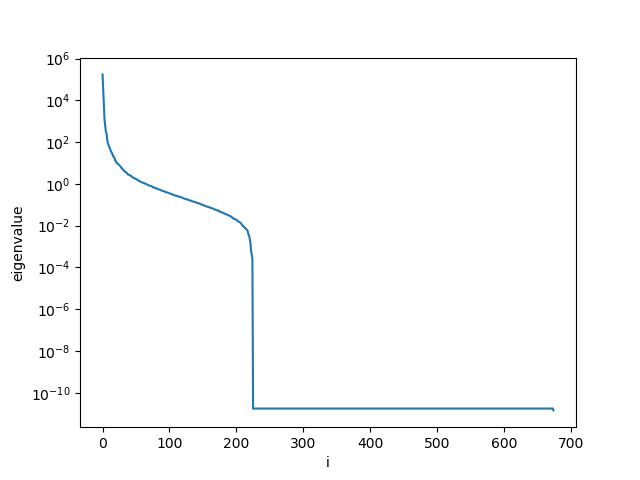
**3b**

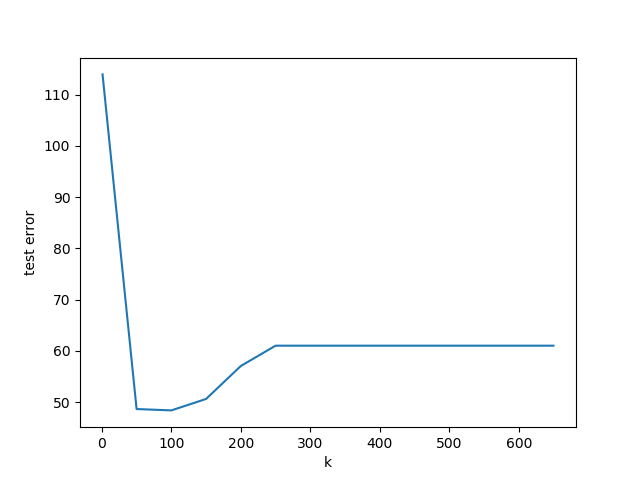


**3c**

Looks like a face if you look carefully. The features(eyes, nose, etc) on the face are the correlated parts between the mooney image and gray scale image.



**3d**



**3e**

We use k=100 since the test error is the smallest









import os

import numpy as np

import cv2

import copy

import glob

import sys

from numpy.random import uniform

import pickle

from scipy.linalg import eig

from scipy.linalg import sqrtm

from numpy.linalg import inv

from numpy.linalg import svd

import numpy.linalg as LA

import matplotlib.pyplot as plt

import IPython

from sklearn.preprocessing import StandardScaler

def standardized(v):

return (v/255.0) \* 2.0 - 1.0

def flatten\_and\_standardize(data):

result = []

for d in data:

d = d.flatten()

d = standardized(d)

result.append(d)

return result

class Mooney(object):

def \_\_init\_\_(self):

self.lmbda = 1e-5

def load\_data(self):

self.x\_train = pickle.load(open('x\_train.p','rb'))

self.y\_train = pickle.load(open('y\_train.p','rb'))

self.x\_test = pickle.load(open('x\_test.p','rb'))

self.y\_test = pickle.load(open('y\_test.p','rb'))

def compute\_covariance\_matrices(self):

# USE STANDARD SCALAR TO DO MEAN SUBTRACTION

ss\_x = StandardScaler(with\_std = False)

ss\_y = StandardScaler(with\_std = False)

num\_data = len(self.x\_train)

x = self.x\_train[0]

y = self.y\_train[0]

x\_f = x.flatten()

y\_f = y.flatten()

x\_f\_dim = x\_f.shape[0]

y\_f\_dim = y\_f.shape[0]

self.x\_dim = x\_f\_dim

self.y\_dim = y\_f\_dim

self.C\_xx = np.zeros([x\_f\_dim,x\_f\_dim])

self.C\_yy = np.zeros([y\_f\_dim,y\_f\_dim])

self.C\_xy = np.zeros([x\_f\_dim,y\_f\_dim])

x\_data = []

y\_data = []

for i in range(num\_data):

x\_image = self.x\_train[i]

y\_image = self.y\_train[i]

# FLATTEN DATA

x\_f = x\_image.flatten()

y\_f = y\_image.flatten()

# STANDARDIZE DATA

x\_f = standardized(x\_f)

y\_f = standardized(y\_f)

x\_data.append(x\_f)

y\_data.append(y\_f)

# SUBTRACT MEAN

ss\_x.fit(x\_data)

x\_data = ss\_x.transform(x\_data)

ss\_y.fit(y\_data)

y\_data = ss\_y.transform(y\_data)

for i in range(num\_data):

x\_f = np.array([x\_data[i]])

y\_f = np.array([y\_data[i]])

#####TO DO:

self.C\_xx += x\_f.T @ x\_f

self.C\_yy += y\_f.T @ y\_f

self.C\_xy += x\_f.T @ y\_f

# DIVIDE BY THE NUMBER OF DATA POINTS

self.C\_xx = 1.0/float(num\_data)\*self.C\_xx

self.C\_yy = 1.0/float(num\_data)\*self.C\_yy

self.C\_xy = 1.0/float(num\_data)\*self.C\_xy

def compute\_projected\_data\_matrix(self,X\_proj):

Y = []

X = []

Y\_test = []

X\_test = []

# LOAD TRAINING DATA

for x in self.x\_train:

x\_f = np.array([x.flatten()])

# STANDARDIZE DATA

x\_f = standardized(x\_f)

# TODO: PROJECT DATA

X.append(x\_f @ X\_proj.T)

Y = flatten\_and\_standardize(self.y\_train)

for x in self.x\_test:

x\_f = np.array([x.flatten()])

# STANDARDIZE DATA

x\_f = standardized(x\_f)

# TODO: PROJECT DATA

X\_test.append(x\_f @ X\_proj.T)

Y\_test = flatten\_and\_standardize(self.y\_test)

# CONVERT TO MATRIX

self.X\_ridge = np.vstack(X)

self.Y\_ridge = np.vstack(Y)

self.X\_test\_ridge = np.vstack(X\_test)

self.Y\_test\_ridge = np.vstack(Y\_test)

def compute\_data\_matrix(self):

X = flatten\_and\_standardize(self.x\_train)

Y = flatten\_and\_standardize(self.y\_train)

X\_test = flatten\_and\_standardize(self.x\_test)

Y\_test = flatten\_and\_standardize(self.y\_test)

# CONVERT TO MATRIX

self.X\_ridge = np.vstack(X)

self.Y\_ridge = np.vstack(Y)

self.X\_test\_ridge = np.vstack(X\_test)

self.Y\_test\_ridge = np.vstack(Y\_test)

def solve\_for\_variance(self):

eigen\_values = np.zeros((675,))

eigen\_vectors = np.zeros((675, 675))

# TODO:

D1 = self.C\_xx+self.lmbda\*np.eye(675)

D1[np.diag\_indices(675)] = 1/(D1.diagonal()\*\*0.5)

D2 = self.C\_yy+self.lmbda\*np.eye(675)

D2[np.diag\_indices(675)] = 1/(D2.diagonal()\*\*0.5)

matrix = D1 @ self.C\_xy @ D2

eigen\_vectors, eigen\_values, dummy = svd(matrix)

return eigen\_values, eigen\_vectors

def project\_data(self, eig\_val,eig\_vec,proj=150):

# TODO: COMPUTE PROJECTION SINGULAR VECTORS

return eig\_vec[:, 0:proj].T

def ridge\_regression(self):

w\_ridge = []

for i in range(self.y\_dim):

# TODO: IMPLEMENT RIDGE REGRESSION

y = self.Y\_ridge[:, i]

w = inv(self.X\_ridge.T @ self.X\_ridge + self.lmbda\*np.eye(self.X\_ridge.shape[1])) @ self.X\_ridge.T @ y

w\_ridge.append(w)

self.w\_ridge = np.vstack(w\_ridge)

def plot\_image(self, vector):

vector = ((vector+1.0)/2.0)\*255.0

vector = np.reshape(vector,(15,15,3))

p = vector.astype("uint8")

p = cv2.resize(p,(100,100))

count = 0

cv2.imwrite('a\_face\_'+str(count)+'.png',p)

def measure\_error(self, X\_ridge, Y\_ridge):

prediction = np.matmul(self.w\_ridge,X\_ridge.T)

evaluation = Y\_ridge.T - prediction

#print(evaluation)

dim,num\_data = evaluation.shape

error = []

for i in range(num\_data):

# COMPUTE L2 NORM for each vector then square

error.append(LA.norm(evaluation[:,i])\*\*2)

# Return average error

return np.mean(error)

def draw\_images(self):

for count, x in enumerate(self.X\_test\_ridge):

prediction = np.matmul(self.w\_ridge,x)

prediction = ((prediction+1.0)/2.0)\*255.0

prediction = np.reshape(prediction,(15,15,3))

p = prediction.astype("uint8")

p = cv2.resize(p,(100,100))

cv2.imwrite('face\_'+str(count)+'.png',p)

for count, x in enumerate(self.x\_test):

x = x.astype("uint8")

x = cv2.resize(x,(100,100))

cv2.imwrite('og\_face\_'+str(count)+'.png',x)

for count, x in enumerate(self.y\_test):

x = x.astype("uint8")

x = cv2.resize(x,(100,100))

cv2.imwrite('gt\_face\_'+str(count)+'.png',x)

if \_\_name\_\_ == '\_\_main\_\_':

mooney = Mooney()

mooney.load\_data()

mooney.compute\_covariance\_matrices()

eig\_val, eig\_vec = mooney.solve\_for\_variance()

plt.semilogy(np.arange(675), eig\_val)

plt.xlabel('i')

plt.ylabel('eigenvalue')

plt.savefig('spectrum.png')

plt.show()

mooney.plot\_image(eig\_vec[:, 0])

proj = [1,50,100,150,200,250,300,350,400,450,500,650]

error\_test = []

#for p in proj:

# X\_proj = mooney.project\_data(eig\_val,eig\_vec, proj=p)

# COMPUTE REGRESSION

# mooney.compute\_projected\_data\_matrix(X\_proj)

# mooney.ridge\_regression()

# training\_error = mooney.measure\_error(mooney.X\_ridge, mooney.Y\_ridge)

# test\_error = mooney.measure\_error(mooney.X\_test\_ridge, mooney.Y\_test\_ridge)

## mooney.draw\_images()

# error\_test.append(test\_error)

#plt.plot(proj,error\_test)

#plt.xlabel('k')

#plt.ylabel('test error')

#plt.savefig('error.png')

#plt.show()

# COMPUTE REGRESSION NO PROJECT

X\_proj = mooney.project\_data(eig\_val,eig\_vec, proj=100)

mooney.compute\_projected\_data\_matrix(X\_proj)

mooney.ridge\_regression()

mooney.draw\_images()

#training\_error = mooney.measure\_error(mooney.X\_ridge, mooney.Y\_ridge)

#test\_error = mooney.measure\_error(mooney.X\_test\_ridge, mooney.Y\_test\_ridge)