```
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}
```

```
115             '''
116
117             ###SCALE AN IDENTITY MATRIX BY THIS TERM AND ADD TO COMPUTED COVARIANCE
                MATRIX TO PREVENT IT BEING SINGULAR ###
118             reg = 1e-5
119
120             y_one_hot = create_one_hot_label(Y, self.NUM_CLASSES)
121             X, y_one_hot = subtract_mean_from_data(X, y_one_hot)
122             Sigma_XX = compute_covariance_matrix(X, X)
123             Sigma_XX += reg * np.eye(Sigma_XX.shape[0])
124             Sigma_XY = compute_covariance_matrix(X, y_one_hot)
125             Sigma_YY = compute_covariance_matrix(y_one_hot, y_one_hot)
126             Sigma_YY += reg * np.eye(Sigma_YY.shape[0])
127             Sigma_XX_inv_sqrt = inv(sqrtm(Sigma_XX))
128             triple_Sigma = Sigma_XX_inv_sqrt.dot(Sigma_XY).dot( inv(sqrtm(Sigma_YY)) )
129             U, L, V = svd(triple_Sigma)
130             Uk = U[:, list(range( min(k, U.shape[1]) ))]
131             return Uk.T, Sigma_XX_inv_sqrt
132
133
134         def project(self,proj,white,X):
135             '''
136             proj, numpy matrix to perform projection
137             whit, numpy matrix to perform whitenting
138             X, list of states
139             '''
140
141             proj = np.matmul(proj,white)
142
143             X_P = np.matmul(proj,np.array(X).T)
144
145             return list(X_P.T)
146
147
148     if __name__ == "__main__":
149
150         X = list(np.load('little_x_train.npy'))
151         Y = list(np.load('little_y_train.npy'))
152
153         CLASS_LABELS = ['apple','banana','eggplant']
154
155         feat_dim = max(X[0].shape)
156         projections = Projections(feat_dim,CLASS_LABELS)
157
158         rand_proj = projections.get_random_proj()
159         # Show Random 2D Projection
160         proj2D_viz = Project2D(rand_proj,CLASS_LABELS)
161         proj2D_viz.project_data(X,Y, white = np.eye(feat_dim))
162
163         #PCA Projection
164         pca_proj = projections.pca_projection(X,Y)
165         #Show PCA 2D Projection
166         proj2D_viz = Project2D(pca_proj,CLASS_LABELS)
167         proj2D_viz.project_data(X,Y, white = np.eye(feat_dim))
168
169         #CCA Projection
170         cca_proj,white_cov = projections.cca_projection(X,Y)
```

```
171         #Show CCA 2D Projection
172         proj2D_viz = Project2D(cca_proj,CLASS_LABELS)
173         proj2D_viz.project_data(X,Y,white = white_cov)
174 
175 
176     ##############################
177 
178 
179     from numpy.random import uniform
180     import random
181     import time
182 
183     import numpy as np
184     import numpy.linalg as LA
185 
186     import sys
187 
188     from sklearn.linear_model import Ridge
189 
190     from utils import create_one_hot_label
191 
192 
193     class Ridge_Model():
194 
195         def __init__(self,class_labels):
196 
197             ###RIDGE HYPERPARAMETER
198             self.lmda = 1.0
199             self.class_labels = class_labels
200             self.ridge_model = Ridge(self.lmda)
201 
202 
203         def train_model(self,X,Y):
204             ''''
205             FILL IN CODE TO TRAIN MODEL
206             MAKE SURE TO ADD HYPERPARAMTER TO MODEL
207 
208             '''
209 
210             X = np.array(X)
211             y_one_hot = create_one_hot_label(Y, len(self.class_labels))
212             self.ridge_model.fit(X, y_one_hot)
213 
214 
215         def eval(self,x):
216             ''''
217             Fill in code to evaluate model and return a prediction
218             Prediction should be an integer specifying a class
219             '''
220             x = x.reshape(1, -1)
221             y = self.ridge_model.predict(x)
222             return np.argmax(y)
223 
224 
225     ##########################
226 
227 
```

```
228 import random
229 import time
230
231 import glob
232 import os
233 import pickle
234 import matplotlib.pyplot as plt
235
236 import numpy as np
237 import numpy.linalg as LA
238
239 import sys
240 from numpy.linalg import inv
241 from numpy.linalg import det
242 from sklearn.svm import LinearSVC
243 from projection import Project2D, Projections
244
245 from utils import subtract_mean_from_data
246 from utils import compute_covariance_matrix
247
248
249 class LDA_Model():
250
251     def __init__(self,class_labels):
252
253         ###SCALE AN IDENTITY MATRIX BY THIS TERM AND ADD TO COMPUTED COVARIANCE
          MATRIX TO PREVENT IT BEING SINGULAR ###
254         self.reg_cov = 0.001
255         self.NUM_CLASSES = len(class_labels)
256
257
258     def train_model(self,X,Y):
259         ''''
260         FILL IN CODE TO TRAIN MODEL
261         MAKE SURE TO ADD HYPERPARAMTER TO MODEL
262
263         '''
264         ps = [ [] for j in range(self.NUM_CLASSES) ]
265         for i, y in enumerate(Y):
266             ps[y].append(X[i])
267
268         self.mean_list = []
269         for lst in ps:
270             self.mean_list.append( np.mean(np.array(lst), axis=0) )
271
272         Sigma_XX = compute_covariance_matrix(X, X)
273         Sigma_XX += self.reg_cov * np.identity(Sigma_XX.shape[0])
274         self.Sigma_inv = inv(Sigma_XX)
275
276
277     def eval(self,x):
278         ''''
279         Fill in code to evaluate model and return a prediction
280         Prediction should be an integer specifying a class
281         '''
282         x = x.reshape(1, -1)
283         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
```

```
340             Prediction should be an integer specifying a class
341             '''
342             x = x.reshape(1, -1)
343             y = {}
344             for i in range(self.NUM_CLASSES):
345                 x_demeaned = x - self.mean_list[i]
346                 f = - x_demeaned.dot(self.Sigma_inv_list[i]).dot(x_demeaned.T)
347                 y[i] = f.flatten()[0]
348             return max(y, key=lambda x: y[x])
349
350
351     ########################
352
353
354     from numpy.random import uniform
355     import random
356     import time
357
358     import matplotlib.pyplot as plt
359
360     import numpy as np
361     import numpy.linalg as LA
362
363     import sys
364
365     from sklearn.svm import LinearSVC
366     from projection import Project2D, Projections
367
368     from utils import create_one_hot_label
369
370
371     class SVM_Model():
372
373         def __init__(self,class_labels,projection=None):
374
375             ###SLACK HYPERPARAMETER
376             self.C = 1.0
377             self.class_labels = class_labels
378             self.svc_model = LinearSVC(C=self.C)
379
380
381         def train_model(self,X,Y):
382             ''''
383             FILL IN CODE TO TRAIN MODEL
384             MAKE SURE TO ADD HYPERPARAMTER TO MODEL
385
386             '''
387             X = np.array(X)
388             self.svc_model.fit(X, Y)
389
390
391         def eval(self,x):
392             ''''
393             Fill in code to evaluate model and return a prediction
394             Prediction should be an integer specifying a class
395             '''
396             x = x.reshape(1, -1)
```

```
397             y = self.svc_model.predict(x)
398             return y[0]
399
```