CS464 Introduction to Machine Learning

Fall 2022

Homework 2

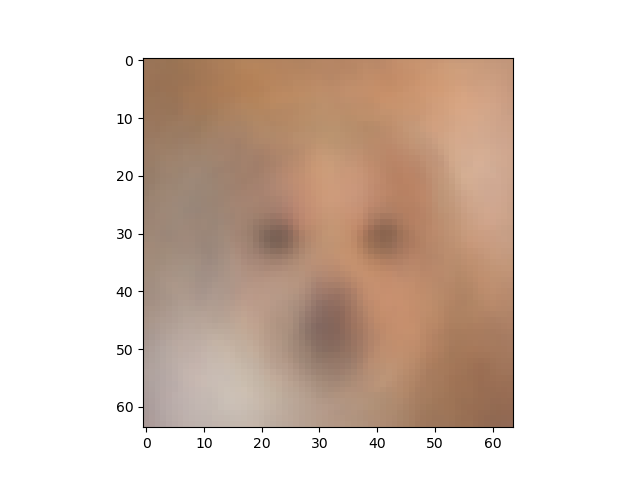
Emre Can Şen-21902516

Q-1.1

First 10 principal components for red, green and blue are computed. Eigen values are computed.

Q-1.2

The found PCA’s are reshaped to 64x64 matrix. Then RGB values of each PCA is stacked to create a 64x64x3 matrix and they are displayed here:



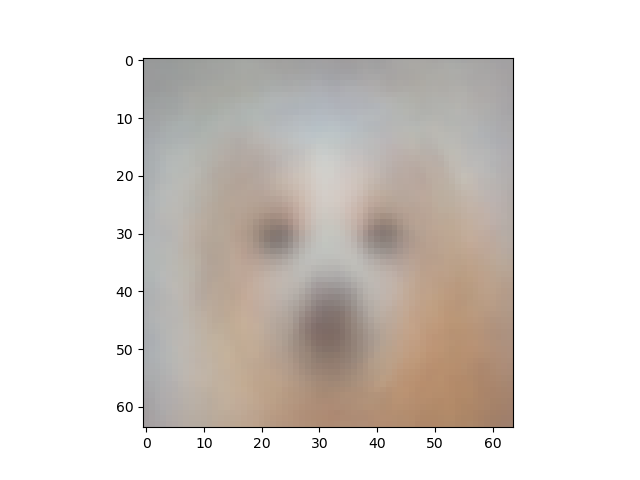
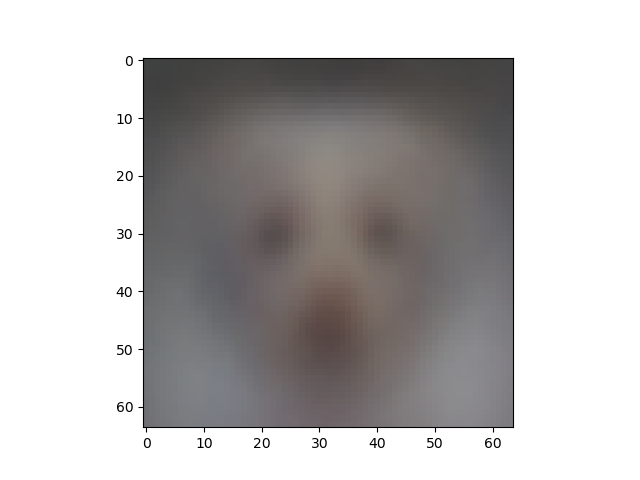


Figure 1- 1st Eigen Image Figure 2- 2nd Eigen Image



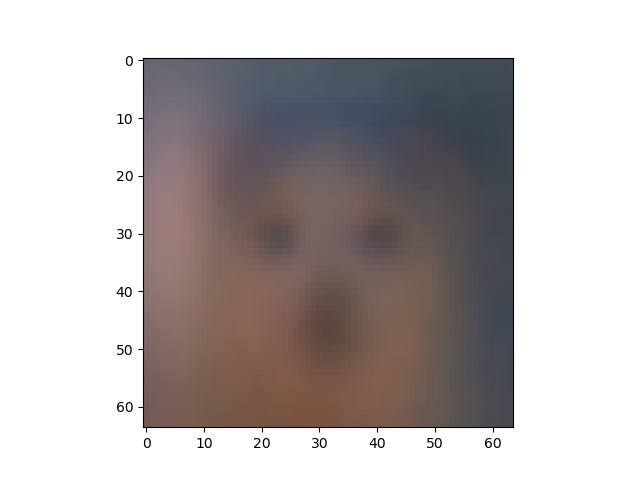
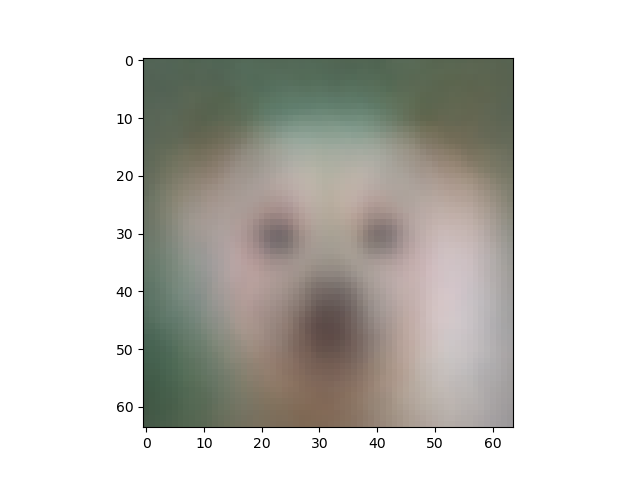


Figure 3- 3rd Eigen Image Figure 4- 4th Eigen Image



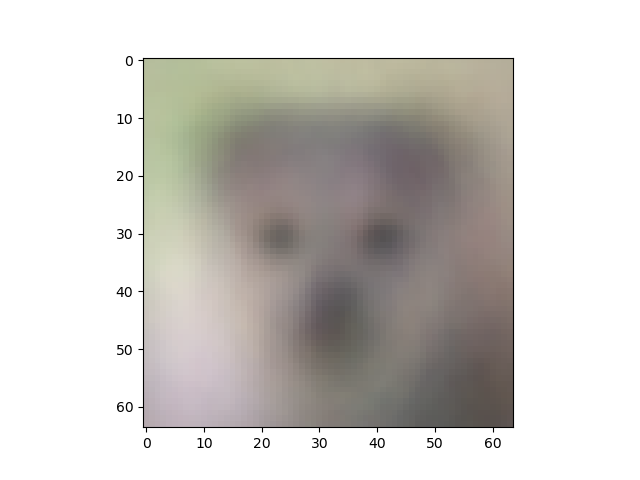
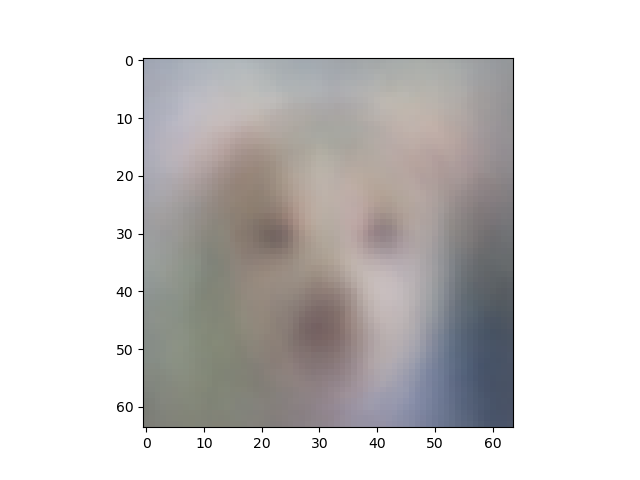


Figure 5- 5th Eigen Image Figure 6- 6th Eigen Image



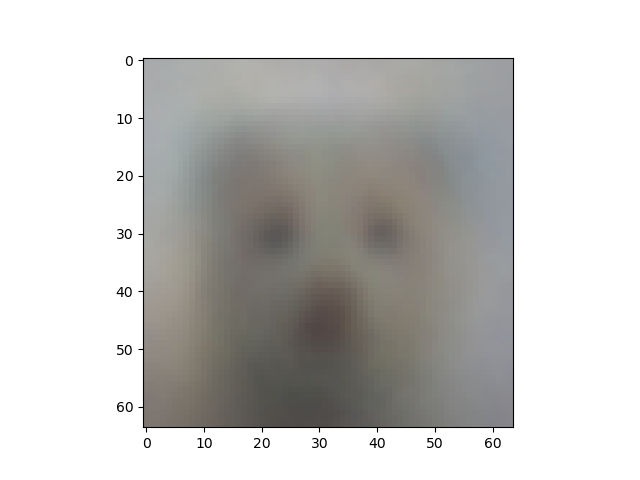
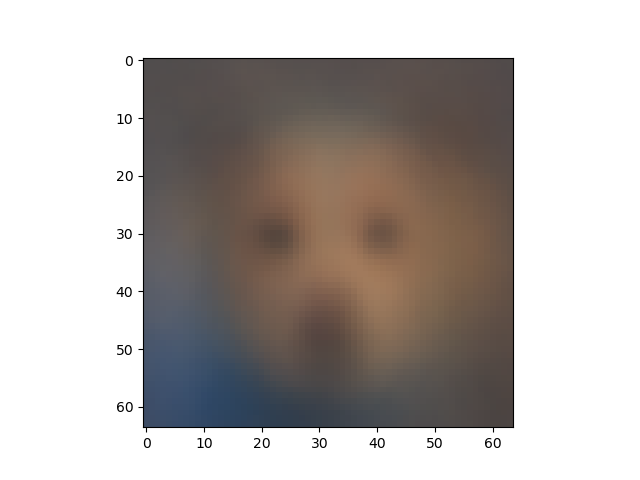


Figure 7 – 7th Eigen Image Figure 8- 8th Eigen Image



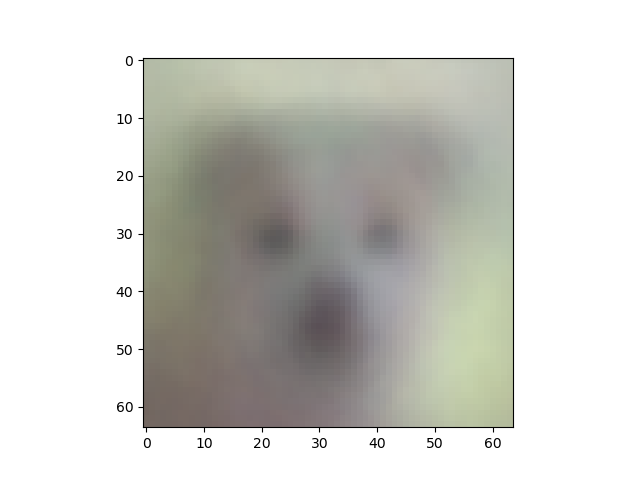
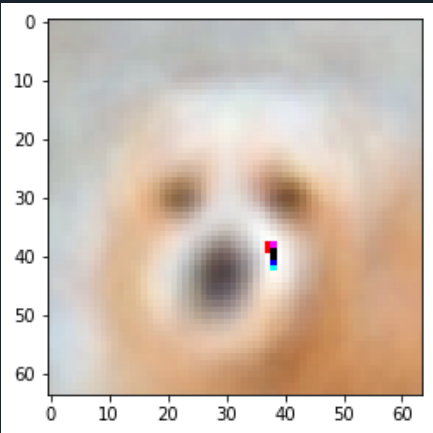


Figure 9 – 9th Eigen Image Figure 10- 10th Eigen Image

These are the most important dimensions that are available on the whole set, projected to first 10 images.

Q-1.3

In order to reconstruct the dog images, it is sufficient to dot product the PCA components with RGB values and then reformat them to fit 64x64 pixel size and draw the image. This dot product results in an array that contains the reconstructions of all images, so the first of the 5239 data was taken to figure. The more principal components used, the more detailed the image is.



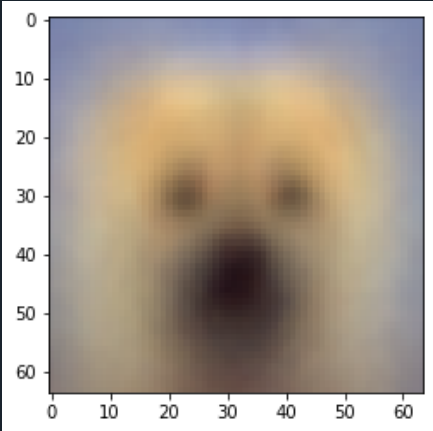
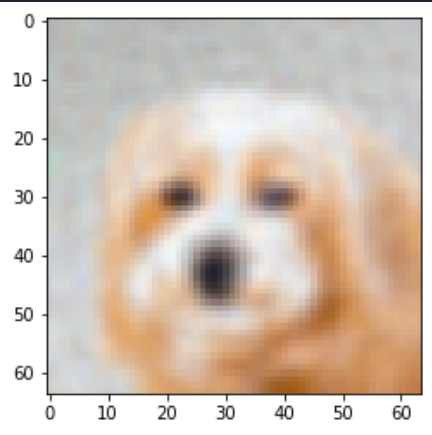


Figure 11- K=1 Image Reconstruction Figure 12- K=50 Image Reconstruction



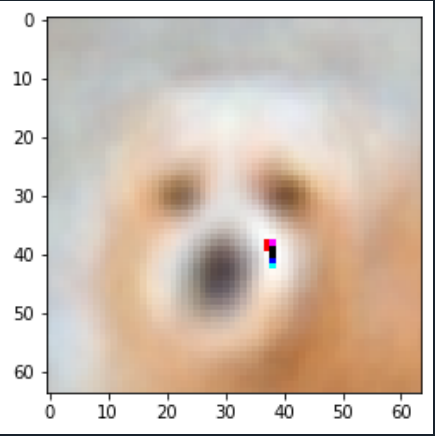
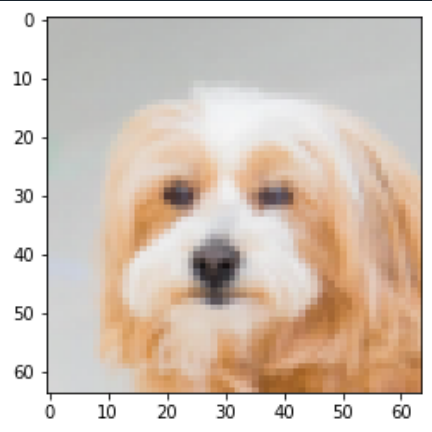


Figure 13- K=250 Image Reconstruction Figure 14- K=500 Image Reconstruction



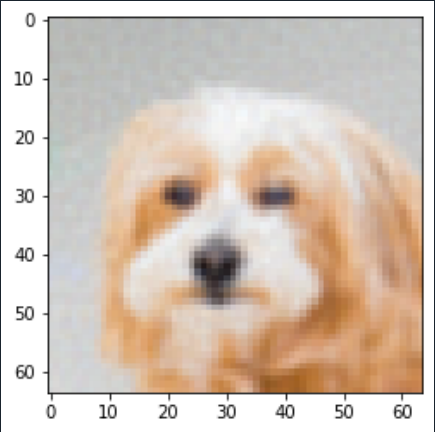


Figure 15- K=1000 Image Reconstruction Figure 16- K=4096 Image Reconstruction

Q.2.1

For this part, batch-gradient algorithm was implemented with Gaussian weights and also learning rate hyper parameter was added.

Cost after iteration 0: 0.818306

Cost after iteration 10: 0.807759

Cost after iteration 20: 0.798251

Cost after iteration 30: 0.789705

Cost after iteration 40: 0.782042

Cost after iteration 50: 0.775187

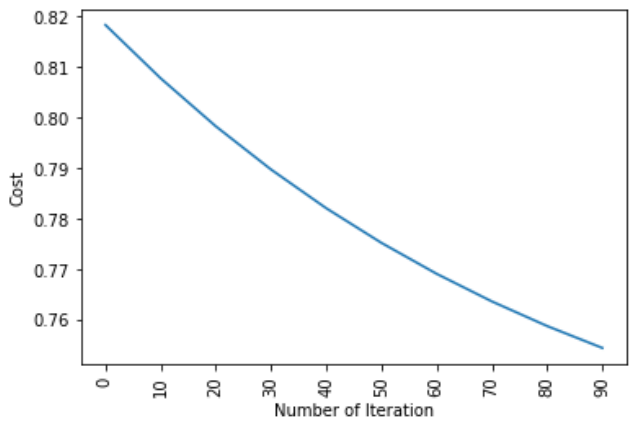
Cost after iteration 60: 0.769065

Cost after iteration 70: 0.763605

Cost after iteration 80: 0.758739

Cost after iteration 90: 0.754404

Test acc.: 62.96666666666667 %



Q2.2

Q2.3

Learning rate hyper parameter was implemented.

Q2.4

Appendix

PART-1

# -\*- coding: utf-8 -\*-

"""

Created on Mon Dec 4 15:31:41 2022

@author: Emre

"""

import numpy as np

import matplotlib.pyplot as plt

from pathlib import Path

from PIL import Image

def open\_directory(path, size):

im\_l =[]

im\_flattened = []

for image in Path(path).glob('\*'):

img=Image.open(image)

img\_array=np.array(img.resize((size,size), Image.BILINEAR))

im\_l.append(img\_array)

img\_flat = img\_array.flatten().reshape((size\*size),3)

im\_flattened.append(img\_flat)

return im\_l, im\_flattened

def PCA(array, num\_comp):

covariance = np.cov(array)

eig\_val, eig\_vec = np.linalg.eigh(covariance)

sort\_eig = np.argsort(-eig\_val)

eig\_val = eig\_val[sort\_eig]

eig\_vec = eig\_vec[:, sort\_eig]

Projection = eig\_vec[:, range(num\_comp)]

Z = Projection @ Projection.T

return Z

file\_directory = 'C:/Users/Emre/.spyder-py3/afhq\_dog'

im\_l, im\_flattened = open\_directory(file\_directory, 64)

im\_rgb = np.asarray(im\_flattened)

im\_red = im\_rgb[:,:,0]

im\_green = im\_rgb[:,:,1]

im\_blue = im\_rgb[:,:,2]

# for k in (1000):

color\_list = []

PVEs = []

for color in (im\_red, im\_green, im\_blue):

a = PCA(color.T, 10)

color\_list.append(a)

Red = im\_red @ color\_list[0]

Green = im\_green @ color\_list[1]

Blue = im\_blue @ color\_list[2]

pca\_out = np.array([Red.T, Green.T, Blue.T])

pca\_out -= pca\_out.min()

pca\_out /= pca\_out.ptp()

pca\_out\_Transpose = pca\_out.T

pca\_out\_final = pca\_out\_Transpose[:10,:,:].reshape(10,64,64,3)

# pca\_out\_final = pca\_out\_Transpose[0,:,:].reshape(64,64,3)

for i in range(10):

plt.figure(i+1)

plt.imshow(Image.fromarray((pca\_out\_final[i,:,:,:]\*255).astype(np.uint8)))

plt.show()

# img1 =(pca\_out\_final\*255).astype(np.uint8)

# plt.imshow(img1)

# img1 = im\_l[0] @ (first \* 255).astype(np.uint8)

# plt.imshow(img1)

PART-2

# -\*- coding: utf-8 -\*-

"""

Created on Mon Dec 5 03:25:51 2022

@author: Emre

"""

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv("dataset.csv")

# y\_sk = data.label.values

# x\_data\_sk = data.drop(["label"],axis=1)

# x\_sk= (x\_data\_sk - np.min(x\_data\_sk)) / (np.max(x\_data\_sk) - np.min(x\_data\_sk)).values

# x\_train\_sk, x\_test\_sk, y\_train\_sk, y\_test\_sk = train\_test\_split(x\_sk,y\_sk,test\_size = 0.2, random\_state = 42)

# x\_train\_sk = x\_train\_sk.T

# x\_test\_sk = x\_test\_sk.T

# y\_train\_sk = y\_train\_sk.T

# y\_test\_sk = y\_test\_sk.T

x\_data = data.drop(["label"],axis=1)

y = data.label.values

x = (x\_data - np.min(x\_data)) / (np.max(x\_data) - np.min(x\_data)).values

x['label']=y.tolist()

data=x

data = data.sample(frac = 1, random\_state=42)

train\_size = 0.7

valid\_size=0.2

train\_index = int(len(data)\*train\_size)

data\_train = data[0:train\_index]

data\_rem = data[train\_index:]

valid\_index = int(len(data)\*valid\_size)

data\_valid = data[train\_index:train\_index+valid\_index]

data\_test = data[train\_index+valid\_index:]

x\_train\_wosk, y\_train\_wosk = data\_train.drop(columns='label').copy(), data\_train['label'].copy(),

x\_valid\_wosk, y\_valid\_wosk = data\_valid.drop(columns='label').copy(), data\_valid['label'].copy()

x\_test\_wosk, y\_test\_wosk = data\_test.drop(columns='label').copy(), data\_test['label'].copy()

x\_train\_old=x\_train\_wosk

y\_train\_old=y\_train\_wosk

x\_train\_wosk=pd.concat([x\_train\_wosk, x\_valid\_wosk])

x\_train\_wosk=x\_train\_wosk.T

x\_test\_wosk = x\_test\_wosk.T

y\_train\_wosk=pd.concat([y\_train\_wosk, y\_valid\_wosk])

y\_train\_wosk=y\_train\_wosk.T

y\_train\_wosk=y\_train\_wosk.to\_numpy(dtype="int64")

y\_test\_wosk = y\_test\_wosk.T

y\_test\_wosk=y\_test\_wosk.to\_numpy(dtype="int64")

def sigmoid(z):

y\_hold = 1 / (1+np.exp(-z))

return y\_hold

def weights\_and\_bias(dimension):

w = np.random.normal(loc=0.0, scale=1.0, size=(dimension, 1))

b = 0.0

return w,b

def full\_batch(w,bias,x\_train,y\_train):

z = np.dot(w.T,x\_train) + bias

y\_hold = sigmoid(z)

loss = (-y\_train)\*np.log(y\_hold) - ((1-y\_train))\*np.log(1-y\_hold)

cost = (np.sum(loss)) / x\_train.shape[1]

#backward propogation

weight = (np.dot(x\_train,((y\_hold-y\_train).T)))/x\_train.shape[1]

derivative\_bias = np.sum(y\_hold-y\_train)/x\_train.shape[1]

gradients = {"weight": weight,"derivative\_bias": derivative\_bias}

return cost,gradients

def predict(w,bias,x\_test):

# x\_test is a input for forward propagation

z = sigmoid(np.dot(w.T,x\_test)+bias)

Y\_prediction = np.zeros((1,x\_test.shape[1]))

# if z is bigger than 0.5, our prediction is one means has diabete (y\_hold=1),

# if z is smaller than 0.5, our prediction is zero means does not have diabete (y\_hold=0),

for i in range(z.shape[1]):

if z[0,i]<= 0.5:

Y\_prediction[0,i] = 0

else:

Y\_prediction[0,i] = 1

return Y\_prediction

def update(w, bias, x\_train, y\_train, learning\_rate,number\_of\_iterarion):

cost\_l = []

cost\_l\_two = []

index = []

# updating(learning) parameters is number\_of\_iterarion times

for i in range(number\_of\_iterarion):

# make forward and backward propagation and find cost and gradients

cost,gradients = full\_batch(w,bias,x\_train,y\_train)

cost\_l.append(cost)

# lets update

w = w - learning\_rate \* gradients["weight"]

bias = bias - learning\_rate \* gradients["derivative\_bias"]

if i % 10 == 0:

cost\_l\_two.append(cost)

index.append(i)

print ("Cost after iteration %i: %f" %(i, cost)) #if section defined to print our cost values in every 10 iteration. We do not need to do that. It's optional.

# we update(learn) parameters weights and bias

parameters = {"weight": w,"bias": bias}

plt.plot(index,cost\_l\_two)

plt.xticks(index,rotation='vertical')

plt.xlabel("Number of Iteration")

plt.ylabel("Cost")

plt.show()

return parameters, gradients, cost\_l

def logistic\_regression(x\_train, y\_train, x\_test, y\_test, learning\_rate , num\_iterations):

# initialize

dimension = x\_train.shape[0]

w,bias = weights\_and\_bias(dimension)

parameters, gradients, cost\_l = update(w, bias, x\_train, y\_train, learning\_rate,num\_iterations)

y\_prediction\_test = predict(parameters["weight"],parameters["bias"],x\_test)

# Print train/test Errors

print("Test acc.: {} %".format(100 - np.mean(np.abs(y\_prediction\_test - y\_test)) \* 100))

logistic\_regression(x\_train\_wosk, y\_train\_wosk, x\_test\_wosk, y\_test\_wosk,learning\_rate = 0.01, num\_iterations = 100)