 Clusters**



**4 Clusters**

****

We first note that setting random centroids performs identical to using the mean of the training data as the initial centroid. This is shown by the f1-scores of the two methods:

* Average of training data: 5513
* Random centroids: 0.5513

I believe setting the mean of the training data as the initial centroid gives the K-Means algorithm a “head-start” relative to starting at random locations, but the algorithm will converge to give the same optimal centroid.

Second, as the number of clusters increases, we see that while the cluster that is *not* label 33 (blue) does not change with increasing numbers of clusters, the cluster for label 33 gets further divided into 2 and 3 clusters for total 3 and 4 clusters respectively.

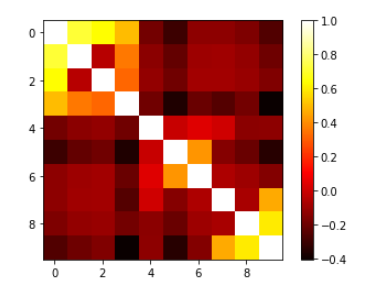
Looking at the 10 labels of interest we have:[[1]](#footnote-1)

* Label 4: C15
* Label 5: C151
* Label 7: C152
* Label 33: CCAT [C]
* Label: 59: ECAT
* Label 70: GCAT
* Label 83: GPOL
* Label 95: M13
* Label 98: M14
* Label 102: MCAT

Label 33 represents topic code CCAT (Corporate/Industrial), which is one of the four hierarchical groups: CCAT, ECAT, GCAT and MCAT. In the hierarchy of categories, we are told these are represented as C, E, G and M respectively. The code also increases in length as we go down the hierarchy, hence C311 is a child of C31, which in turn would be a child of C3 and so on. Furthermore, we are told that related codes have similar numbers, that is, C31 and C32 are related1.

Looking at the above, I suspect the K-Means clustering algorithm can effectively distinguish between label 33 (CCAT) and those not label 33 (as shown by the plot for 2b with 2 clusters). Doing so manually from the above labels leaves: [C15, C151, C152, CCAT] as those topics related to label 33 (CCAT) and the remainder: [ECAT, GCAT, GPOL, M13, M14, MCAT]. Further adding clusters attempts to further categorize topics within the [C15, C151, C152, CCAT] set. Topics C151 and C152 are related, and both are children of C15. I believe the plot for 2b with 3 clusters finds C15 *within* the label 33 cluster (CCAT). Increasing the number of clusters to 4 may have found C151 or C152 *within* C15.

The correlation between labels 4, 5, 7 and 33 are evident when we compute a 10-by-10 correlation matrix between the labels:



The first three columns (corresponding to labels 4, 5 and 7) show correlation to the fourth label (33) than to the remaining labels.

**Appendix: K-Nearest Neighbors**

# K-Nearest Neighbor:

from sklearn.metrics import f1\_score

from sklearn.neighbors import KNeighborsClassifier

kvec = np.arange(1,110,2)

k\_data = np.empty(shape = [3,len(labels)])

j = 0

limit = 1000

for n in range(0,len(labels)):

target = rcv\_y[:limit,labels[0][n]] # for each classification, target contains 1 or 0

target2 = target.toarray().ravel()

# validation data targets

valid\_target = validation\_target[:,labels[0][n]].toarray().ravel()

i = 0

f1sc = np.empty(shape = [1,len(kvec)])

for k in kvec:

knn = KNeighborsClassifier(n\_neighbors=k)

# 1. Train our model: for the measurements in rcv\_x, we have a true North of target2

knn.fit(rcv\_x[:limit,:], target2)

# 2. Validate model:

# fit target:

valid\_pred = knn.predict(validation\_data)

# satisfy check\_output

#print(type(valid\_pred))

#print(valid\_pred.ndim)

#print(valid\_pred.shape[0])

#print(i)

# compare fitted target to true target in validation\_target

f1sc[0,i] = f1\_score(valid\_target, valid\_pred, average = 'binary')

i = i+1

f1sc\_list = f1sc.tolist()

# start filling k\_data matrix:

k\_data[0,j] = labels[0][n]

k\_data[1,j] = kvec[f1sc\_list[0].index(max(f1sc\_list[0]))]

k\_data[2,j] = f1sc\_list[0][f1sc\_list[0].index(max(f1sc\_list[0]))]

print(j)

j = j+1

print(k\_data)

**Appendix: Naïve Bayes**

# Naive Bayes Bernoulli:

from sklearn.naive\_bayes import BernoulliNB

from sklearn.metrics import f1\_score

nb\_clf = BernoulliNB()

i = 0

nb\_f1sc = np.empty(shape = [1,len(labels)])

for n in range(0,len(labels)):

#for n in range(0,len(labels)):

target = rcv\_y[:,labels[0][n]] # for each classification, target contains 1/0

target2 = target.toarray().ravel()

# validation data targets

valid\_target = validation\_target[:,labels[0][n]].toarray().ravel()

# 1. Train our model

nb\_clf.fit(rcv\_x, target2)

# 2. Validate model:

# fit target:

valid\_pred = nb\_clf.predict(validation\_data)

# compare fitted target to true target in validation\_target

nb\_f1sc[0,i] = f1\_score(valid\_target, valid\_pred)

i = i + 1

print(nb\_f1sc)

print(labels)

**Appendix: K-Means Clustering**

# K-Means

from sklearn.cluster import KMeans

from scipy.sparse import find

for n in range(0,10):

bin\_clf = rcv\_y[:,n].toarray().ravel() # 23149 by 1

ones\_ind = (bin\_clf != 0)

zeros\_ind = (bin\_clf == 0)

centroid0 = np.mean(rcv\_x[zeros\_ind,:],axis=0)

centroid1 = np.mean(rcv\_x[ones\_ind,:],axis=0)

centroids\_array = np.array(np.vstack((centroid0, centroid1)))

# default behavior of KMeans is to initialize algorithm using different centroids.

# the number of times this happens is controlled by n\_init parameter.

# Since we are submitting an array in to the init argument, only a single initialization will be performed

# hence we set n\_init = 1

# train KMeans clustering:

kmeans = KMeans(n\_clusters=2, init = centroids\_array, n\_init=1).fit(rcv\_x)

# predict:

clster\_pred = kmeans.predict(validation\_data)

print(centroids\_array)

1. 1. http://www.jmlr.org/papers/volume5/lewis04a/lewis04a.pdf

   [↑](#footnote-ref-1)