Assignment 2

Krutika Injamuri, M.Tech CS, 18MCMT20

Question 1 : Implement PCA and LDA

PCA

In [1]:

```
import numpy as np
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt


def sorted_values(eigen_values, eigen_vectors):
    eig_pairs = [(np.abs(eigen_values[i]), eigen_vectors[:,i]) for i in range(len(eigen_values))]
#     print(eig_pairs)
    eig_pairs.sort(reverse = True, key=lambda k: k[0])
    return [x[0] for x in eig_pairs], [y[1] for y in eig_pairs]

def pca(input_array,components,pcaIndex1,pcaIndex2):
    number_of_features = input_array.shape[1]
    scaler = StandardScaler()
    scaled_input = scaler.fit_transform(input_array)
    covariance_mat = np.cov(scaled_input.T) # Mean centeres the data and finds the covarience
    eigen_values, eigen_vectors = np.linalg.eig(covariance_mat)
    eigen_values, eigen_vectors = sorted_values(eigen_values, eigen_vectors)
    stacked_eigen_mat = np.hstack((eigen_vectors[pcaIndex1].reshape(number_of_features,1),eigen_vectors[pcaI
ndex2].reshape(number_of_features,1)))

    return scaled_input.dot(stacked_eigen_mat)
```

LDA

In [2]:

```
def within_class_scatter_matrix(data , mean_vectors, number_of_features, number_of_classes, unequal_class_sa
mples = False):
    S_W = np.zeros((number_of_features,number_of_features))
    for class_val,mean_vec in enumerate(mean_vectors):
        scatter_matrix = np.zeros((number_of_features,number_of_features))

        # scatter matrix for every class
        for data_instance in data[class_val]:
            data_instance, mean_vec = data_instance.reshape(number_of_features,1), mean_vec.reshape(number_o
f_features,1) # make column vectors
            scatter_matrix += (data_instance-mean_vec).dot((data_instance-mean_vec).T)
        S_W += scatter_matrix
    return S_W

def between_class_scatter_matrix(data , mean_vectors, number_of_features, number_of_classes, unequal_class_s
amples = False):
    S_B = np.zeros((number_of_features,number_of_features))
    for class_val,mean_vec in enumerate(mean_vectors):
        n = data[class_val].shape[0]
        mean_vec = mean_vec.reshape(number_of_features,1)
        overall_mean = np.mean(mean_vectors, axis=0).reshape(number_of_features,1)
        S_B += n * (mean_vec - overall_mean).dot((mean_vec - overall_mean).T)

    return S_B


def lda(data, components, unequal_class_samples = False):
    if(unequal_class_samples):
        number_of_classes = data.shape[0]
        number_of_features = data[0].shape[1]

        mean_vectors = np.mean(data, axis=1)
        S_W = within_class_scatter_matrix(data ,mean_vectors ,number_of_features, number_of_classes,unequal_
class_samples)
        S_B = between_class_scatter_matrix(data ,mean_vectors ,number_of_features, number_of_classes,unequal
_class_samples)
    else:
        number_of_classes = data.shape[0]
        number_of_features = data.shape[2]
        mean_vectors = np.mean(data, axis=1)
        S_W = within_class_scatter_matrix(data ,mean_vectors ,number_of_features, number_of_classes)
        S_B = between_class_scatter_matrix(data ,mean_vectors ,number_of_features, number_of_classes)

    eigen_values, eigen_vectors = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))
    eigen_values, eigen_vectors = sorted_values(eigen_values, eigen_vectors)
    stacked_eigen_mat = eigen_vectors[0]
    for _ in range(1,components):
        stacked_eigen_mat = np.hstack((stacked_eigen_mat.reshape(number_of_features, 1),eigen_vectors[_].res
hape(number_of_features, 1)))

    return data.dot(stacked_eigen_mat)
```

Question 2: Iris Dataset

In [3]:

```
iris_dataset = np.genfromtxt("data/iris.csv",delimiter=",")
iris_dataset = iris_dataset[:,:4]
iris_dataset = iris_dataset.reshape(3,50,4)
```

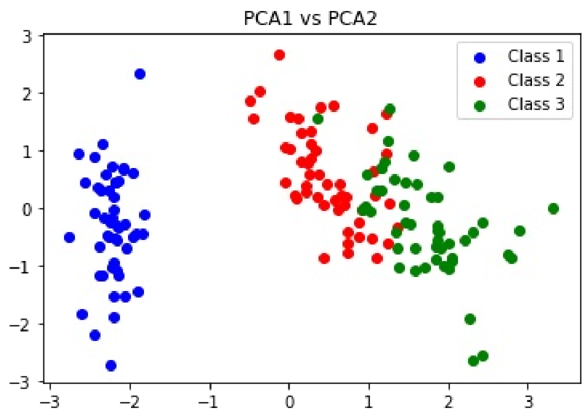
PCA on Iris Dataset

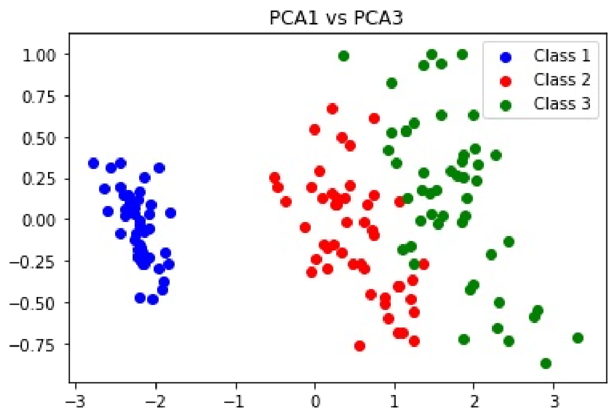
In [4]:

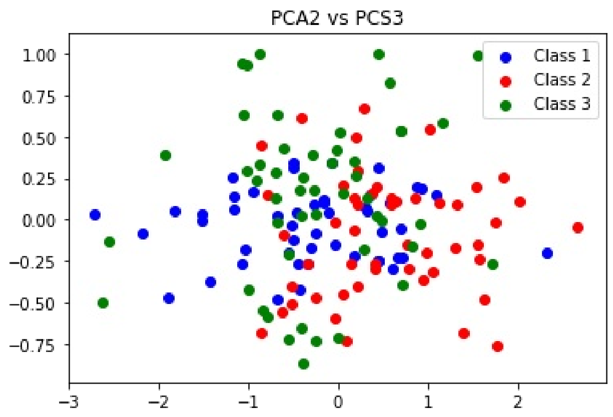
```
def plot_PCA(projected_data,title):
    projected_data = projected_data.reshape(3,50,2)
    plt.title(title)
    plt.scatter(x=projected_data[0,:,0], y=projected_data[0,:,1],c="b",label="Class 1")
    plt.scatter(x=projected_data[1,:,0], y=projected_data[1,:,1],c="r",label="Class 2")
    plt.scatter(x=projected_data[2,:,0], y=projected_data[2,:,1],c="g",label="Class 3")
    plt.legend()
    plt.show()
```

In [5]:

```
plot_PCA(pca(iris_dataset.reshape(150,4), 2,0,1),"PCA1 vs PCA2")
plot_PCA(pca(iris_dataset.reshape(150,4), 2,0,2), "PCA1 vs PCA3")
plot_PCA(pca(iris_dataset.reshape(150,4), 2,1,2), "PCA2 vs PCS3")
```



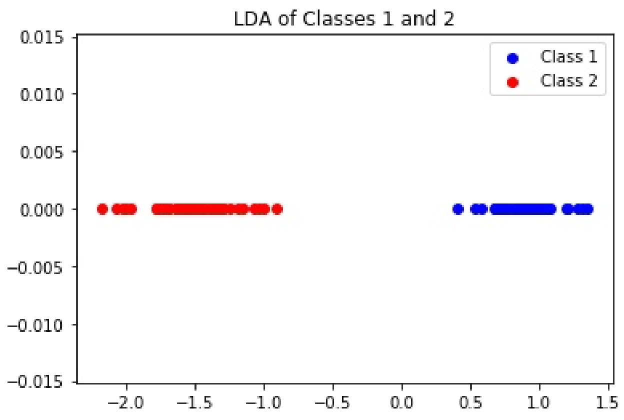


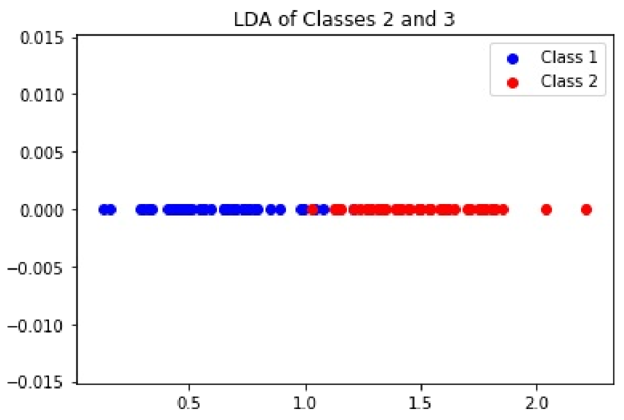


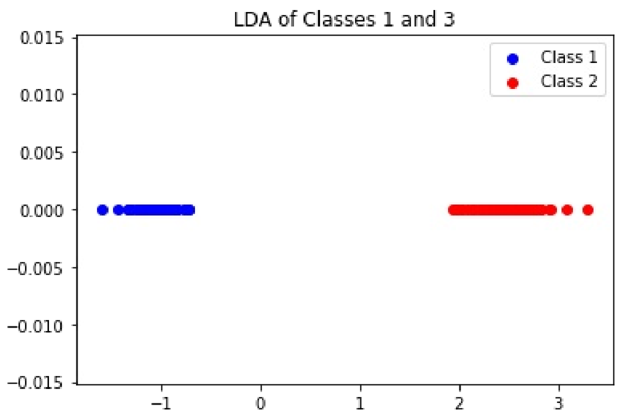
1-dimensional LDA on Iris Dataset

In [6]:

```
def plot_LDA(projected_data,title):
    plt.title(title)
    y_axis = np.zeros((50,1))
    plt.scatter(x=projected_data[0], y= y_axis, c ="b",label="Class 1")
    plt.scatter(x=projected_data[1], y= y_axis, c="r",label="Class 2")
    plt.legend()
    plt.show()

plot_LDA(lda(iris_dataset[:2, :,:], 1),"LDA of Classes 1 and 2")
plot_LDA(lda(iris_dataset[1:,:,:], 1),"LDA of Classes 2 and 3")
plot_LDA(lda(np.vstack((iris_dataset[0, :,:],iris_dataset[2, :,:])).reshape(2,50,4),1),"LDA of Classes 1 and
3")
```







Question 3: Arcene Cancer Dataset

Scree Plot

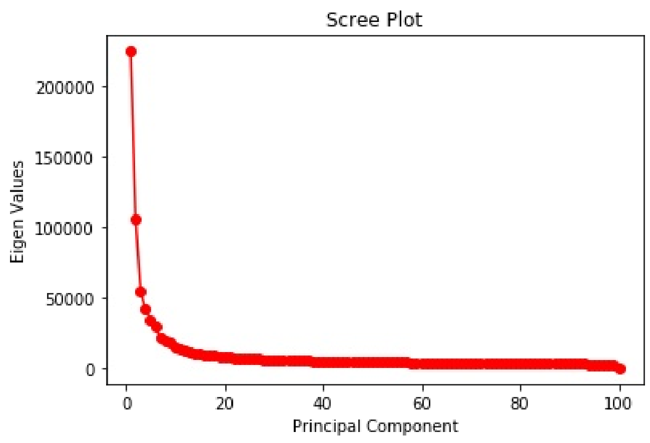
In [7]:

```
file = open('data/arcene_train.data')
X = np.array([list(map(int, file.readline().strip().split(' '))) for _ in range(100)])

file = open('data/arcene_train.labels')
y = np.array([int(file.readline().strip()) for _ in range(100)])

class1 = X[y == 1]
class2 = X[y == -1]
cancer_dataset = np.vstack((class1,class2))


scaler = StandardScaler()
X_std = scaler.fit_transform(cancer_dataset)
u,s,vt = np.linalg.svd(X_std)
```

```
/home/user/anaconda3/lib/python3.6/site-packages/sklearn/utils/validation.py:475: DataConversio
nWarning: Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [8]:

```
eigen_values = s**2
plt.plot(np.arange(len(eigen_values)) + 1, eigen_values, 'ro-')
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Eigen Values')
plt.show()
```



Number of Components to choose for required Variance

In [9]:

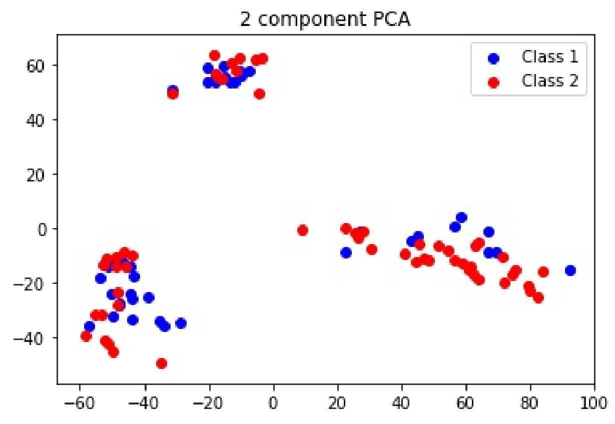
```
variances = np.cumsum(eigen_values)/np.sum(eigen_values)
for i,j in enumerate([0.85,0.90,0.95,0.99]):
    print("{}% variance is covered with {} components".format(j*100, np.where(variances >= j)[0][0]))
```

```
85.0% variance is covered with 53 components
90.0% variance is covered with 66 components
95.0% variance is covered with 81 components
99.0% variance is covered with 95 components
```

Projection of data into first two PCs

In [10]:

```
Y = X_std.dot(vt.T[:,:2])
plt.title("2 component PCA");
plt.scatter(x=Y[:44,0], y=Y[:44,1],c="b",label="Class 1")
plt.scatter(x=Y[44:,0], y=Y[44:,1],c="r",label="Class 2")
plt.legend()
plt.show()
```



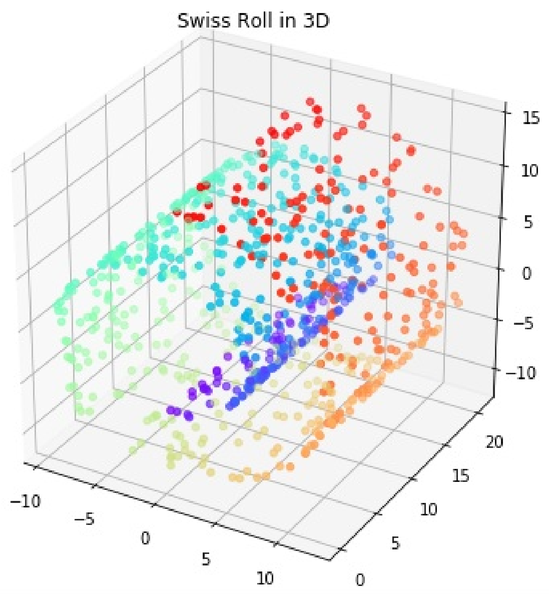
Comments:

- As the dimension of data is far greater than that of the number of samples, dual PCA is performed on the dataset rather than the normal PCA.
- 85%, 90%, 95% and 99% of variance is achieved using 53, 61, 83, 35 components respectively.
- As the variance is not much covered using the only 2 PCs we do not get a proper representation in the lower dimensional subspace.
- It is observed that 85% of variance is achieved using 53 components, so using just two Pronciple Components does not solve the purpose.

Question 4: Swiss Roll Dataset

In [11]:

```
from sklearn.datasets.samples_generator import make_swiss_roll
from mpl_toolkits.mplot3d import Axes3D
from sklearn.decomposition import PCA, KernelPCA
from sklearn.manifold import locally_linear_embedding


X, color = make_swiss_roll(n_samples=800, random_state=123)

fig = plt.figure(figsize=(7,7))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=color, cmap=plt.cm.rainbow)
plt.title('Swiss Roll in 3D')
plt.show()
```



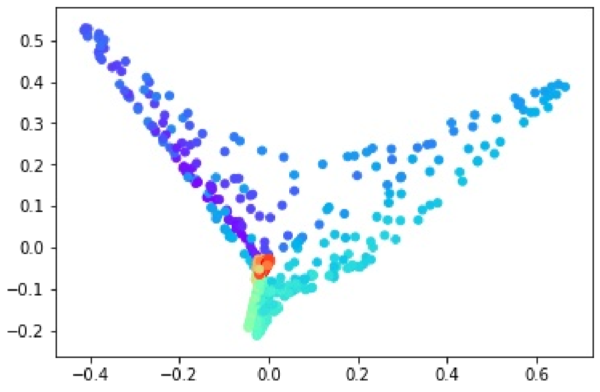
Kernel PCA

In [12]:

```
from sklearn.decomposition import KernelPCA
kpca = KernelPCA(kernel="rbf", fit_inverse_transform=True, gamma=0.1,n_components=2)
X_kpca = kpca.fit_transform(X)
plt.scatter(X_kpca[:, 0], X_kpca[:, 1], c=color, s=25,  cmap=plt.cm.rainbow)
```

Out[12]:

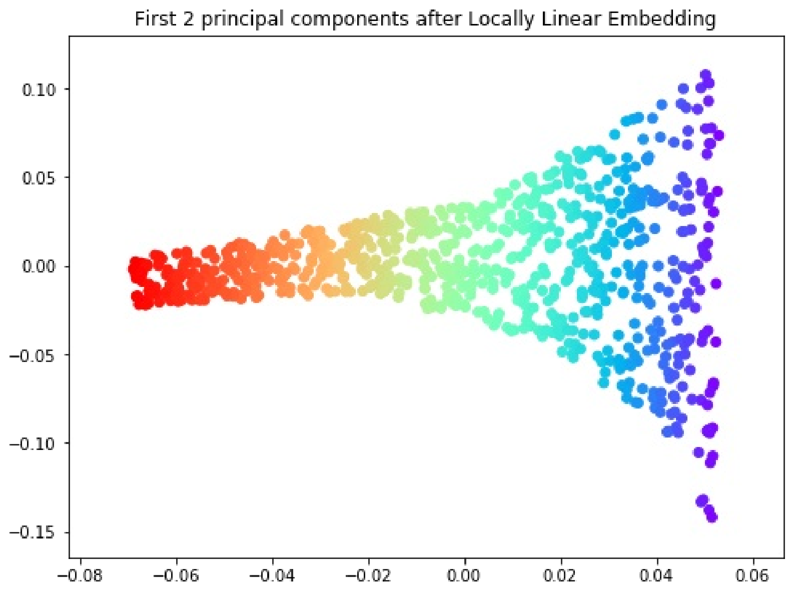
```
<matplotlib.collections.PathCollection at 0x7f5426801a90>
```



Local Linear Embedding

In [13]:

```
X_lle, err = locally_linear_embedding(X, n_neighbors=12, n_components=2)

plt.figure(figsize=(8,6))
plt.scatter(X_lle[:, 0], X_lle[:, 1], c=color, cmap=plt.cm.rainbow)

plt.title('First 2 principal components after Locally Linear Embedding')
plt.show()
```



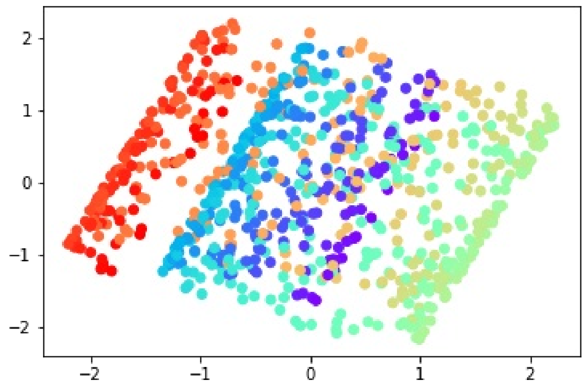
PCA

In [14]:

```
X_pca = pca(X,2,0,1)
plt.scatter(X_pca[:,0], X_pca[:,1],c = color, cmap=plt.cm.rainbow)
```

Out[14]:

```
<matplotlib.collections.PathCollection at 0x7f53e14e2f28>
```



Comments:

Kernel PCA:

- It performs PCA in the higher dimensions.
- It does non linear dimensionality reduction in the higher space.
- Non linear mapping is done in the dataset.

LLE:

- It unrolls and flattens the 3D swiss roll manifold dataset.
- It finds the local patches periodically and falttens the data set
- The transformation of dataset with LLE is way more better than PCA, Kernel PCA.
- The dataset can be classified easily with minimal error.
- The local structure is preserved, preventing the mixing of colors.

PCA:

- There is mixture of colors along the manifold.
- The dataset is not linearly seperable. It is very difficult to identify different classes of dataset.