! This class has been made inactive. No posts will be allowed until an instructor reactivates the class.

private note @39 2 views

HW2_2017310936_Md_Shirajum_Munir

I/ PCA Iris dataset:

1) Load (all features, using Pandas), standardize this d-dimension dataset (d is number of features) and Split the Iris dataset to training and test sets with ratio 70% and 30%, respectively.

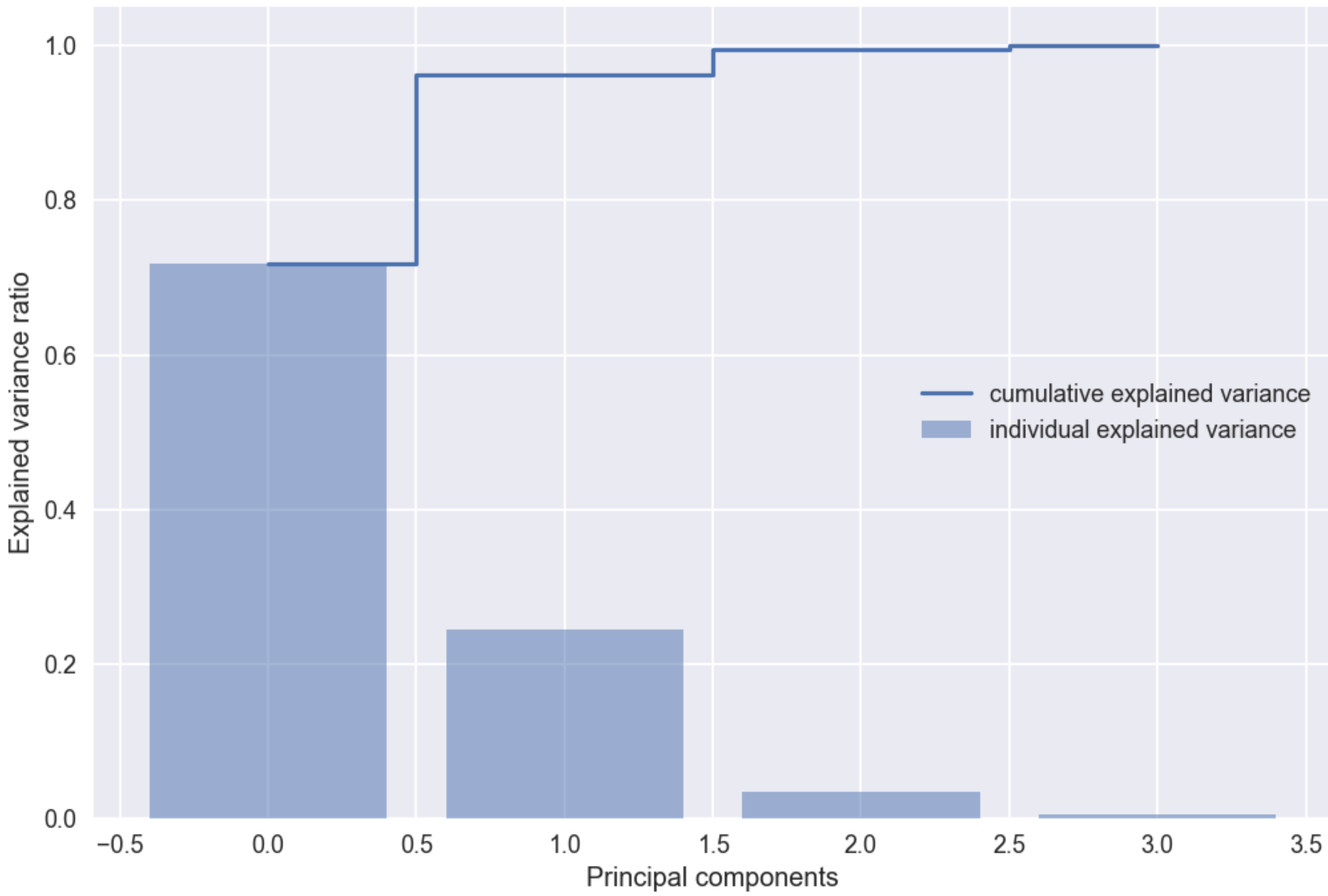
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import seaborn as sns


sns.set()
sns.set_context("talk")

#1) Load (all features, using Pandas), standardize this d-dimension dataset (d is number of features) and Split
# the Iris dataset to training and test sets with ratio 70% and 30%, respectively.

iris_df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data', header=None)
print(iris_df.head())
print(iris_df.tail())

X = iris_df.iloc[:,0:4].values
y = iris_df.iloc[:,4].values

X_train, X_test, y_train, y_test =  train_test_split(X, y, test_size=0.3, random_state=0)

sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)

Output:
iris_df.head():
0    1    2    3           4
0  5.1  3.5  1.4  0.2  Iris-setosa
1  4.9  3.0  1.4  0.2  Iris-setosa
2  4.7  3.2  1.3  0.2  Iris-setosa
3  4.6  3.1  1.5  0.2  Iris-setosa
4  5.0  3.6  1.4  0.2  Iris-setosa
iris_df.tail():
 0    1    2    3              4
145  6.7  3.0  5.2  2.3  Iris-virginica
146  6.3  2.5  5.0  1.9  Iris-virginica
147  6.5  3.0  5.2  2.0  Iris-virginica
148  6.2  3.4  5.4  2.3  Iris-virginica
149  5.9  3.0  5.1  1.8  Iris-virginica
```

2) Write your own function to calculate the covariance matrix. Then compute the eigenvalues and eigenvectors of this matrix.

```
mean_vec = np.mean(X_train_std, axis=0)
cov_mat = (X_train_std - mean_vec).T.dot((X_train_std - mean_vec)) / (X_train_std.shape[0]-1)
print('Covariance matrix \n%s' %cov_mat)
eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
print('\nEigenvalues \n%s' % eigen_vals)

Output:
Covariance matrix
[[ 1.00961538 -0.03658858  0.89282533  0.84055197]
 [-0.03658858  1.00961538 -0.3421826  -0.28950335]
 [ 0.89282533 -0.3421826   1.00961538  0.97784588]
 [ 0.84055197 -0.28950335  0.97784588  1.00961538]]

Eigenvalues
[ 2.89976476  0.98710977  0.13476983  0.01681717]
```

3) Plot the cumulative variance ratio (c.f. block [11] of the nbviewer of Chapter 5).

```
tot = sum(eigen_vals)
var_exp = [(i / tot) for i in sorted(eigen_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
plt.bar(range(0, 4), var_exp, alpha=0.5, align='center',label='individual explained variance')
plt.step(range(0, 4), cum_var_exp, where='mid', label='cumulative explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.legend(loc='center right')
plt.tight_layout()
plt.show()
```

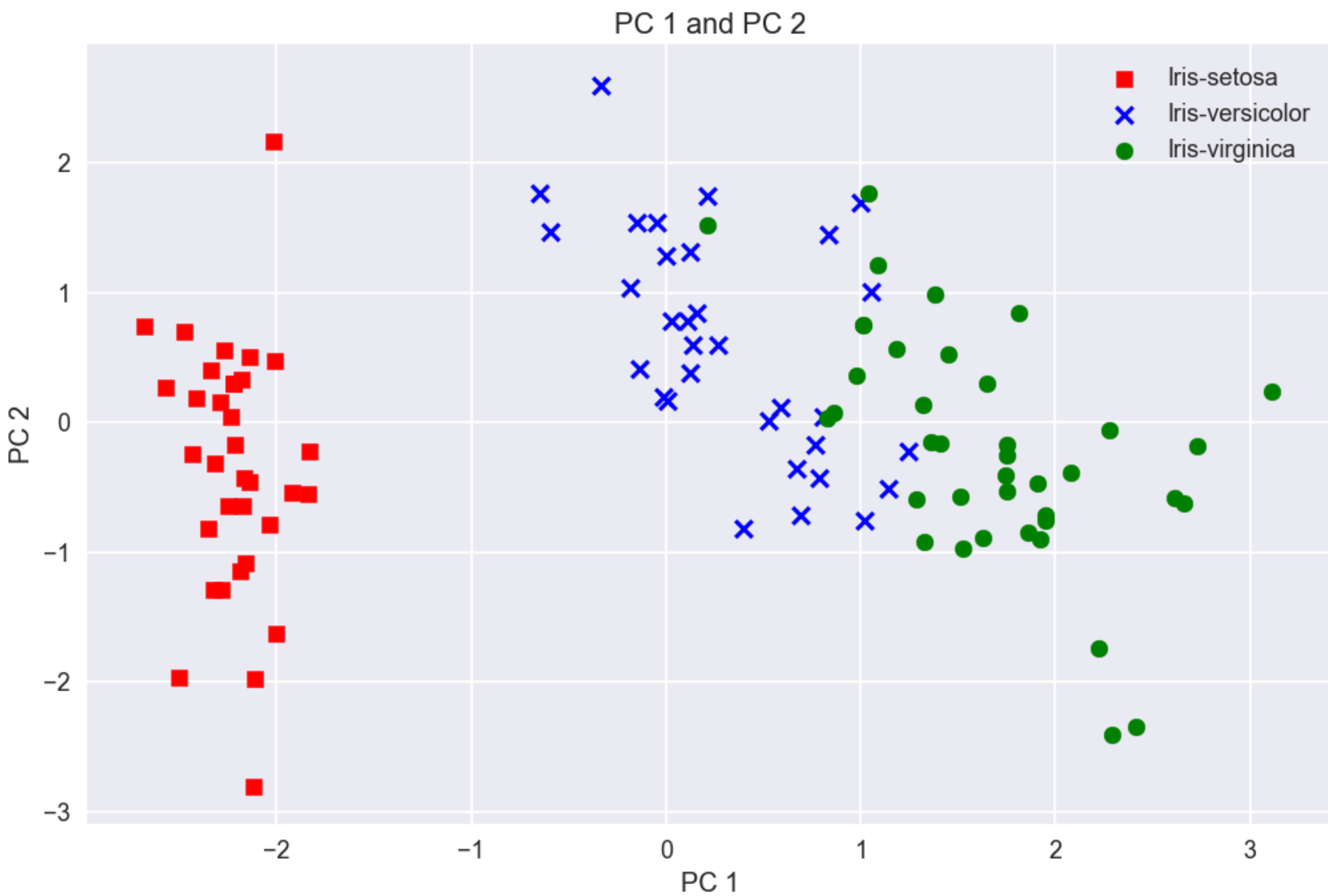


4) Choose the k=3 eigenvectors that correspond to the k largest eigenvalues to construct a d×k-dimensional transformation matrix W ; the eigenvectors are the columns of this matrix

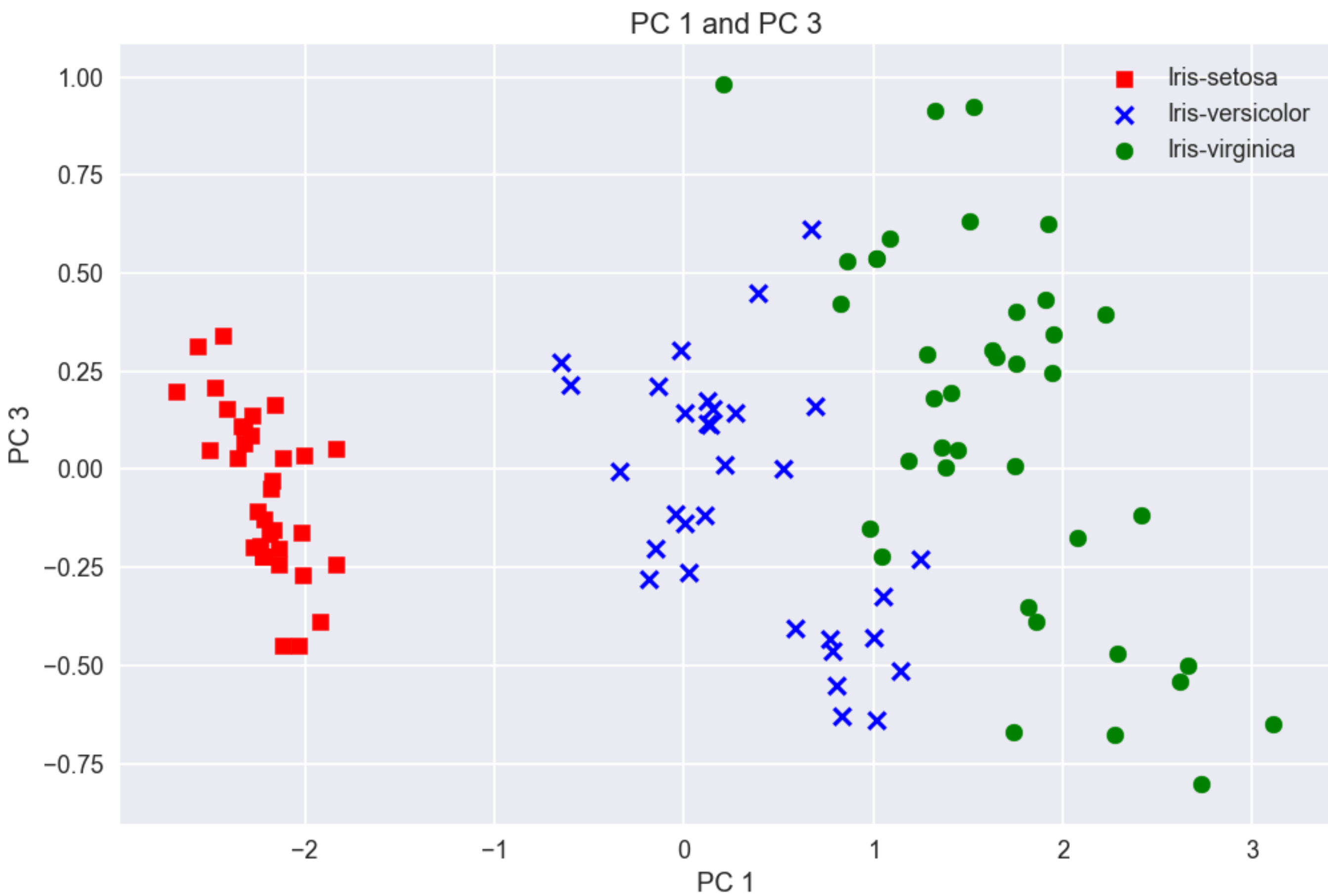
```
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:, i])
               for i in range(len(eigen_vals))]
print(eigen_pairs)
W = np.hstack((eigen_pairs[0][1][:, np.newaxis],
               eigen_pairs[1][1][:, np.newaxis], eigen_pairs[2][1][:, np.newaxis]))
print('Matrix W:\n', W)
print("X_train_std[0].dot(W) ",X_train_std[0].dot(W))

Output:
Matrix W:
 [[  5.35500399e-01  -3.25611548e-01  -7.32041268e-01]
 [ -2.04195389e-01  -9.44913832e-01   2.30263378e-01]
 [  5.86174262e-01   9.09058855e-04   1.37061857e-01]
 [  5.72663340e-01  -3.33787741e-02   6.26345277e-01]]
X_train_std[0].dot(W)  [-0.33664981  2.59032124 -0.00593814]
```

5) Project the samples onto the new feature subspace, and plot the projected data using the transformation matrix W (c.f. block [14] of the nbviewer of Chapter 5)

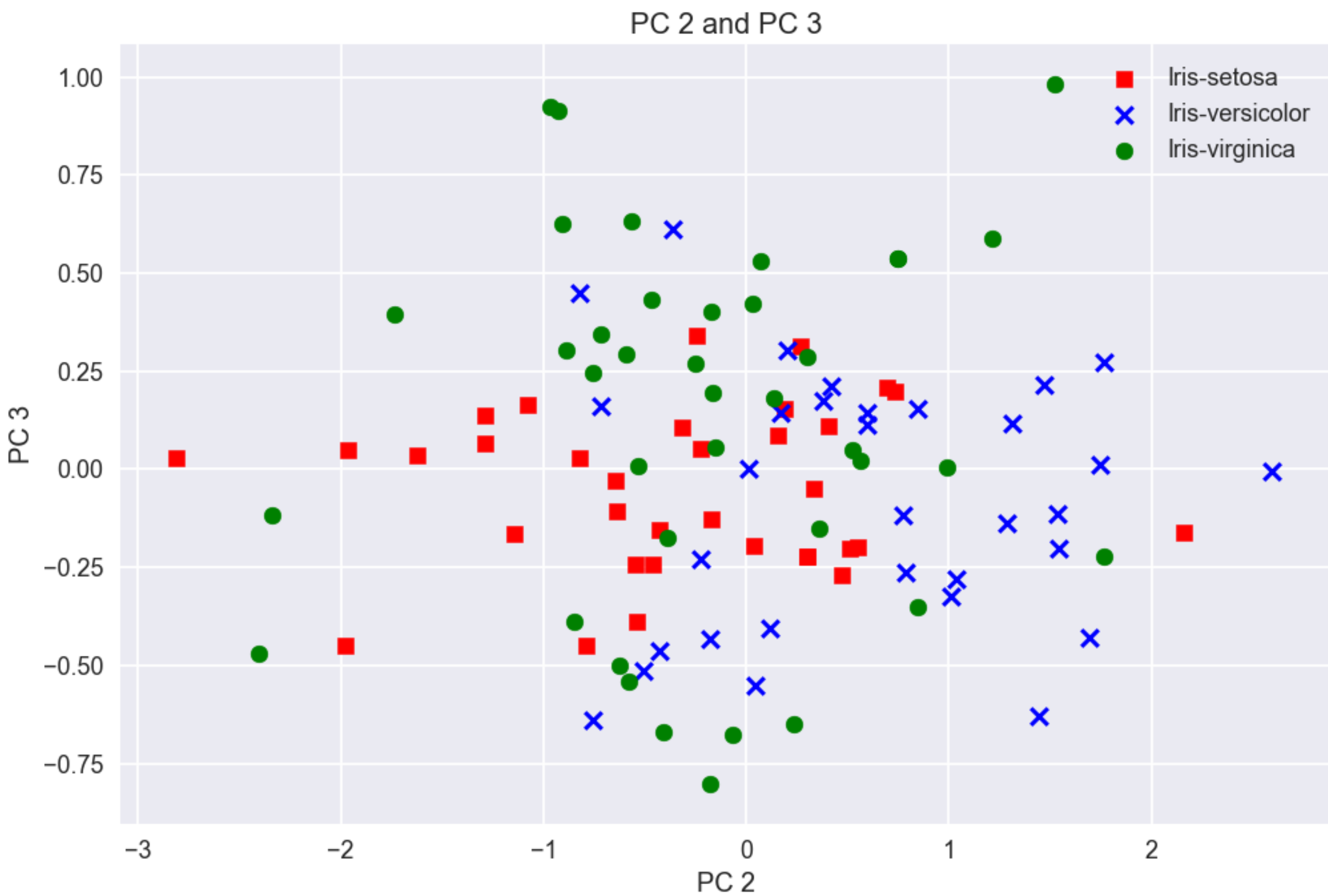
```
X_train_pca = X_train_std.dot(W)
print("X_train_pca ",X_train_pca)

colors = ['r', 'b', 'g']
markers = ['s', 'x', 'o']
#PCA 1 and PCA2
for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_pca[y_train == l, 0],
                X_train_pca[y_train == l, 1],
                c=c, label=l, marker=m)
plt.title('PC 1 and PC 2')
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='upper right')
plt.tight_layout()
plt.show()
```



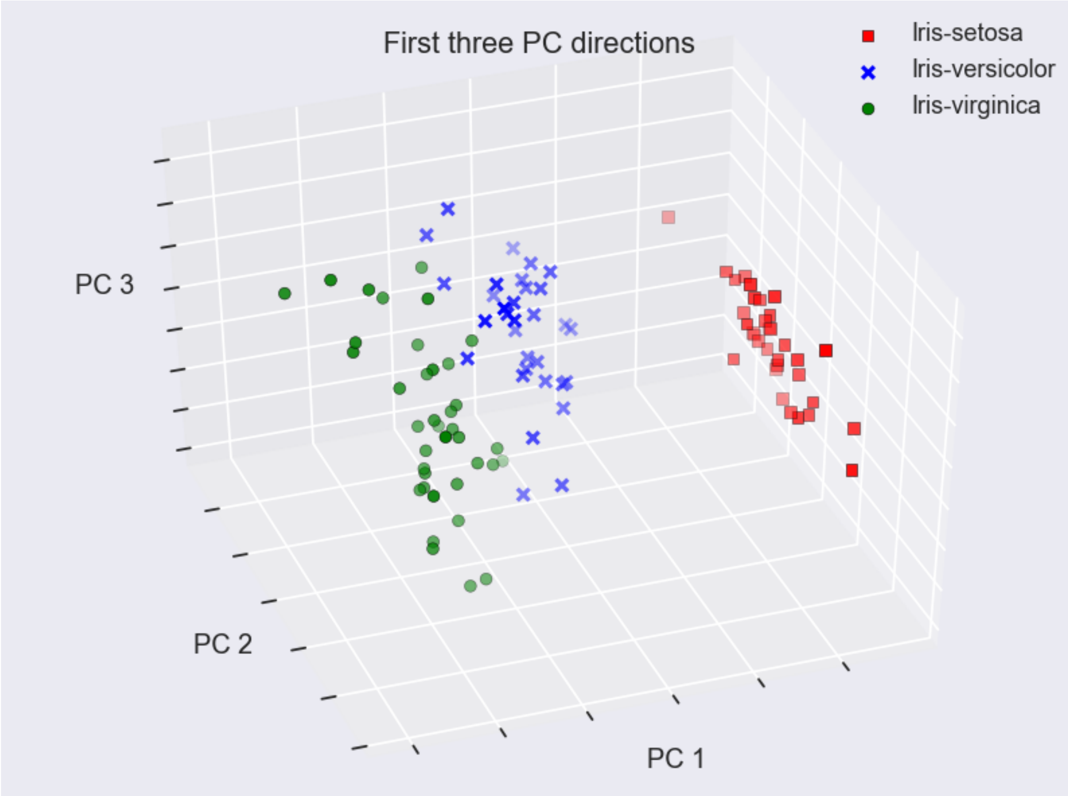

```
#PCA 1 and PCA 3
for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_pca[y_train == l, 0],
                X_train_pca[y_train == l, 2],
                c=c, label=l, marker=m)
plt.title('PC 1 and PC 3')
plt.xlabel('PC 1')
plt.ylabel('PC 3')
plt.legend(loc='upper right')
plt.tight_layout()
plt.show()
```




```
#PCA 2 and PCA 3
for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_pca[y_train == l, 1],
                X_train_pca[y_train == l, 2],
                c=c, label=l, marker=m)
plt.title('PC 2 and PC 3')
plt.xlabel('PC 2')
plt.ylabel('PC 3')
plt.legend(loc='upper right')
plt.tight_layout()
plt.show()
```




```
#PC1, PC2 and PC3
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(1, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)
for l, c, m in zip(np.unique(y_train), colors, markers):
    ax.scatter(X_train_pca[y_train == l, 0], X_train_pca[y_train == l, 1], X_train_pca[y_train == l, 2],c=c,   label=l,marker=m, edgecolor='k'
, s=40)

ax.set_title("First three PC directions")
ax.set_xlabel("PC 1")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("PC 2")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("PC 3")
ax.w_zaxis.set_ticklabels([])
plt.legend(loc='upper right')

plt.show()
```



II/ LDA for Iris:

1) Load (all features, using Pandas), standardize this d-dimension dataset (d is number of features) and Split the Iris dataset to training and test sets with ratio 70% and 30%, respectively.

```
feature_dict = {i:label for i,label in zip(
            range(4),
              ('sepal length in cm',
              'sepal width in cm',
              'petal length in cm',
              'petal width in cm', ))}

print(feature_dict)

# reading the CSV file directly from the UCI machine learning repository
df = pd.io.parsers.read_csv(
    filepath_or_buffer='https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',
    header=None,
    sep=',',
    )

df.columns = [l for i,l in sorted(feature_dict.items())] + ['class label']
df.dropna(how="all", inplace=True) # to drop the empty line at file-end

print(df.tail())

# convert pandas DataFrame to simple numpy arrays
X = df[['sepal length in cm','sepal width in cm','petal length in cm','petal width in cm']].values
# X = df[[0,1,2,3]].values
y = df['class label'].values
# convert class labels from strings to integers
enc = LabelEncoder()
label_encoder = enc.fit(y)
y = label_encoder.transform(y)

print("X: ",X)
print("y: ",y)

X_train, X_test, y_train, y_test =  train_test_split(X, y, test_size=0.3, random_state=0)

sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)

print("X_train_std: ",X_train_std)
print("X_test_std: ",X_test_std)

Output:
{0: 'sepal length in cm', 1: 'sepal width in cm', 2: 'petal length in cm', 3: 'petal width in cm'}
     sepal length in cm  sepal width in cm  petal length in cm  \
```

```
145              6.7            3.0             5.2
146              6.3            2.5             5.0
147              6.5            3.0             5.2
148              6.2            3.4             5.4
149              5.9            3.0             5.1

     petal width in cm     class label
145               2.3  Iris-virginica
146               1.9  Iris-virginica
147               2.0  Iris-virginica
148               2.3  Iris-virginica
149               1.8  Iris-virginica
```

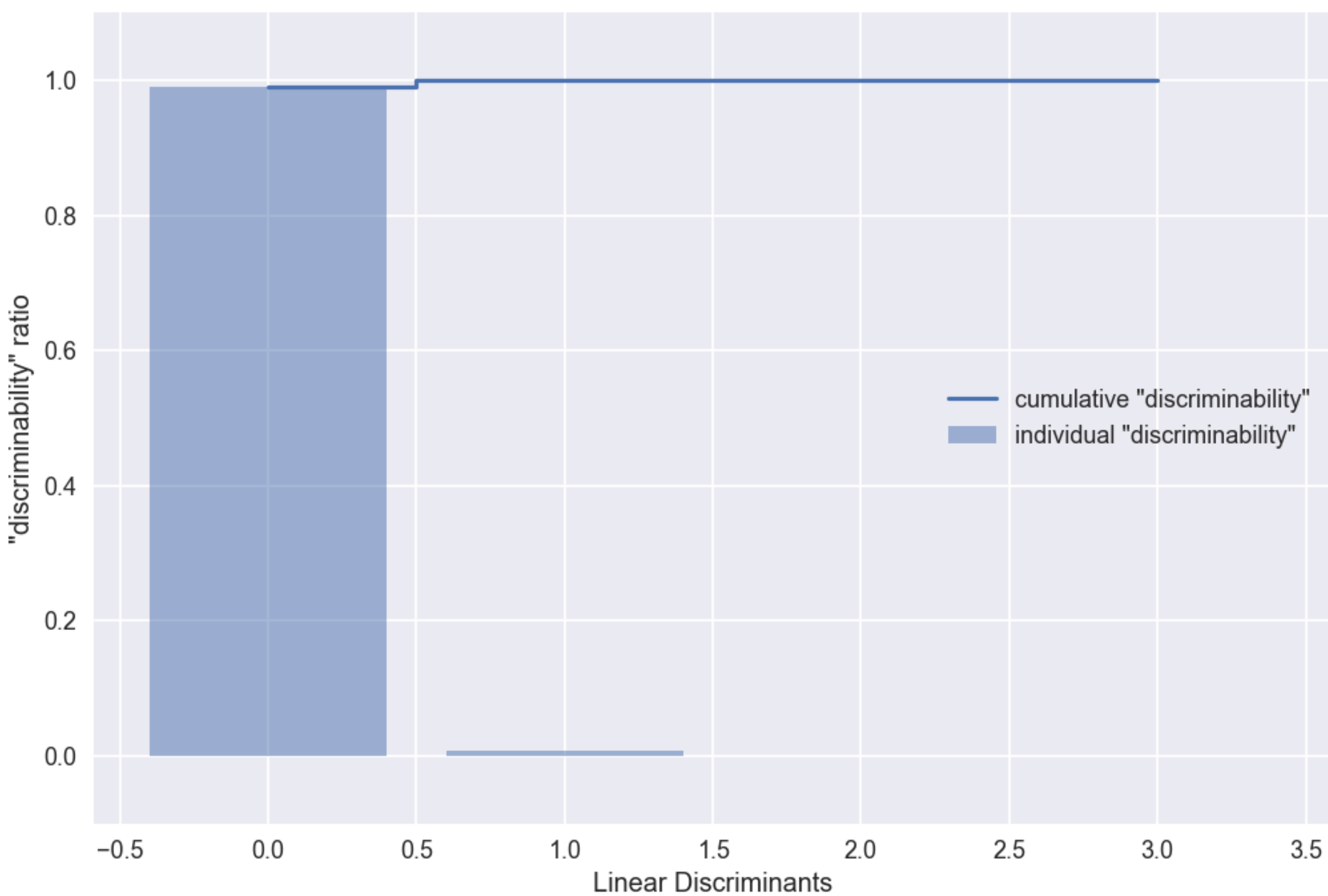
2) Write your own function to calculate matrix S−1WSB instead of covariance matrix.. Then compute the eigenvalues and eigenvectors of this matrix.

```
np.set_printoptions(precision=4)
mean_vecs = []
for label in range(0, 3):
    mean_vecs.append(np.mean(X_train_std[y_train == label], axis=0))
    print('MV %s: %s\n' % (label, mean_vecs[label - 1]))
# Compute the within-class scatter matrix:
d = 4  # number of features
S_W = np.zeros((d, d))
for label, mv in zip(range(0, 3), mean_vecs):
    class_scatter = np.zeros((d, d))  # scatter matrix for each class
    for row in X_train_std[y_train == label]:
        row, mv = row.reshape(d, 1), mv.reshape(d, 1)  # make column vectors
        class_scatter += (row - mv).dot((row - mv).T)
    S_W += class_scatter                             # sum class scatter matrices
print('Within-class scatter matrix: %sx%s' % (S_W.shape[0], S_W.shape[1]))
print('Class label distribution: %s' % np.bincount(y_train)[0:])

# Compute the between-class scatter matrix:
mean_overall = np.mean(X_train_std, axis=0)
d = 4  # number of features
S_B = np.zeros((d, d))
for i, mean_vec in enumerate(mean_vecs):
    n = X_train[y_train == i + 1, :].shape[0]
    mean_vec = mean_vec.reshape(d, 1)  # make column vector
    mean_overall = mean_overall.reshape(d, 1)  # make column vector
    S_B += n * (mean_vec - mean_overall).dot((mean_vec - mean_overall).T)
print('Between-class scatter matrix: %sx%s' % (S_B.shape[0], S_B.shape[1]))
# Solve the generalized eigenvalue problem for the matrix S−1WSBSW−1SB:
eigen_vals, eigen_vecs = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))
print('\nEigenvalues \n%s' % eigen_vals)
print('\neigen_vecs \n%s' % eigen_vecs)
# Sort eigenvectors in decreasing order of the eigenvalues:
# Make a list of (eigenvalue, eigenvector) tuples
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:, i])
               for i in range(len(eigen_vals))]
# Sort the (eigenvalue, eigenvector) tuples from high to low
eigen_pairs = sorted(eigen_pairs, key=lambda k: k[0], reverse=True)
# Visually confirm that the list is correctly sorted by decreasing eigenvalues
print("eigen_pairs: ",eigen_pairs)
print('Eigenvalues in decreasing order:\n')
for eigen_val in eigen_pairs:
    print("eigen_val[0]: ",eigen_val[0])


Output:
MV 0: [-1.0304  0.7685 -1.3228 -1.2823]

MV 1: [-1.0304  0.7685 -1.3228 -1.2823]

MV 2: [ 0.0327 -0.6568  0.2051  0.1023]

Within-class scatter matrix: 4x4
Class label distribution: [34 32 39]
Between-class scatter matrix: 4x4

Eigenvalues
[  2.0112e+01   1.8037e-01  -5.9862e-15   1.8871e-15]

eigen_vecs
[[-0.221   0.0279 -0.6273  0.2163]
 [-0.1062  0.2991  0.1762  0.0844]
 [ 0.8937 -0.558   0.7424  0.6128]
 [ 0.3758  0.7735 -0.1561 -0.7554]]
eigen_pairs:  [(20.111659365444382, array([-0.221 , -0.1062,  0.8937,  0.3758])), (0.18037319649086753, array([ 0.0279,  0.2991, -0.558 ,  0.7
735])), (5.986167287094797e-15, array([-0.6273,  0.1762,  0.7424, -0.1561])), (1.8870703522524938e-15, array([ 0.2163,  0.0844,  0.6128, -0.75
54]))]
Eigenvalues in decreasing order:

eigen_val[0]:  20.1116593654
eigen_val[0]:  0.180373196491
eigen_val[0]:  5.98616728709e-15
```

```
eigen_val[0]:  1.88707035225e-15
cum_discr:  [ 0.9911  1.      1.      1.    ]
discr:  [0.99111113211846413, 0.0088888678815359327, 9.2995630008614786e-17, -2.950008713431661e-16]
```

3) Plot the cumulative variance ratio (c.f. block [11] of the nbviewer of Chapter 5).

```
tot = sum(eigen_vals.real)
discr = [(i / tot) for i in sorted(eigen_vals.real, reverse=True)]
cum_discr = np.cumsum(discr)
print("cum_discr: ",cum_discr)
print("discr: ",discr)
plt.bar(range(0, 4), discr, alpha=0.5, align='center',
        label='individual "discriminability"')
plt.step(range(0, 4), cum_discr, where='mid',label='cumulative "discriminability"')
plt.ylabel('"discriminability" ratio')
plt.xlabel('Linear Discriminants')
plt.ylim([-0.1, 1.1])
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

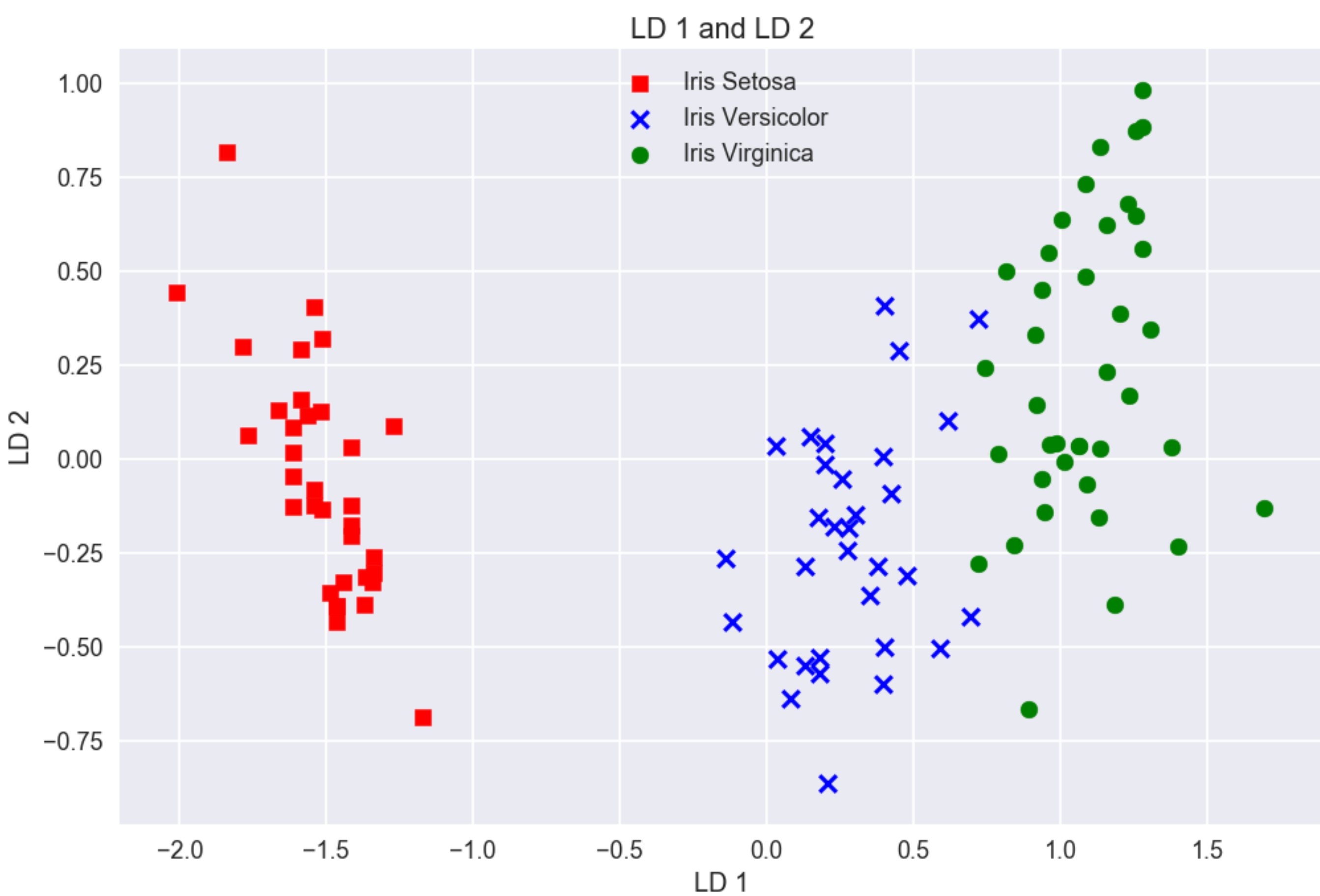


4) Choose the k=3 eigenvectors that correspond to the k largest eigenvalues to construct a d×k-dimensional transformation matrix W ; the eigenvectors are the columns of this matrix

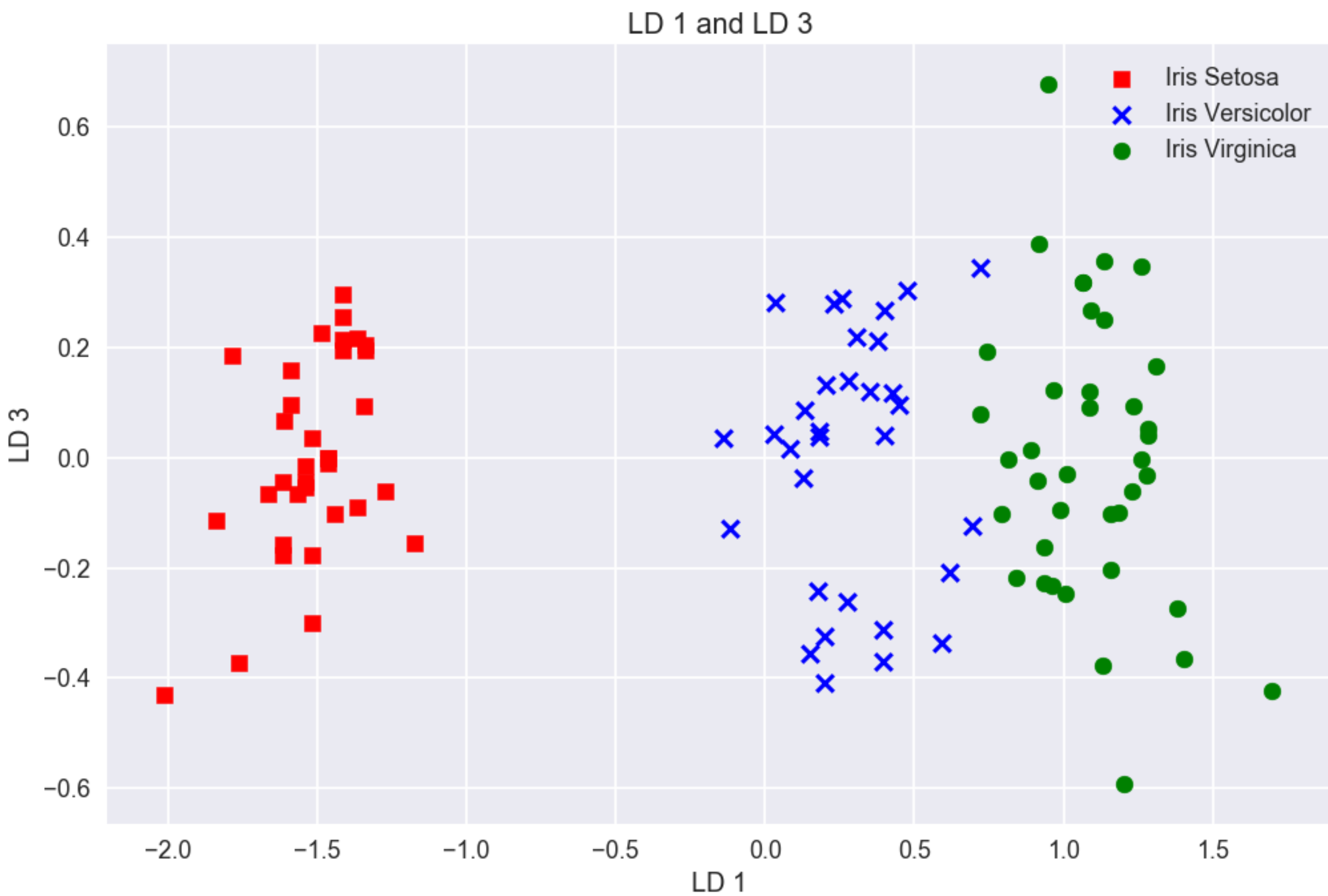
```
W = np.hstack((eigen_pairs[0][1][:, np.newaxis],
               eigen_pairs[1][1][:, np.newaxis], eigen_pairs[2][1][:, np.newaxis]))
print('Matrix W:\n', W)
print("X_train_std[0].dot(W) ",X_train_std[0].dot(W))
X_train_lda = X_train_std.dot(W)
print("X_train_pca ",X_train_lda)

Output:
Matrix W:
 [[-0.221   0.0279 -0.6273]
 [-0.1062  0.2991  0.1762]
 [ 0.8937 -0.558   0.7424]
 [ 0.3758  0.7735 -0.1561]]
X_train_std[0].dot(W)  [ 0.2059 -0.8635  0.1327]
```

5) Project the samples onto the new feature subspace, and plot the projected data using the transformation matrix W (c.f. block [14] of the nbviewer of Chapter 5)

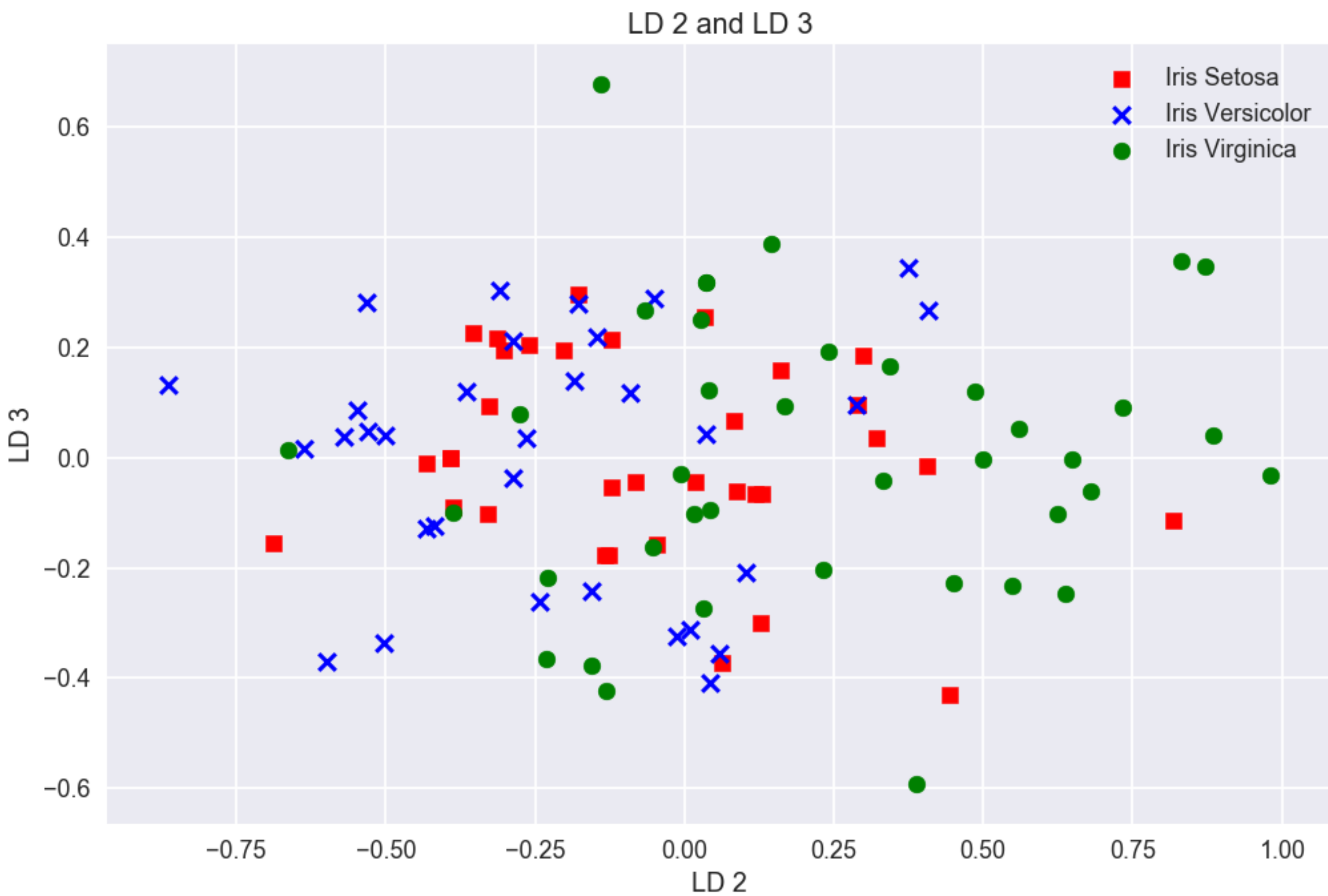
```
colors = ['r', 'b', 'g']
markers = ['s', 'x', 'o']
label_dict = {0: 'Iris Setosa', 1: 'Iris Versicolor', 2:'Iris Virginica'}
#LD 1 and LD 2
for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_lda[y_train == l, 0],
                X_train_lda[y_train == l, 1],
                c=c, label=label_dict[l], marker=m)
    print(l)
    print(label_dict[l])
plt.title('LD 1 and LD 2')
plt.xlabel('LD 1')
plt.ylabel('LD 2')
plt.legend(loc='upper center')
plt.tight_layout()
plt.show()
```



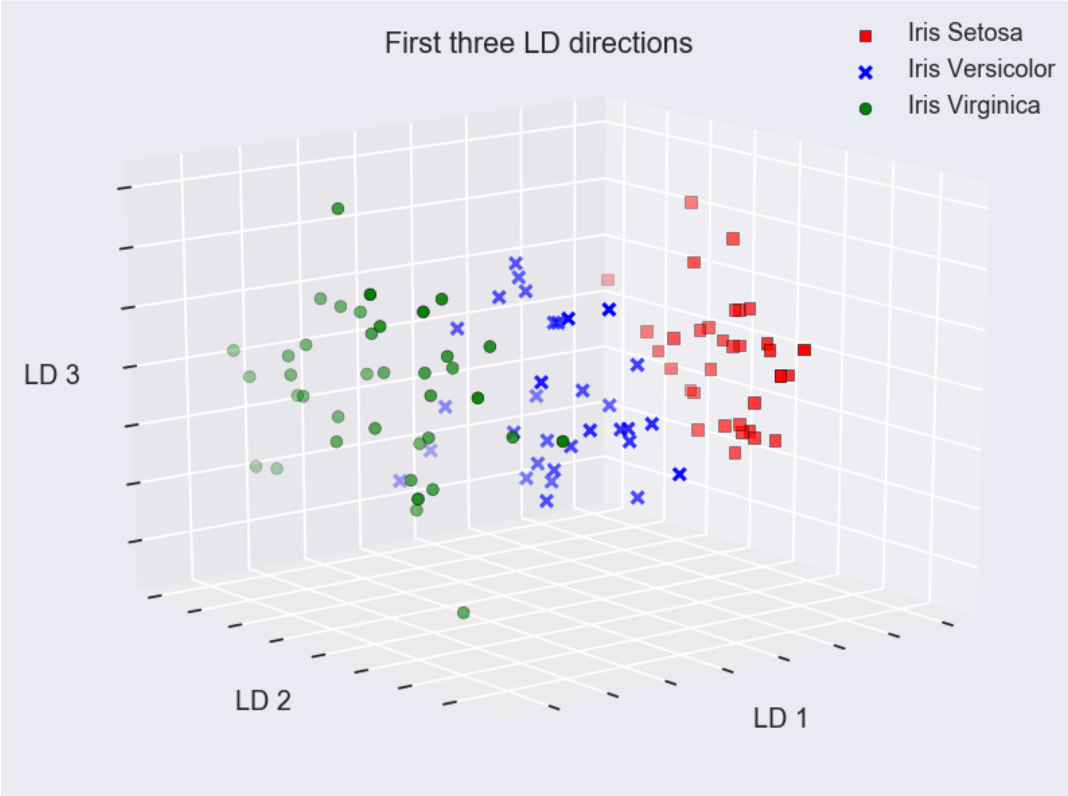

```
#LD 1 and LD 3
for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_lda[y_train == l, 0],
                X_train_lda[y_train == l, 2],
                c=c, label=label_dict[l], marker=m)
plt.title('LD 1 and LD 3')
plt.xlabel('LD 1')
plt.ylabel('LD 3')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```




```
#LD 2 and LD 3
for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_lda[y_train == l, 1],
                X_train_lda[y_train == l, 2],
                c=c, label=label_dict[l], marker=m)
plt.title('LD 2 and LD 3')
plt.xlabel('LD 2')
plt.ylabel('LD 3')
plt.legend(loc='upper right')
plt.tight_layout()
plt.show()
```




```
#LD1, LD2 and LD3
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(1, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)
for l, c, m in zip(np.unique(y_train), colors, markers):
    ax.scatter(X_train_lda[y_train == l, 0], X_train_lda[y_train == l, 1], X_train_lda[y_train == l, 2],c=c,  label=label_dict[l], marker=m, e
dgecolor='k', s=40)
    print('%s' %label_dict[l])
ax.set_title("First three LD directions")
ax.set_xlabel("LD 1")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("LD 2")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("LD 3")
ax.w_zaxis.set_ticklabels([])
ax.legend(loc='upper right')
plt.show()
```



hw2

~ An instructor (Nguyen H. Tran) thinks this is a good note ~

This private post is only visible to Instructors and MD SHIRAJUM MUNIR

Updated 2 years ago by MD SHIRAJUM MUNIR

followup discussions *for lingering questions and comments*