Practical NO.1

Source Code:

import numpy as np

import copy

import matplotlib.pyplot as plt

import h5py

import scipy

from PIL import Image

from scipy import ndimage

def load\_dataset():

train\_ds = h5py.File(r"C:\Users\Lenovo\Downloads\train\_catvnoncat.h5")

train\_set\_x = np.array(train\_ds['train\_set\_x'][:])

train\_set\_y = np.array(train\_ds['train\_set\_y'][:])

test\_ds = h5py.File(r"C:\Users\Lenovo\Downloads\test\_catvnoncat.h5")

test\_set\_x = np.array(test\_ds['test\_set\_x'][:])

test\_set\_y = np.array(test\_ds['test\_set\_y'][:])

classes = np.array(test\_ds['list\_classes'][:])

train\_set\_y = train\_set\_y.reshape((1, train\_set\_y.shape[0]))

test\_set\_y = test\_set\_y.reshape((1, test\_set\_y.shape[0]))

return train\_set\_x, train\_set\_y, test\_set\_x, test\_set\_y, classes

train\_set\_x\_orig, train\_set\_y, test\_set\_x\_orig, test\_set\_y, classes = load\_dataset()

index = 30

plt.imshow(train\_set\_x\_orig[index])

print ("y = " + str(train\_set\_y[:, index]) + ", it's a '" + classes[np.squeeze(train\_set\_y[:, index])].decode("utf-8") +

"' picture.")

index = 25

plt.imshow(train\_set\_x\_orig[index])

print ("y = " + str(train\_set\_y[:, index]) + ", it's a '" + classes[np.squeeze(train\_set\_y[:, index])].decode("utf-8") +

"' picture.")

m\_train = train\_set\_x\_orig.shape[0]

m\_test = test\_set\_x\_orig.shape[0]

num\_px = train\_set\_x\_orig.shape[1]

print ("Number of training examples: m\_train = " + str(m\_train))

print ("Number of testing examples: m\_test = " + str(m\_test))

print ("Height/Width of each image: num\_px = " + str(num\_px))

print ("Each image is of size: (" + str(num\_px) + ", " + str(num\_px) + ", 3)")

print ("train\_set\_x shape: " + str(train\_set\_x\_orig.shape))

print ("train\_set\_y shape: " + str(train\_set\_y.shape))

print ("test\_set\_x shape: " + str(test\_set\_x\_orig.shape))

print ("test\_set\_y shape: " + str(test\_set\_y.shape))

train\_set\_x\_flatten = train\_set\_x\_orig.reshape(train\_set\_x\_orig.shape[0], -1).T

test\_set\_x\_flatten = test\_set\_x\_orig.reshape(test\_set\_x\_orig.shape[0], -1).T

print ("train\_set\_x\_flatten shape: " + str(train\_set\_x\_flatten.shape))

print ("train\_set\_y shape: " + str(train\_set\_y.shape))

print ("test\_set\_x\_flatten shape: " + str(test\_set\_x\_flatten.shape))

print ("test\_set\_y shape: " + str(test\_set\_y.shape))

train\_set\_x = train\_set\_x\_flatten / 255.

test\_set\_x = test\_set\_x\_flatten / 255.

def sigmoid(z):

s = 1 / (1 + np.exp(-z))

return s

print ("sigmoid([0, 2]) = " + str(sigmoid(np.array([0,2]))))

x = np.array([0.5, 0, 2.0])

output = sigmoid(x)

print(output)

def initialize\_with\_zeros(dim):

w = np.zeros(shape=(dim, 1), dtype=np.float32)

b = 0.0

return w, b

dim = 2

w, b = initialize\_with\_zeros(dim)

assert type(b) == float

print ("w = " + str(w))

print ("b = " + str(b))

def propagate(w, b, X, Y):

m = X.shape[1]

A = sigmoid(w.T @ X + b)

cost = np.sum(Y \* np.log(A) + (1 - Y) \* np.log(1 - A)) / -m

dw = X @ (A - Y).T / m

db = np.sum(A - Y) / m

cost = np.squeeze(np.array(cost))

grads = {'dw': dw, 'db': db}

return grads, cost

w = np.array([[1.], [2]])

b = 1.5

X = np.array([[1., -2., -1.], [3., 0.5, -3.2]])

Y = np.array([[1, 1, 0]])

grads, cost = propagate(w, b, X, Y)

assert type(grads["dw"]) == np.ndarray

assert grads["dw"].shape == (2, 1)

assert type(grads["db"]) == np.float64

print ("dw = " + str(grads["dw"]))

print ("db = " + str(grads["db"]))

print ("cost = " + str(cost))

def optimize(w, b, X, Y, num\_iterations=100, learning\_rate=0.009, print\_cost=False):

w = copy.deepcopy(w)

b = copy.deepcopy(b)

costs = []

for i in range(num\_iterations):

grads, cost = propagate(w, b, X, Y)

dw = grads["dw"]

db = grads["db"]

w -= learning\_rate \* dw

b -= learning\_rate \* db

if i % 100 == 0:

costs.append(cost)

if print\_cost:

print ("Cost after iteration %i: %f" %(i, cost))

params = {"w": w,

"b": b}

grads = {"dw": dw,

"db": db}

return params, grads, costs

params, grads, costs = optimize(w, b, X, Y, num\_iterations=100, learning\_rate=0.009, print\_cost=False)

print ("w = " + str(params["w"]))

print ("b = " + str(params["b"]))

print ("dw = " + str(grads["dw"]))

print ("db = " + str(grads["db"]))

print("Costs = " + str(costs))

def predict(w, b, X):

m = X.shape[1]

Y\_prediction = np.zeros((1, m))

w = w.reshape(X.shape[0], 1)

# compute vector 'A' predicting the probabilities of a cat being present in the picture

A = sigmoid(w.T @ X + b)

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

# convert probabilities A[0, i] to actual predictions p[0, i]

if A[0, i] > 0.5:

Y\_prediction[0, i] = 1

else:

Y\_prediction[0, i] = 0

return Y\_prediction

def model(X\_train, Y\_train, X\_test, Y\_test, num\_iterations=3000, learning\_rate=0.5, print\_cost=False):

w, b = initialize\_with\_zeros(dim=X\_train.shape[0])

# Gradient descent

params, grads, costs = optimize(w, b, X\_train, Y\_train, num\_iterations, learning\_rate, print\_cost)

# Retrieve parameters w and b from dictionary "params"

w = params['w']

b = params['b']

# Predict test/train set examples

Y\_prediction\_test = predict(w, b, X\_test)

Y\_prediction\_train = predict(w, b, X\_train)

# Print train/test Errors

if print\_cost:

print("train accuracy: {} %".format(100 - np.mean(np.abs(Y\_prediction\_train - Y\_train)) \* 100))

print("test accuracy: {} %".format(100 - np.mean(np.abs(Y\_prediction\_test - Y\_test)) \* 100))

d = {"costs": costs,

"Y\_prediction\_test": Y\_prediction\_test,

"Y\_prediction\_train" : Y\_prediction\_train,

"w" : w,

"b" : b,

"learning\_rate" : learning\_rate,

"num\_iterations": num\_iterations}

return d

logistic\_regression\_model = model(train\_set\_x, train\_set\_y, test\_set\_x, test\_set\_y, num\_iterations=3500, learning\_rate=0.001, print\_cost=True)

index = 1

plt.imshow(test\_set\_x[:, index].reshape((num\_px, num\_px, 3)))

print ("y = " + str(test\_set\_y[0,index]) + ", you predicted that it is a \"" + classes[int(logistic\_regression\_model['Y\_prediction\_test'][0,index])].decode("utf-8") + "\" picture.")

learning\_rates = [0.01, 0.001, 0.0001]

models = {}

for lr in learning\_rates:

print ("Training a model with learning rate: " + str(lr))

models[str(lr)] = model(train\_set\_x, train\_set\_y, test\_set\_x, test\_set\_y, num\_iterations=1500, learning\_rate=lr, print\_cost=False)

print ('\n' + "-------------------------------------------------------" + '\n')

for lr in learning\_rates:

plt.plot(np.squeeze(models[str(lr)]["costs"]), label=str(models[str(lr)]["learning\_rate"]))

plt.ylabel('cost')

plt.xlabel('iterations (hundreds)')

legend = plt.legend(loc='upper center', shadow=True)

frame = legend.get\_frame()

frame.set\_facecolor('0.90')

plt.show()

Output:-



