```python
from numpy.random import uniform
from numpy.random import randn
import random
import time

import matplotlib.pyplot as plt

from scipy.linalg import eig
from scipy.linalg import sqrtm
from numpy.linalg import inv
from numpy.linalg import svd

from utils import create_one_hot_label
from utils import subtract_mean_from_data
from utils import compute_covariance_matrix

import numpy as np
import numpy.linalg as LA

import sys
from numpy.linalg import svd


class Project2D():

    '''
    Class to draw projection on 2D scatter space
    '''

    def __init__(self,projection, clss_labels):

        self.proj = projection
        self.clss_labels = clss_labels


    def project_data(self,X,Y,white=None):

        '''
        Takes list of state space and class labels
        State space should be 2D
        Labels shoud be int
        '''

        p_a = []
        p_b = []
        p_c = []

        # Project all Data
        proj = np.matmul(self.proj,white)

        X_P = np.matmul(proj,np.array(X).T)

        for i in range(len(Y)):

            if Y[i] == 0:
                p_a.append(X_P[:,i])
            elif Y[i] == 1:
                p_b.append(X_P[:,i])
            else:
                p_c.append(X_P[:,i])

        p_a = np.array(p_a)
        p_b = np.array(p_b)
        p_c = np.array(p_c)

        plt.scatter(p_a[:,0],p_a[:,1],label = 'apple')
        plt.scatter(p_b[:,0],p_b[:,1],label = 'banana')
        plt.scatter(p_c[:,0],p_c[:,1],label = 'eggplant')

        plt.legend()

        plt.show()


class Projections():

    def __init__(self,dim_x,classes):

        '''
        dim_x: the dimension of the state space x
        classes: The list of class labels
        '''

        self.d_x = dim_x
        self.NUM_CLASSES = len(classes)


    def get_random_proj(self):
        '''
        Return A which is size 2 by 729
        '''
        A = np.zeros( (2, 729), "float" )
        for i in range(A.shape[0]):
            for j in range(A.shape[1]):
                A[i, j] = random.gauss(0, 1)
        return A


    def pca_projection(self,X,Y):

        '''
        Return U_2^T
        '''
        X_demeaned, _ = subtract_mean_from_data(X, X)
        Sigma_XX = compute_covariance_matrix(X_demeaned, X_demeaned)
        U, L, V = svd(Sigma_XX)
        U2 = U[:, [0, 1]]
        return U2.T


    def cca_projection(self,X,Y,k=2):

        '''
        Return U_K^T, \Simgma_{XX}^{-1/2}
        '''

        ###SCALE AN IDENTITY MATRIX BY THIS TERM AND ADD TO COMPUTED COVARIANCE MATRIX TO PREVENT IT BEING SINGULAR ###
        reg = 1e-5

        y_one_hot = create_one_hot_label(Y, self.NUM_CLASSES)
        X, y_one_hot = subtract_mean_from_data(X, y_one_hot)
        Sigma_XX = compute_covariance_matrix(X, X)
        Sigma_XX += reg * np.eye(Sigma_XX.shape[0])
        Sigma_XY = compute_covariance_matrix(X, y_one_hot)
        Sigma_YY = compute_covariance_matrix(y_one_hot, y_one_hot)
        Sigma_YY += reg * np.eye(Sigma_YY.shape[0])
        Sigma_XX_inv_sqrt = inv(sqrtm(Sigma_XX))
        triple_Sigma = Sigma_XX_inv_sqrt.dot(Sigma_XY).dot( inv(sqrtm(Sigma_YY)) )
        U, L, V = svd(triple_Sigma)
        Uk = U[:, list(range( min(k, U.shape[1]) ))]
        return Uk.T, Sigma_XX_inv_sqrt


    def project(self,proj,white,X):
        '''
        proj, numpy matrix to perform projection
        whit, numpy matrix to perform whitenting
        X, list of states
        '''

        proj = np.matmul(proj,white)

        X_P = np.matmul(proj,np.array(X).T)

        return list(X_P.T)


if __name__ == "__main__":

    X = list(np.load('little_x_train.npy'))
    Y = list(np.load('little_y_train.npy'))

    CLASS_LABELS = ['apple','banana','eggplant']

    feat_dim = max(X[0].shape)
    projections = Projections(feat_dim,CLASS_LABELS)

    rand_proj = projections.get_random_proj()
    # Show Random 2D Projection
    proj2D_viz = Project2D(rand_proj,CLASS_LABELS)
    proj2D_viz.project_data(X,Y, white = np.eye(feat_dim))

    #PCA Projection
    pca_proj = projections.pca_projection(X,Y)
    #Show PCA 2D Projection
    proj2D_viz = Project2D(pca_proj,CLASS_LABELS)
    proj2D_viz.project_data(X,Y, white = np.eye(feat_dim))

    #CCA Projection
    cca_proj,white_cov = projections.cca_projection(X,Y)
    #Show CCA 2D Projection
    proj2D_viz = Project2D(cca_proj,CLASS_LABELS)
    proj2D_viz.project_data(X,Y,white = white_cov)


###############################


from numpy.random import uniform
import random
import time

import numpy as np
import numpy.linalg as LA

import sys

from sklearn.linear_model import Ridge

from utils import create_one_hot_label


class Ridge_Model():

    def __init__(self,class_labels):

        ###RIDGE HYPERPARAMETER
        self.lmda = 1.0
        self.class_labels = class_labels
        self.ridge_model = Ridge(self.lmda)


    def train_model(self,X,Y):
        ''''
        FILL IN CODE TO TRAIN MODEL
        MAKE SURE TO ADD HYPERPARAMTER TO MODEL

        '''

        X = np.array(X)
        y_one_hot = create_one_hot_label(Y, len(self.class_labels))
        self.ridge_model.fit(X, y_one_hot)


    def eval(self,x):
        ''''
        Fill in code to evaluate model and return a prediction
        Prediction should be an integer specifying a class
        '''
        x = x.reshape(1, -1)
        y = self.ridge_model.predict(x)
        return np.argmax(y)


##########################


```

```
import random
import time

import glob
import os
import pickle
import matplotlib.pyplot as plt

import numpy as np
import numpy.linalg as LA

import sys
from numpy.linalg import inv
from numpy.linalg import det
from sklearn.svm import LinearSVC
from projection import Project2D, Projections

from utils import subtract_mean_from_data
from utils import compute_covariance_matrix


class LDA_Model():

    def __init__(self,class_labels):

        ###SCALE AN IDENTITY MATRIX BY THIS TERM AND ADD TO COMPUTED COVARIANCE MATRIX TO PREVENT IT BEING SINGULAR ###
        self.reg_cov = 0.001
        self.NUM_CLASSES = len(class_labels)


    def train_model(self,X,Y):
        ''''
        FILL IN CODE TO TRAIN MODEL
        MAKE SURE TO ADD HYPERPARAMTER TO MODEL

        '''
        ps = [ [] for j in range(self.NUM_CLASSES) ]
        for i, y in enumerate(Y):
            ps[y].append(X[i])

        self.mean_list = []
        for lst in ps:
            self.mean_list.append( np.mean(np.array(lst), axis=0) )

        Sigma_XX = compute_covariance_matrix(X, X)
        Sigma_XX += self.reg_cov * np.identity(Sigma_XX.shape[0])
        self.Sigma_inv = inv(Sigma_XX)


    def eval(self,x):
        ''''
        Fill in code to evaluate model and return a prediction
        Prediction should be an integer specifying a class
        '''
        x = x.reshape(1, -1)
        y = {}
        for i in range(self.NUM_CLASSES):
            x_demeaned = x - self.mean_list[i]
            f = - x_demeaned.dot(self.Sigma_inv).dot(x_demeaned.T)
            y[i] = f.flatten()[0]
        return max(y, key=lambda x: y[x])


############################


import random
import time

import numpy as np
import numpy.linalg as LA

from numpy.linalg import inv
from numpy.linalg import det

from projection import Project2D, Projections

from utils import subtract_mean_from_data
from utils import compute_covariance_matrix


class QDA_Model():

    def __init__(self,class_labels):

        ###SCALE AN IDENTITY MATRIX BY THIS TERM AND ADD TO COMPUTED COVARIANCE MATRIX TO PREVENT IT BEING SINGULAR ###
        self.reg_cov = 0.01
        self.NUM_CLASSES = len(class_labels)


    def train_model(self,X,Y):
        ''''
        FILL IN CODE TO TRAIN MODEL
        MAKE SURE TO ADD HYPERPARAMTER TO MODEL

        '''
        ps = [ [] for j in range(self.NUM_CLASSES) ]
        for i, y in enumerate(Y):
            ps[y].append(X[i])

        self.mean_list = []
        self.Sigma_inv_list = []
        for lst in ps:
            self.mean_list.append( np.mean(np.array(lst), axis=0) )
            Sigma_XX = compute_covariance_matrix(lst, lst)
            Sigma_XX += self.reg_cov * np.identity(Sigma_XX.shape[0])
            self.Sigma_inv_list.append(inv(Sigma_XX))


    def eval(self,x):
        ''''
        Fill in code to evaluate model and return a prediction
        Prediction should be an integer specifying a class
        '''
        x = x.reshape(1, -1)
        y = {}
        for i in range(self.NUM_CLASSES):
            x_demeaned = x - self.mean_list[i]
            f = - x_demeaned.dot(self.Sigma_inv_list[i]).dot(x_demeaned.T)
            y[i] = f.flatten()[0]
        return max(y, key=lambda x: y[x])


#########################


from numpy.random import uniform
import random
import time

import matplotlib.pyplot as plt

import numpy as np
import numpy.linalg as LA

import sys

from sklearn.svm import LinearSVC
from projection import Project2D, Projections

from utils import create_one_hot_label


class SVM_Model():

    def __init__(self,class_labels,projection=None):

        ###SLACK HYPERPARAMETER
        self.C = 1.0
        self.class_labels = class_labels
        self.svc_model = LinearSVC(C=self.C)


    def train_model(self,X,Y):
        ''''
        FILL IN CODE TO TRAIN MODEL
        MAKE SURE TO ADD HYPERPARAMTER TO MODEL

        '''
        X = np.array(X)
        self.svc_model.fit(X, Y)


    def eval(self,x):
        ''''
        Fill in code to evaluate model and return a prediction
        Prediction should be an integer specifying a class
        '''
        x = x.reshape(1, -1)
        y = self.svc_model.predict(x)
        return y[0]

```