CSE151A_PA1

March 18, 2021

[1]:
```python
import numpy as np
import pandas as pd
```

[2]:
```python
train = np.loadtxt('data/pa1train.txt')
test = np.loadtxt('data/pa1test.txt')
valid = np.loadtxt('data/pa1validate.txt')
```

[3]:
```python
Xtrain = train[:,:784]
ytrain = train[:,-1]

Xtest = test[:,:784]
ytest = test[:,-1]

Xvalid = valid[:,:784]
yvalid = valid[:,-1]
```

1. For k = 1, 5, 9 and 15, build k-nearest neighbor classifiers from the training data. For each of these values of k, write down a table of training errors (error on the training data) and the validation errors (error on the validation data). Which of these classifiers performs the best on validation data? What is the test error of this classifier? [Hint: As a check for your code, the training error for k = 3 should be about 0.04.]

[4]:
```python
def knn(k, x, y):
    pred = [0]*len(y)
    for i, xi in enumerate(x):
        #compute euclidean distance
        distances = np.linalg.norm(Xtrain - xi, ord =2, axis =1)
        #gets indices of sorted distances
        label = np.argsort(distances, axis = 0)
        #gets labels of k smallest distances
        label_pred = ytrain[label[:k]].ravel().astype(np.int)
        #retrieves most common value found in labels
          #bincount returns the count of each value(label)
          #argmax returns the index of the first maximum value(count)
        pred[i] = np.argmax(np.bincount(label_pred))
    return pred
```

[5]:
```
#model pipeline
K = [1,5,9,15]
knn_train_err = []
knn_valid_err = []
for k in K:
    #knn on training data
    train_correct = knn(k, Xtrain,ytrain) == ytrain
    train_error = 1 - (sum(train_correct) / len(train_correct))
    knn_train_err.append(train_error)
    #knn on validation data
    valid_correct = knn(k, Xvalid,yvalid) == yvalid
    valid_error = 1 - (sum(valid_correct) / len(valid_correct))
    knn_valid_err.append(valid_error)
```

[6]:
```
pd.DataFrame({'k': K, 'training error':knn_train_err, 'validation error':␣
 ↪knn_valid_err})
```

[6]:
```
    k  training error  validation error
0   1          0.0000             0.082
1   5          0.0565             0.095
2   9          0.0685             0.104
3  15          0.0925             0.108
```

[7]:
```
k3_train_correct = knn(3, Xtrain,ytrain) == ytrain
k3_train_error = 1 - (sum(k3_train_correct) / len(k3_train_correct))
print('k = 3 training error: ', k3_train_error)
```

```
k = 3 training error:  0.04349999999999998
```

0.0.1 k =1 gives the lowest training and validation error / (1NN) performs best on the validation data

[8]:
```
K = [1]
knn_test_err = []
for k in K:
    #knn on test data
    test_correct = knn(k, Xtest,ytest) == ytest
    test_error = 1 - (sum(test_correct) / len(test_correct))
    knn_test_err.append(test_error)
```

[9]:
```
pd.DataFrame({'k': K, 'test error':knn_test_err})
```

[9]:
```
   k  test error
0  1       0.094
```

0.0.2 observe the test error above, the test error of the 1NN classifier also performs the best, followed by the 5NN classifier

2. In the first few lectures, we talked about preprocessing data with projections. In this part of the assignment, we will look at how using a projection as a pre-processing step affects the accuracy and running-time of nearest neighbor classification. Download the file projection.txt from the class website. This file represents a projection matrix P with 784 rows and 20 columns. Each column is a 784-dimensional unit vector, and the columns are orthogonal to each other.

Project the training, validation and test data onto the column space of this matrix, and repeat part (1) of the problem. For k = 1, 5, 9, 15 write down a table of the training and validation errors, as well as the test error of the classifier which performs best on the validation data. [Hint: As a check for your code, the training error for k = 3 after projection should be about 0.16.] How is the classification accuracy affected by projection? How does the running time of your program change when you run it on projected data?

[10]:
```
proj_matrix = np.loadtxt('data/projection.txt')
```

[11]:
```
#Xtrain shape = 2000 x 784   784 x 20 proj_matrix.shape
```

[12]:
```
#X_train_proj = np.matmul(Xtrain.dot(proj_matrix),proj_matrix.T) / (np.linalg.
 ↪norm(proj_matrix)**2)
```

[13]:
```
# def knn_2(k, x, y):
#     pred = [0]*len(y)
#     X_proj_Tr = np.matmul(Xtrain.dot(proj_matrix),proj_matrix.T) / (np.linalg.
 ↪norm(proj_matrix)**2)
#     X_proj = np.matmul(x.dot(proj_matrix),proj_matrix.T) / (np.linalg.
 ↪norm(proj_matrix)**2)
#     for i, xi in enumerate(X_proj):
#         #compute euclidean distance
#         distances = np.linalg.norm(X_proj_Tr - xi, ord =2, axis =1)
#         #gets indices of sorted distances
#         label = np.argsort(distances)
#         #gets labels of k smallest distances
#         label_pred = ytrain[label[:k]].ravel().astype(np.int)
#         #retrieves most common value found in labels
#           #bincount returns the count of each value(label)
#           #argmax returns the index of the first maximum value(count)
#         pred[i] = np.argmax(np.bincount(label_pred))
#     return pred
```

[14]:
```
# #model pipeline
# K = [1,5,9,15]
# knn_train_err = []
# knn_valid_err = []
# for k in K:
#     #knn on training data
#     train_correct = knn_2(k, Xtrain,ytrain) == ytrain
#     train_error = 1 - (sum(train_correct) / len(train_correct))
#     knn_train_err.append(train_error)
#     #knn on validation data
#     valid_correct = knn_2(k, Xvalid,yvalid) == yvalid
#     valid_error = 1 - (sum(valid_correct) / len(valid_correct))
#     knn_valid_err.append(valid_error)
```

[15]:
```
def knn_3(k, x, y):
    pred = [0]*len(y)
    X_proj_Tr = np.matmul(Xtrain,proj_matrix)
    X_proj = np.matmul(x,proj_matrix)
    for i, xi in enumerate(X_proj):
        #compute euclidean distance
        distances = np.linalg.norm(X_proj_Tr - xi, ord =2, axis =1)
        #gets indices of sorted distances
        label = np.argsort(distances)
        #gets labels of k smallest distances
        label_pred = ytrain[label[:k]].ravel().astype(np.int)
        #retrieves most common value found in labels
          #bincount returns the count of each value(label)
          #argmax returns the index of the first maximum value(count)
        pred[i] = np.argmax(np.bincount(label_pred))
    return pred
```

[16]:
```
#model pipeline
K = [1,5,9,15]
knn_train_err = []
knn_valid_err = []
for k in K:
    #knn on training data
    train_correct = knn_3(k, Xtrain,ytrain) == ytrain
    train_error = 1 - (sum(train_correct) / len(train_correct))
    knn_train_err.append(train_error)
    #knn on validation data
    valid_correct = knn_3(k, Xvalid,yvalid) == yvalid
    valid_error = 1 - (sum(valid_correct) / len(valid_correct))
    knn_valid_err.append(valid_error)
```

[17]:
```
# pd.DataFrame({'k': K, 'training error':knn_train_err, 'validation error':␣
 ↪knn_valid_err})
```

[18]:
```
pd.DataFrame({'k': K, 'training error':knn_train_err, 'validation error':␣
 ↪knn_valid_err})
```

[18]:
```
    k  training error  validation error
0   1          0.0000             0.320
1   5          0.1945             0.299
2   9          0.2305             0.302
3  15          0.2570             0.289
```

[19]:
```
train_correct = knn_3(3, Xtrain,ytrain) == ytrain
k3_train_error = 1 - (sum(train_correct) / len(train_correct))
print('k = 3 training error: ', k3_train_error)
```

```
k = 3 training error:  0.16049999999999998
```

0.0.3 k =15 gives the lowest validation error / (15NN) performs best on the validation data

[20]:
```
K = [15]
knn_test_err = []
for k in K:
    #knn on test data
    test_correct = knn_3(k, Xtest,ytest) == ytest
    test_error = 1 - (sum(test_correct) / len(test_correct))
    knn_test_err.append(test_error)
```

[21]:
```
pd.DataFrame({'k': K, 'test error':knn_test_err})
```

[21]:
```
    k  test error
0  15       0.296
```

0.0.4 the classificaton accuracy suffers a bit from projection, however makes up for its performance in significant runtime reduction with the projected data

[ ]:
```
```