HW_KNN

February 6, 2022

[2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from collections import defaultdict, Counter
import matplotlib.pyplot as plt
```

[155]:
```
train_path = 'newsgroups/trainMatrixModified.txt'
test_path = 'newsgroups/testMatrixModified.txt'
train_labels_path = 'newsgroups/trainClasses.txt'
test_labels_path = 'newsgroups/testClasses.txt'
words_path = 'newsgroups/modifiedterms.txt'
```

[156]:
```
# Load data in dataframe
train_data_df = pd.read_csv(train_path, delimiter='\t', header=None)
test_data_df = pd.read_csv(test_path, delimiter='\t', header=None)
```

[157]:
```
# Load data via numpy and Transpose
train_data = pd.read_table(train_path, delimiter='\t', header=None)
test_data = pd.read_table(test_path, delimiter='\t', header=None)
train_labels = pd.read_table(train_labels_path, delimiter='\t', header=None,␣
 ↪index_col=0)
test_labels = pd.read_table(test_labels_path, delimiter='\t', header=None,␣
 ↪index_col=0)
```

[6]:
```
print('There are 800 documents and 5500 unique words')
print('Training Data Details')
print('shape: ',train_data.shape)
print('dimensions: ', train_data.ndim)
print('num of unique values:', len(np.unique(train_data)))
print('Test Data Details')
print(test_data.shape)
```

```
There are 800 documents and 5500 unique words
Training Data Details
shape:  (5500, 800)
dimensions:  2
num of unique values: 69
Test Data Details
```

```
(5500, 200)
```

Because of this we want to transpoe the data so that rows are documents and columns are the words.

[7]:
```
train_data = train_data.transpose()
test_data = test_data.transpose()
```

[152]:
```
train_labels = np.array(train_labels).flatten()
test_labels = np.array(train_labels).flatten()
```

[9]:
```
words = np.loadtxt(words_path,dtype='object', delimiter='\t')
print(train_labels.shape)
print(train_labels.ndim)
```

```
(800,)
1
```

1 Part a

[10]:
```
target = train_data[0]
print(target)
print(len(target))
```

```
0      2.0
1      0.0
2      0.0
3      2.0
4      2.0
      ...
795    0.0
796    0.0
797    0.0
798    0.0
799    0.0
Name: 0, Length: 800, dtype: float64
800
```

[11]:
```
def knn_class(target, data, labels, k, dist=None):
    neighbors_index, distances = knn_neighbors(target, data, labels, k, dist)
    neighbors_labels = labels[neighbors_index]
    count = Counter(neighbors_labels)
    predicted_label = count.most_common(1)[0][0]
    return neighbors_index, predicted_label

def knn_neighbors(target, data, labels, k, dist=None):
    '''returns the index of the k nearest neighbors and a distance matrix
    of all distances'''
```

```
    if dist:
        data_m = np.array([np.linalg.norm(data[i]) for i in range(len(data))])
        target_m = np.linalg.norm(target)
        cosine = np.dot(data, target) / (data_m * target_m)
        distance = 1 - cosine
    else:
        distance = np.sqrt(((data-target)**2).sum(axis=1))
    index = np.argsort(distance)
    return index[:k], distance
```

[12]:
```
dist, label = knn_class(target, train_data, train_labels, 5)
print(label)
```

```
0
```

2 B

[13]:
```
def knn_evaluation(train_data, train_labels, test_data, test_labels, k,␣
↪dist=None):
    errors = 0
    length = test_data.shape[0]
    for i in range(length):
        target = test_data.loc[i, :]
        _, target_class = knn_class(target, train_data, train_labels, k, dist)
        if target_class != test_labels[i]:
            errors += 1
    return round(errors / length, 2)
```

[14]:
```
%%time
knn_evaluation(train_data, train_labels, test_data, test_labels, 4)
```

```
CPU times: user 4.65 s, sys: 1.28 s, total: 5.93 s
Wall time: 5.9 s
```

[14]:
```
0.53
```

3 C

[22]:
```
ks = [x for x in range(0, 101, 5)][1:]
```

[23]:
```
euclid = []
```

[157]:
```
cosine = []
```

[25]:
```
%%time
for k in ks:
    euclid.append(knn_evaluation(train_data, train_labels,
```

```
                                    test_data, test_labels,
                                    k))
    print(f'{k}/{max(ks)}', end=' | ')
```

```
5/100 | 10/100 | 15/100 | 20/100 | 25/100 | 30/100 | 35/100 | 40/100 | 45/100 |
50/100 | 55/100 | 60/100 | 65/100 | 70/100 | 75/100 | 80/100 | 85/100 | 90/100 |
95/100 | 100/100 | CPU times: user 1min 27s, sys: 25.1 s, total: 1min 53s
Wall time: 1min 53s
```

[158]:
```
%%time
for k in ks:
    cosine.append(knn_evaluation(train_data, train_labels,
                                    test_data, test_labels,
                                    k, dist=True))
    print(f'{k}/{max(ks)}', end=' | ')
```

```
5/100 | 10/100 | 15/100 | 20/100 | 25/100 | 30/100 | 35/100 | 40/100 | 45/100 |
50/100 | 55/100 | 60/100 | 65/100 | 70/100 | 75/100 | 80/100 | 85/100 | 90/100 |
95/100 | 100/100 | CPU times: user 2min 2s, sys: 4min 24s, total: 6min 27s
Wall time: 49.8 s
```

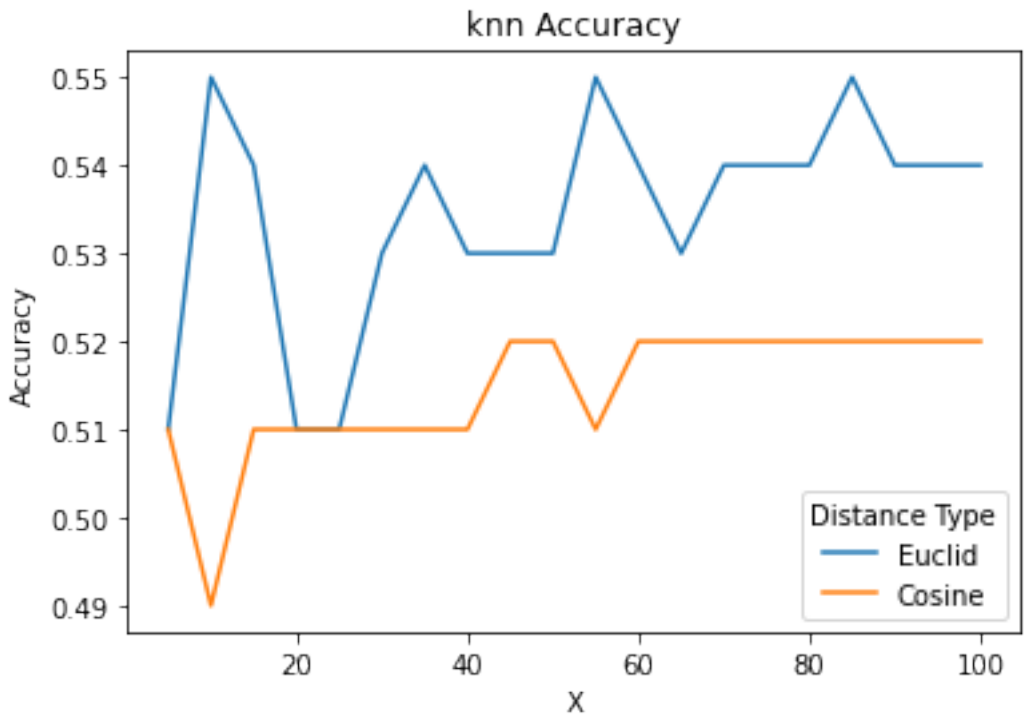
[30]:
```
print(len(euclid))
print(len(cosine))
with open('euclid', 'w') as the_file:
    for i in euclid:
        the_file.write(f'{i},')
```

```
20
20
```

[31]:
```
with open('cosine', 'w') as the_file:
    for i in cosine:
        the_file.write(f'{i},')
```

[159]:
```
plot_data = pd.DataFrame({'X':ks, 'Euclid':euclid, 'Cosine':cosine})
plot_data = pd.melt(plot_data, id_vars=['X'], value_vars=['Euclid', 'Cosine'],
                    var_name='Distance Type', value_name='Accuracy')
sns.lineplot(data=plot_data, x='X', y='Accuracy', hue='Distance Type').
 ↪set(title='knn Accuracy')
```

[159]:
```
[Text(0.5, 1.0, 'knn Accuracy')]
```



4 D

[15]:
```
# TD = data
# DF = doc frequency
# NMatrix = shape_matrix
# IDF = idf
def tfidf_transform(data):
    word_freq = data.sum(axis=1)
    num_of_docs, num_of_words = data.shape
    doc_freq = pd.DataFrame([(data != 0).sum(1)]).T
    shape_matrix = np.ones(np.shape(data), dtype=float) * num_of_docs
    idf = np.log2(np.divide(shape_matrix, np.array(doc_freq)))
    td_tfidf = data * idf
    return td_tfidf
```

[139]:
```
idf_train_data = tfidf_transform(train_data)
idf_test_data = tfidf_transform(test_data)
```

[162]:
```
idf_cosine = []
```

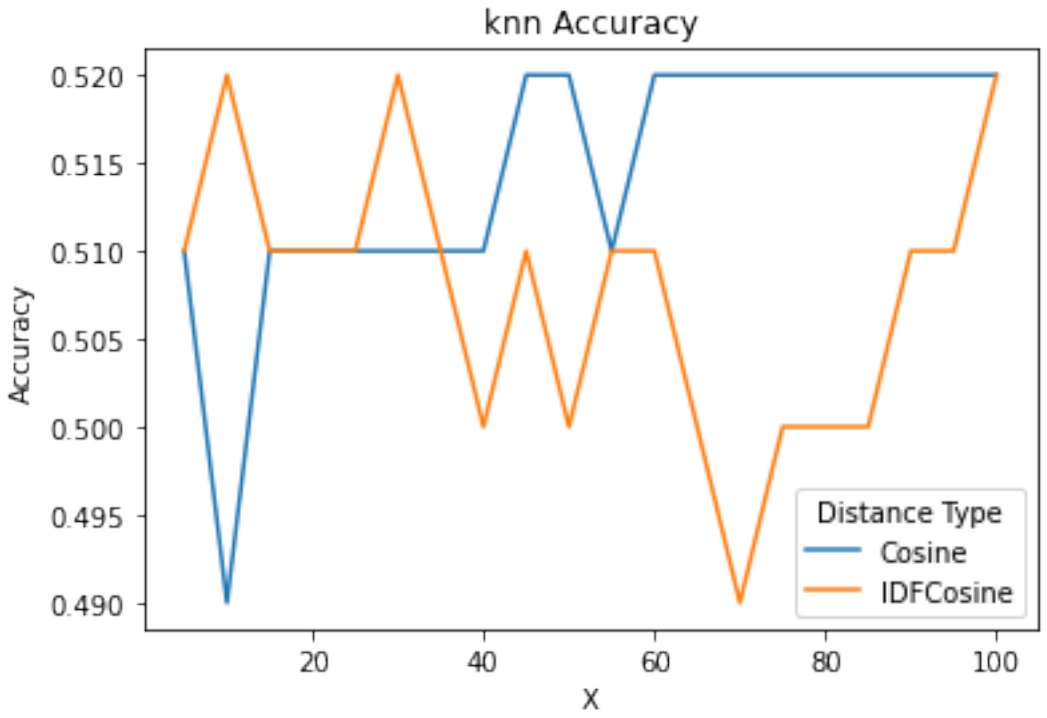
[163]:
```
%%time
for k in ks:
    idf_cosine.append(knn_evaluation(idf_train_data, train_labels,
```

```
                                    idf_test_data, test_labels,
                                    k, dist=True))
    print(f'{k}/{max(ks)}', end=' | ')
```

```
5/100 | 10/100 | 15/100 | 20/100 | 25/100 | 30/100 | 35/100 | 40/100 | 45/100 |
50/100 | 55/100 | 60/100 | 65/100 | 70/100 | 75/100 | 80/100 | 85/100 | 90/100 |
95/100 | 100/100 | CPU times: user 2min 4s, sys: 4min 33s, total: 6min 38s
Wall time: 51.6 s
```

[164]:
```
plot_data = pd.DataFrame({'X':ks, 'Cosine':cosine, 'IDFCosine':idf_cosine})
plot_data = pd.melt(plot_data, id_vars=['X'], value_vars=['Cosine',␣
 ↪'IDFCosine'],
                    var_name='Distance Type', value_name='Accuracy')
sns.lineplot(data=plot_data, x='X', y='Accuracy', hue='Distance Type').
 ↪set(title='knn Accuracy')
```

[164]:
```
[Text(0.5, 1.0, 'knn Accuracy')]
```



5 E

[71]:
```
def rocchio_training(training_data, training_labels):
    data_df = pd.DataFrame(training_data)
    data_df['Label'] = training_labels
    labels = data_df['Label'].unique()
    centroids = defaultdict()
    for label in labels:
        # Create df of just class c
        D_c = data_df[data_df['Label']==label]
        # exclude the lable column
        D_c = D_c.drop(['Label'], axis=1)
        length = D_c.shape[0]
        centroid_c = np.array((1/length)) * D_c.sum(axis=1)
        centroids[str(label)] = centroid_c
    return centroids
```

[73]:
```
(rocchio_training(idf_train_data, train_labels))
```

[73]:
```
defaultdict(None,
            {'0': 0      1.480580
             2      0.536402
             4      0.406815
             11     0.767049
             12     1.931802
                      …
             784    1.111546
             785    0.401638
             790    1.205701
             793    0.331791
             795    0.391087
             Length: 401, dtype: float64,
             '1': 1      0.735179
             3      0.441274
             5      1.420226
             6      0.921360
             7      0.576800
                      …
             794    0.726264
             796    0.495681
             797    0.757462
             798    0.043328
             799    0.836352
             Length: 399, dtype: float64})
```

[103]:
```
def classify_rocchio(target, training_data, training_labels):
    centroids = rocchio_training(training_data, training_labels)
```

```
    dis_to_centroid = defaultdict()
    for c, array in centroids.items():
        distance = 1 - (target - array)
        dis_to_centroid[c] = distance

    return max(dis_to_centroid)
```

[106]:
```
classify_rocchio(idf_train_data.loc[0], idf_train_data, train_labels)
```

[106]:
```
'1'
```

6 F

[149]:
```
from sklearn.neighbors import NearestCentroid
from sklearn.metrics import confusion_matrix as cm
```

[140]:
```
idf_train_data.shape
```

[140]:
```
(800, 5500)
```

[141]:
```
centroid = NearestCentroid()
centroid.fit(idf_train_data, train_labels)
```

[141]:
```
NearestCentroid()
```

[143]:
```
y_predicted = pd.DataFrame(centroid.predict(idf_test_data))
```

[161]:
```
print('Confusion Matrix for Nearest Centroid')
cm(test_labels, y_predicted)
```

```
Confusion Matrix for Nearest Centroid
```

[161]:
```
array([[95,  4],
       [45, 56]])
```

[ ]:
```

```