modularized_lda

March 6, 2024

1 Linear Descriminant Analysis

import the required libraries

[1]:
```
import numpy as np
import scipy as sp
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import DatasetSplitter as ds
import matplotlib.pyplot as plt

training_data, training_labels, testing_data, testing_labels = ds.splitData()
```

Split the training data into 40 classes

[3]:
```
def split_data(training_data):
    mat = np.array(training_data)
    return np.split(mat, indices_or_sections=40)
```

Calculate the class means

[4]:
```
def get_class_means(split_data):
    return np.mean(split_data, axis=1)
```

Calculate the overall mean

[5]:
```
def get_overall_mean(training_data):
    return np.mean(training_data, axis=0)
```

Calculate the centered classes matrix

[6]:
```
def get_centered_classes(split_data, class_means):
    centered_classes = []
    for class_matrix, class_mean in zip(split_data, class_means):
        centered_class = []
        for row in class_matrix:
            centered_class.append(row - class_mean)
        centered_classes.append(centered_class)
    return centered_classes
```

Calculate the between-class scatter matrix (Sb)

[7]:
```
def get_between_class(class_means, overall_mean):
    Sb = np.zeros((10304, 10304))
    for class_mean in class_means:
        Sb += 5 * np.outer(class_mean - overall_mean, class_mean - overall_mean)
    return Sb
```

Calculate the within-class scatter matrix and its inverse (Sw, Sw-1)

[8]:
```
def get_within_class(centered_classes):
    Sw = np.zeros((10304, 10304))
    for centered_class in centered_classes:
        mat = np.array(centered_class)
        Sw += np.dot(np.transpose(mat), mat)
    Sw_inverse = sp.linalg.pinv(Sw)
    return Sw_inverse
```

Calculate the eigenvalues and eigenvectors for the inverse of Sw multiplied by Sb

[9]:
```
def get_eigens(Sw_inverse, Sb):
    eigenvalues, eigenvectors = np.linalg.eig(np.dot(Sw_inverse, Sb))
    return eigenvalues, eigenvectors
```

Calculate the 39 most significant eigenvectors (Corresponding to the most significant eigenvalues)

[10]:
```
def get_U(eigenvalues, eigenvectors):
    sorted_indecies = np.argsort(eigenvalues)[::-1]
    sorted_eigen_vectors = eigenvectors[:,sorted_indecies]
    U = np.real(sorted_eigen_vectors[:,:39])
    return U
```

The LDA method

[12]:
```
def LDA(training_data):
    splited_data = split_data(training_data)
    class_means = get_class_means(splited_data)
    overall_mean = get_overall_mean(training_data)
    centered_classes = get_centered_classes(splited_data, class_means)
    Sb = get_between_class(class_means, overall_mean)
    Sw_inverse = get_within_class(centered_classes)
    eigenvalues, eigenvectors = get_eigens(Sw_inverse, Sb)
    U = get_U(eigenvalues, eigenvectors)
    return U
```

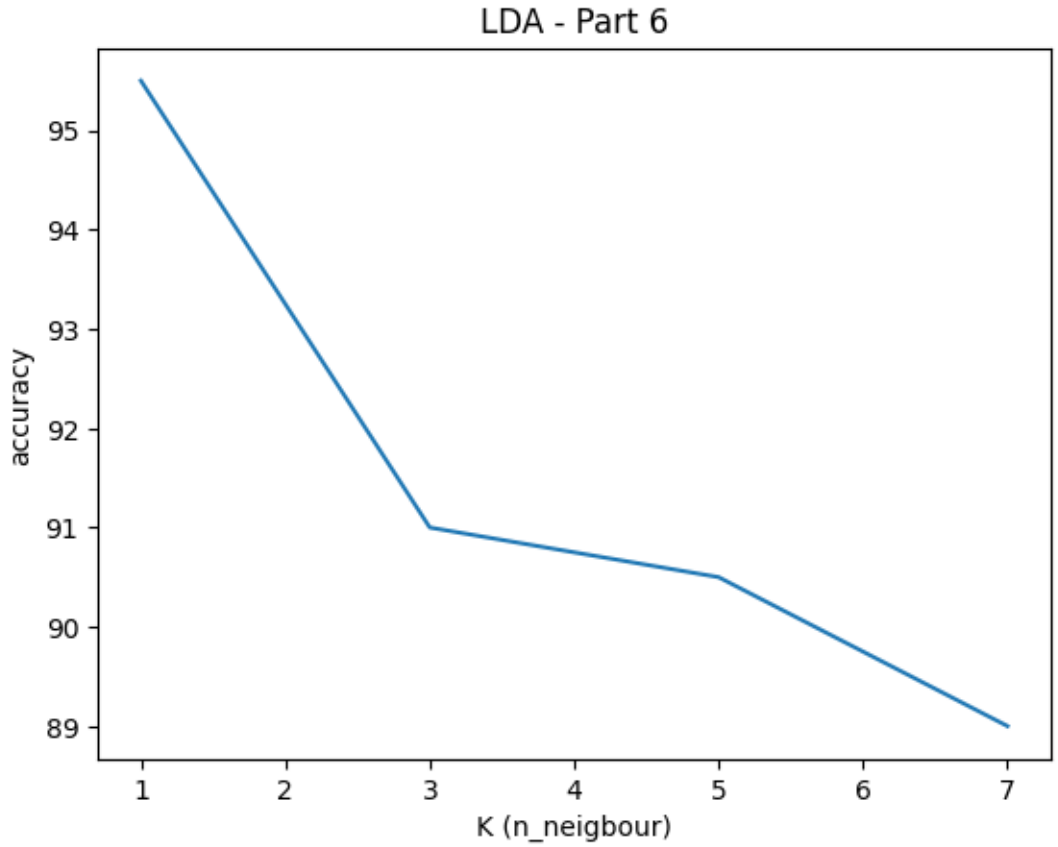
The Driver Code

[12]:
```
ks = [1, 3, 5, 7]
accuracy_results = np.array([])
projection_matrix = LDA(training_data)
projected_training_data = np.dot(training_data, projection_matrix)
projected_testing_data = np.dot(testing_data, projection_matrix)
```

```
for k in ks:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(projected_training_data, training_labels)
    predicted_labels = knn.predict(projected_testing_data)
    accuracy = accuracy_score(testing_labels, predicted_labels)
    accuracy_results = np.append(accuracy_results, accuracy * 100)
    print("Accuracy for k = " + str(k) + " is " + str(accuracy * 100))
```

```
Accuracy for k = 1 is 95.5
Accuracy for k = 3 is 91.0
Accuracy for k = 5 is 90.5
Accuracy for k = 7 is 89.0
```

[13]:
```
plt.plot(ks, accuracy_results)
plt.xlabel('K (n_neigbour)')
plt.ylabel('accuracy')
plt.title('LDA - Part 6')
plt.show()
```



[13]:
```
training_data70_30, training_labels70_30, testing_data70_30,␣
 ↪testing_labels70_30 = ds.differentSplitData()
projection_matrix70_30 = LDA(training_data70_30)
projected_training_data70_30 = np.dot(training_data70_30,␣
 ↪projection_matrix70_30)
projected_testing_data70_30 = np.dot(testing_data70_30, projection_matrix70_30)
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(projected_training_data70_30, training_labels70_30)
predicted_labels70_30 = knn.predict(projected_testing_data70_30)
accuracy70_30 = accuracy_score(testing_labels70_30, predicted_labels70_30)
print("Accuracy for k = 1 is " + str(accuracy70_30 * 100))
```

```
Accuracy for k = 1 is 95.0
```