COSC 3337 : Data Science I

N. Rizk

College of Natural and Applied Sciences

Department of Computer Science

University of Houston

Review PCA

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.

The number of principal components is less than or equal to the number of original variables.

Goals

- The main goal of a PCA analysis is to identify patterns in data
- PCA aims to detect the correlation between variables.
- It attempts to reduce the dimensionality.

Dimensionality Reduction

It reduces the dimensions of a d-dimensional dataset by projecting it onto a (k)-dimensional subspace (where k<d) in order to increase the computational efficiency while retaining most of the information.

Transformation

This transformation is defined in such a way that the first principal component has the largest possible variance and each succeeding component in turn has the next highest possible variance.

PCA Approach

- Standardize the data.
- Perform Singular Vector Decomposition to get the Eigenvectors and Eigenvalues.
- Sort eigenvalues in descending order and choose the k-eigenvectors
- Construct the projection matrix from the selected k-eigenvectors.
- Transform the original dataset via projection matrix to obtain a k-dimensional feature subspace.

Limitation of PCA

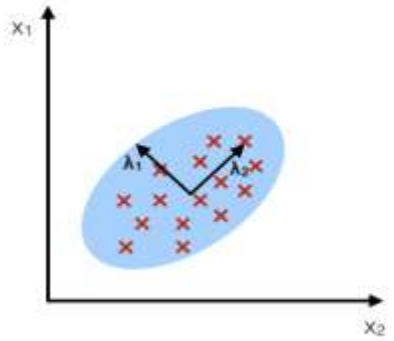
The results of PCA depend on the scaling of the variables.

A scale-invariant form of PCA has been developed.

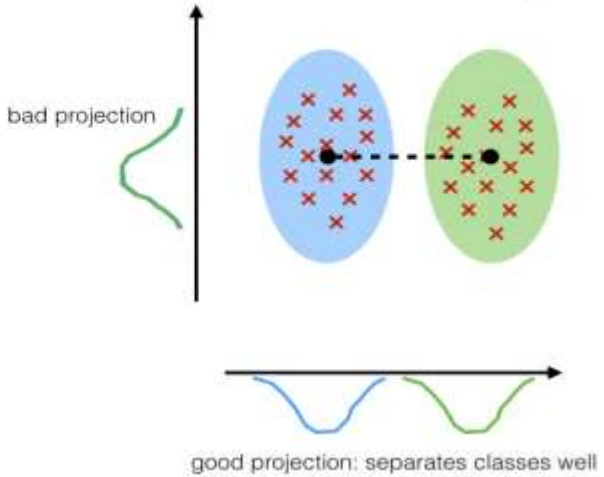
Fisher Linear Discriminant Analysis

- In a two-class classification problem, given n samples in a d-dimensional feature space. n1 in class 1 and n2 in class 2.
- Goal: to find a vector w, and project the n samples on the axis y=w'x, so that the projected samples are well separated.

PCA:
component axes that maximize the variance



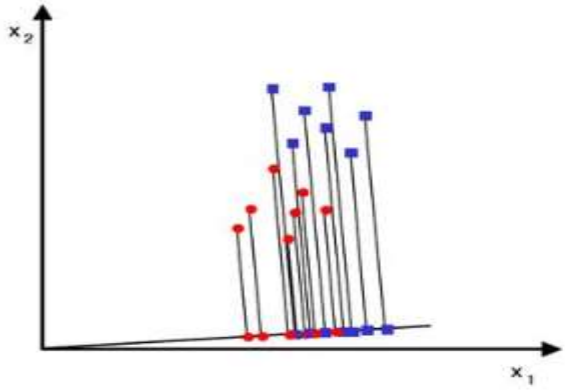
LDA:
maximizing the component axes for class-separation



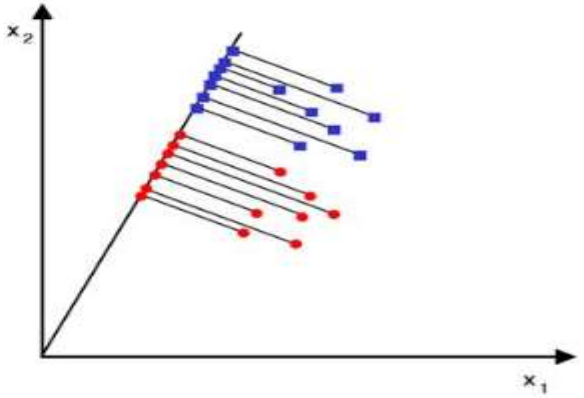
LDA vs PCA

- Both are linear transformation techniques that are commonly used for dimensionality reduction.
- PCA can be described as an “unsupervised” algorithm(it “ignores” class labels)
- PCA goal is to find the directions (the so-called principal components) that maximize the variance in a dataset.
- LDA is “supervised” and computes the directions (“linear discriminants”) that will represent the axes that that maximize the separation between multiple classes.

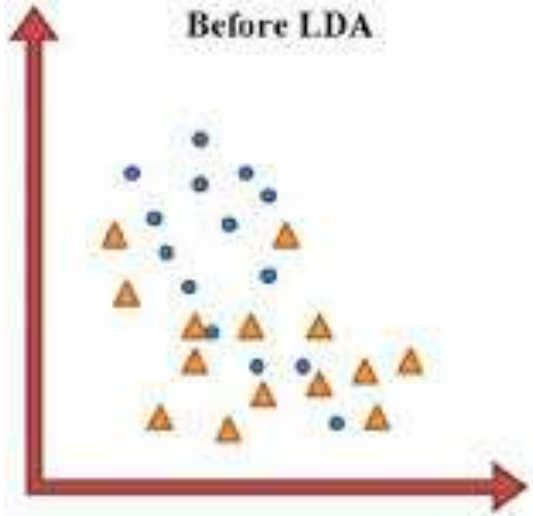
Find the axes that reduce dimensionality and preserve as much class discriminatory (distinction) information as possible

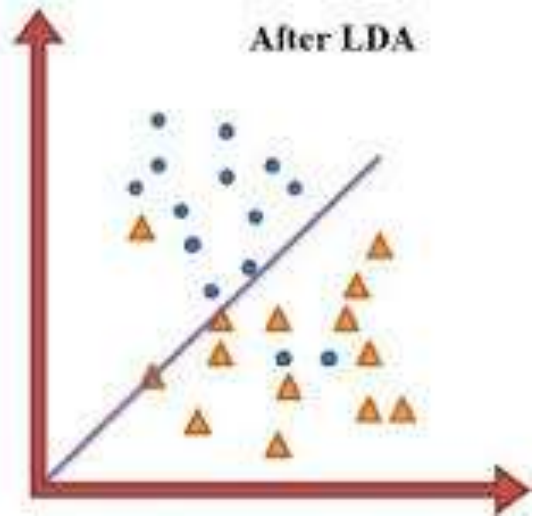


A projection with non-ideal separation



A projection with ideal separation

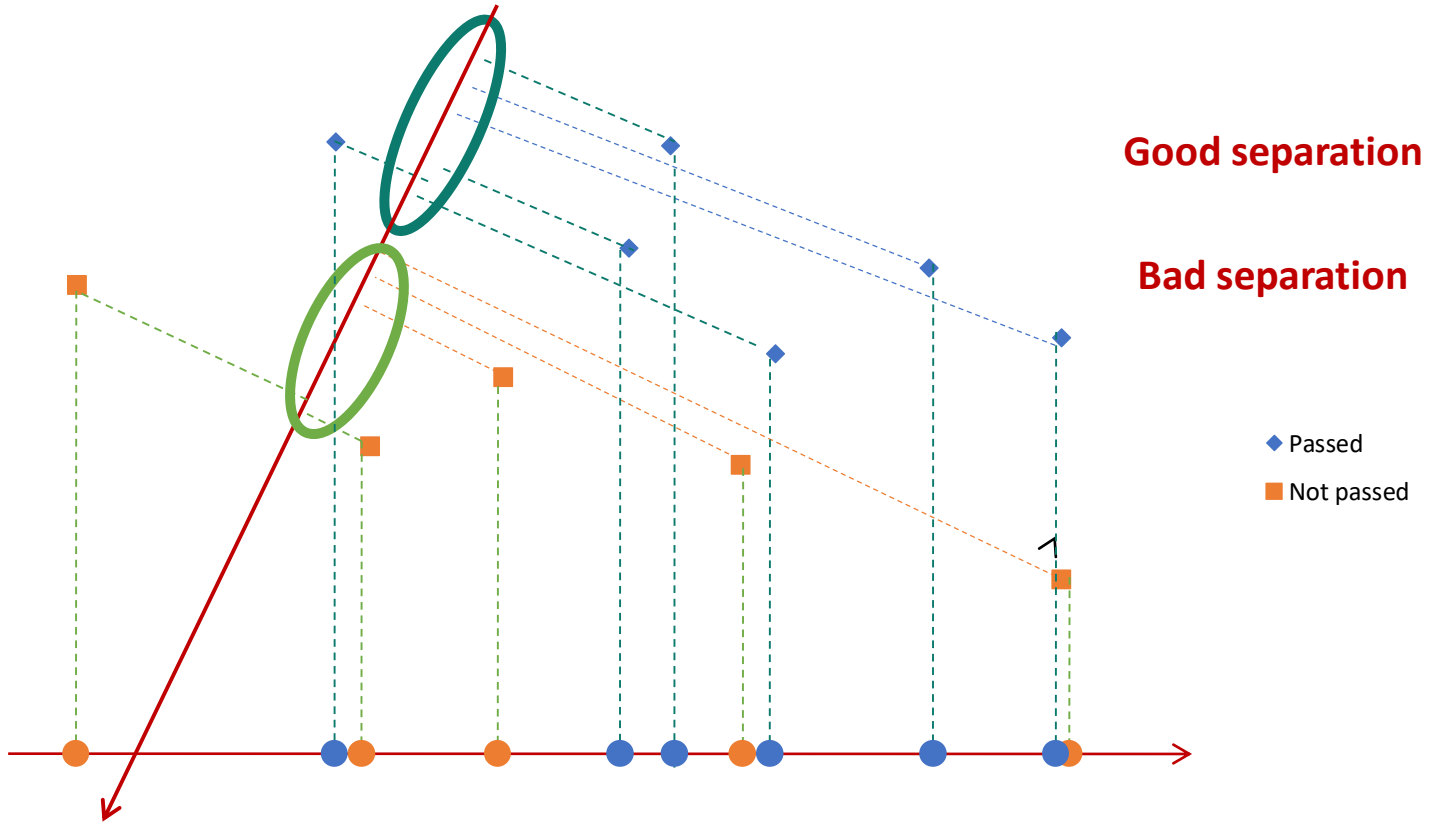




Purpose

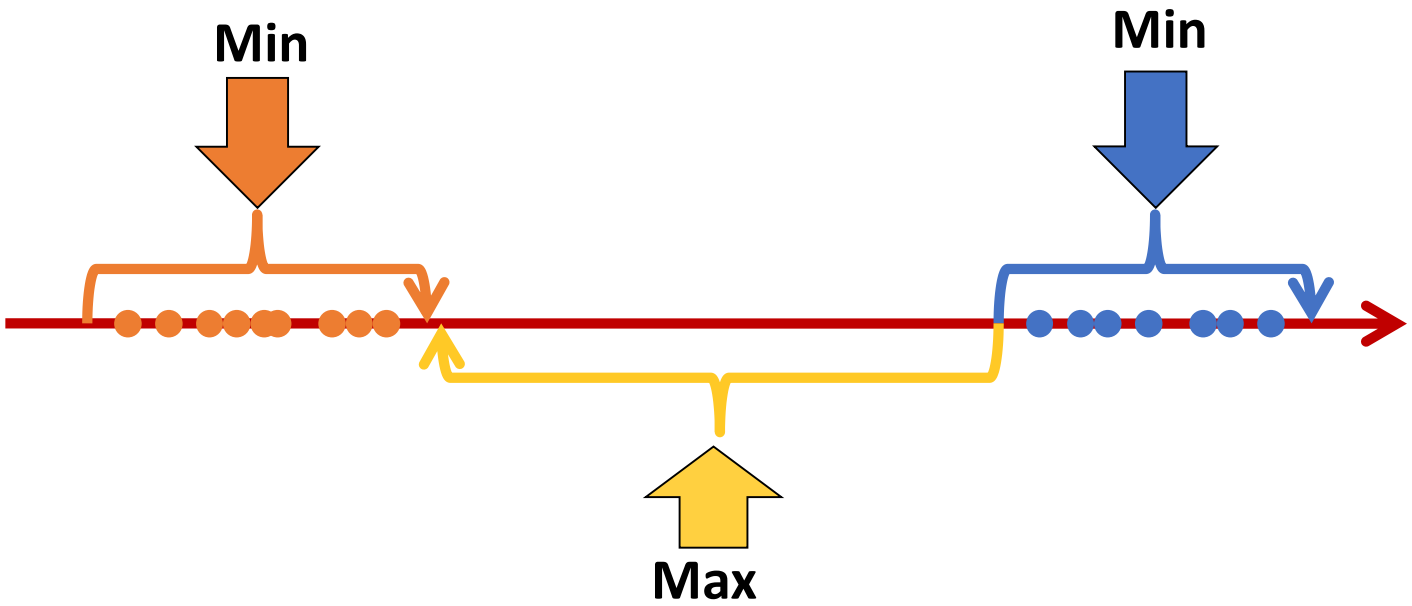
- Discriminant Analysis classifies objects
  in two or more groups
  according to *linear combination* of features
- **Feature selection**
  - Which *set of features* can best determine group membership of the object?
  - dimension reduction
- **Classification**
  - What is the classification *rule* or *model* to best separate those groups?

Visually



Method

- Maximize the between-class scatter
  - Difference of mean values (m1-m2)
- Minimize the within-class scatter
  - Covariance



LDA Approach

1. Compute the within class and between class scatter matrices
2. Compute the eigenvectors and corresponding eigenvalues for the scatter matrices
3. Sort the eigenvalues and select the top ***k***
4. Create a new matrix containing eigenvectors that map to the ***k*** eigenvalues
5. Obtain the new features (i.e. LDA components) by taking the dot product of the data and the matrix from step 4

Example: Wine data set

```
from sklearn.datasets import load_wine
import pandas as pd
import numpy as np
np.set_printoptions(precision=4)
from matplotlib import pyplot as plt
import seaborn as sns
sns.set()
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

```
wine = load_wine()
X = pd.DataFrame(wine.data, columns=wine.feature_names)
y = pd.Categorical.from_codes(wine.target, wine.target_names)
```

```
X.shape
```

```
(178, 13)
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/o
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	

1.Scatter Matrix

S
Orange
Mean C
Orange

S
Blue
Mean
Blue

$$S_W = \sum_{i=1}^{c} S_i$$

```
#Compute the within class and between class scatter matrices
class_feature_means = pd.DataFrame(columns=wine.target_names)
for c, rows in df.groupby('class'):
    class_feature_means[c] = rows.mean()
class_feature_means
```

and

$n_i)^T$

	class_0	class_1	class_2
alcohol	13.744746	12.278732	13.153750
malic_acid	2.010678	1.932676	3.333750
ash	2.455593	2.244789	2.437083
alcalinity_of_ash	17.037288	20.238028	21.416667
magnesium	106.338983	94.549296	99.312500
total_phenols	2.840169	2.258873	1.678750
flavanoids	2.982373	2.080845	0.781458
nonflavanoid_phenols	0.290000	0.363662	0.447500
proanthocyanins	1.899322	1.630282	1.153542
color_intensity	5.528305	3.086620	7.396250

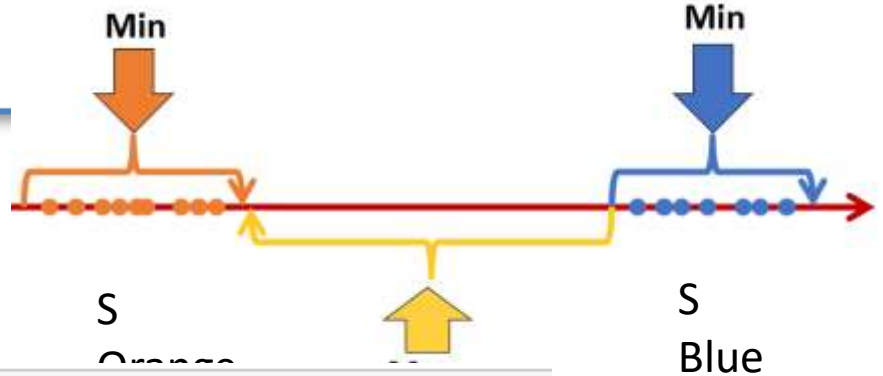
1.Scatter Matrix…cont


```
# mean vectors (mi) into the equation from before in order to obtain the within class scatter matrix
within_class_scatter_matrix = np.zeros((13,13))
for c, rows in df.groupby('class'):
    rows = rows.drop(['class'], axis=1)
    s = np.zeros((13,13))
for index, row in rows.iterrows():
        x, mc = row.values.reshape(13,1), class_feature_means[c].values.reshape(13,1)

        s += (x - mc).dot((x - mc).T)

        within_class_scatter_matrix += s
```

```
within_class_scatter_matrix
```

```
array([[ 2.9229e+02,  6.2033e+01,  1.4718e+01,  1.9552e+02, -6.7906e+02,
         3.0044e+01,  3.9477e+00,  1.1577e+00,  7.3445e+01,  4.5908e+02,
        -5.8483e+00,  4.0079e+01, -1.3837e+04],
       [ 6.2033e+01,  1.5010e+03,  1.4554e+01,  5.0418e+02, -3.8950e+03,
        -7.4734e+01, -1.3327e+02,  3.6566e+01, -9.6153e+01, -3.2497e+02,
         9.7079e+00,  2.0324e+01, -9.8806e+03],
       [ 1.4718e+01,  1.4554e+01,  3.8050e+01,  3.5658e+02,  4.2941e+02,
         2.8423e+01,  1.1072e+01,  9.1072e-01,  8.6140e+00,  4.7579e+01,
```

1.Scatter Matrix…cont

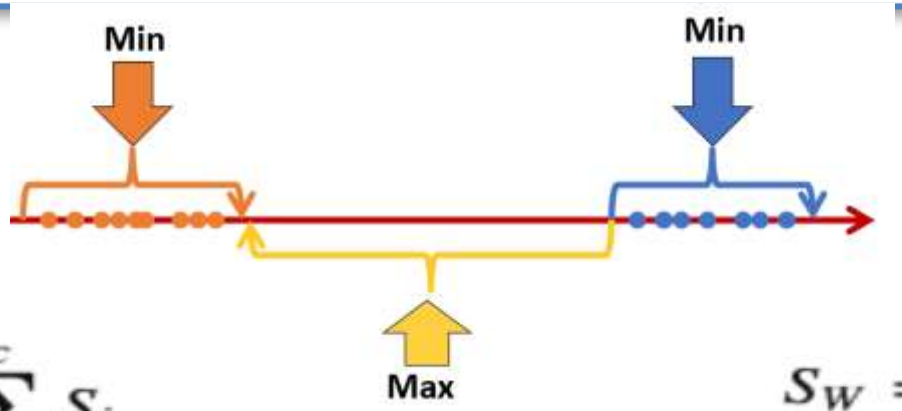



```
# between class scatter matrix
feature_means = df.mean()
between_class_scatter_matrix = np.zeros((13,13))
for c in class_feature_means:
    n = len(df.loc[df['class'] == c].index)

    mc, m = class_feature_means[c].values.reshape(13,1), feature_means.values.reshape(13,1)

    between_class_scatter_matrix += n * (mc - m).dot((mc - m).T)
```

```
between_class_scatter_matrix
```

```
array([[ 7.0795e+01,  1.3723e+01,  1.0668e+01, -1.3186e+02,  5.5262e+02,
         2.1257e+01,  3.0029e+01, -2.6178e+00,  8.3076e+00,  1.3888e+02,
        -2.4933e+00,  8.2530e+00,  2.6987e+04],
       [ 1.3723e+01,  6.5578e+01,  5.1556e+00,  1.1793e+02,  1.5062e+00,
        -3.8943e+01, -7.9531e+01,  5.4292e+00, -2.7994e+01,  1.5941e+02,
        -1.7995e+01, -5.9906e+01, -6.1709e+03],
       [ 1.0668e+01,  5.1556e+00,  1.7592e+00, -1.2829e+01,  7.8095e+01,
         1.0900e+00,  3.3671e-01, -1.0316e-01, -2.0087e-01,  2.7430e+01,
        -1.2351e+00, -1.7747e+00,  3.5073e+03],
```

2.Compute eigen values & vectors



$$S_W = \sum_{i=1}^{c} S_i$$

$$S_W = \sum_{i=1}^{c} S_i$$

$$S_B = \sum_{i=1}^{c} N_i(\boldsymbol{m}_i - \boldsymbol{m})(\boldsymbol{m}_i - \boldsymbol{m})^T$$

```
eigen_values, eigen_vectors =
np.linalg.eig(np.linalg.inv(within_class_scatter_matrix).dot(between_class_scatter_matrix
))
```

$$S_W^{-1} S_B$$

2. Compute the eigenvectors and eigenvalues

```
eigen_values, eigen_vectors = np.linalg.eig(np.linalg.inv(within_class_scatter_matrix).dot(between_class_scatter_matrix))
```

```
pairs = [(np.abs(eigen_values[i]), eigen_vectors[:,i]) for i in range(len(eigen_values))]
pairs = sorted(pairs, key=lambda x: x[0], reverse=True)
for pair in pairs:
    print(pair[0])
```

```
9.884546449232964
2.903361061716055
8.328748314463861e-16
8.328748314463861e-16
4.886713002064387e-16
4.489875014453336e-16
3.4635722681863882e-16
3.4635722681863882e-16
1.0056538289828519e-16
7.842049664151311e-17
7.842049664151311e-17
5.900795033178523e-17
0.0
```

```
eigen_value_sums = sum(eigen_values)
print('Explained Variance')
for i, pair in enumerate(pairs):
    print('Eigenvector {}: {}'.format(i, (pair[0]/eigen_value_sums).real))
```

```
Explained Variance
Eigenvector 0: 0.7729604269322252
Eigenvector 1: 0.2270395730677747
Eigenvector 2: 6.51298760749777e-17
Eigenvector 3: 6.51298760749777e-17
Eigenvector 4: 3.821354664850663e-17
Eigenvector 5: 3.511031817057716e-17
Eigenvector 6: 2.7084746001023813e-17
Eigenvector 7: 2.7084746001023813e-17
Eigenvector 8: 7.864099956320532e-18
Eigenvector 9: 6.132394731066783e-18
Eigenvector 10: 6.132394731066783e-18
Eigenvector 11: 4.614355419857593e-18
Eigenvector 12: 0.0
```

3. Sort the eigenvalues and select the top ***k***

```
#Create a matrix W with the first two eigenvectors.
w_matrix = np.hstack((pairs[0][1].reshape(13,1), pairs[1][1].reshape(13,1))).real
```

```
# dot product of X and W into a new matrix Y . Y= X.W
X_lda = np.array(X.dot(w_matrix))
X_lda
```

```
[-3.0367,  3.5544],
[-2.8555,  3.8611],
[-3.049 ,  3.5449],
[-3.1342,  3.2478],
[-3.455 ,  3.5023],
[-2.7984,  3.6056],
[-3.0704,  3.6012],
[-3.7015,  3.5086],
[-3.6974,  3.7308],
[-2.9534,  3.8804],
[-3.0524,  3.9394],
[-3.166 ,  3.7823],
[-3.8682,  4.2839],
[-2.8555,  3.2937],
[-2.9092,  2.9224],
[-2.5376,  2.7191],
[-3.1722,  3.03  ],
[-2.7119,  3.0313],
[-2.8484,  2.8515],
```

```
le = LabelEncoder()
y = le.fit_transform(df['class'])
```

4.Create a new matrix containing eigenvectors that map to the ***k*** eigenvalues

Y = X .W

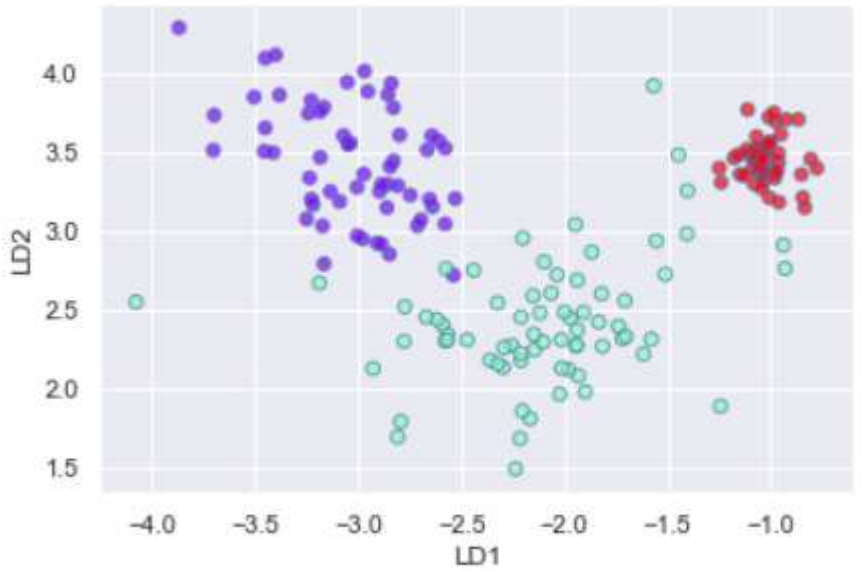
X is a ***n×d*** matrix with ***n*** samples and ***d*** dimensions
Y is a ***n×k*** matrix with ***n*** samples and ***k*** (k<n) dimensions.

➔ ***Y*** is composed of the LDA components
➔ It is the new feature space.

5. Obtain the new features (i.e. LDA components)

```
plt.xlabel('LD1')
plt.ylabel('LD2')
plt.scatter(
    X_lda[:,0],
    X_lda[:,1],
    c=y,
    cmap='rainbow',
    alpha=0.7,
    edgecolors='b'
)
```

```
<matplotlib.collections.PathCollection at 0x14e33a5eac8>
```



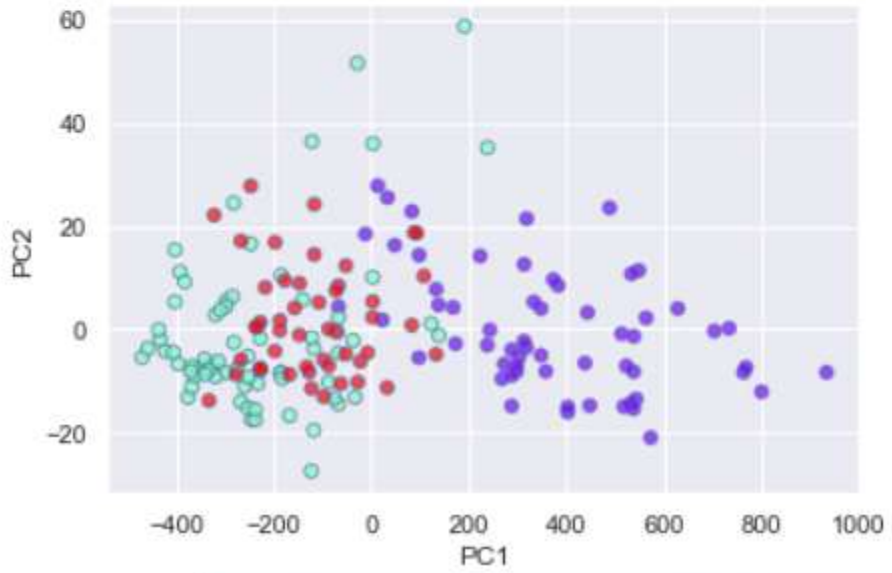
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X, y)
```

```
pca.explained_variance_ratio_
```

```
array([0.9981, 0.0017])
```

```
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.scatter(
    X_pca[:,0],
    X_pca[:,1],
    c=y,
    cmap='rainbow',
    alpha=0.7,
    edgecolors='b')
```

```
<matplotlib.collections.PathCollection at 0x14e33c17148>
```




```
#using sklearn
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
X_lda = lda.fit_transform(X, y)
```

```
lda.explained_variance_ratio_
```

```
array([0.6875, 0.3125])
```

```
plt.xlabel('LD1')
plt.ylabel('LD2')
plt.scatter(
    X_lda[:,0],
    X_lda[:,1],
    c=y,
    cmap='rainbow',
    alpha=0.7,
    edgecolors='b'
)
```

```
<matplotlib.collections.PathCollection at 0x14e33a5ea88>
```

Accuracy scores LDA, PCA , none

```
def run_randomForest(X_train,X_test,y_train,y_test):
    clf=RandomForestClassifier(n_estimators=100, random_state=0,n_jobs=-1)
    clf.fit(X_train,y_train)
    y_pred = clf.predict(X_test)
    print('Accuracy on test set: ')
    print(accuracy_score(y_test,y_pred))
```

```
%%time
from sklearn.metrics import accuracy_score
run_randomForest(X_train_lda,X_test_lda,y_train,y_test)
```

```
Accuracy on test set:
0.93725
Wall time: 795 ms
```

```
%%time
from sklearn.metrics import accuracy_score
run_randomForest(X_train_pca,X_test_pca,y_train,y_test)
```

```
Accuracy on test set:
0.9565
Wall time: 749 ms
```

```
%%time
run_randomForest(X_train,X_test,y_train,y_test)
```

```
Accuracy on test set:
0.9585
Wall time: 1.79 s
```