import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

# Load the data

data = np.loadtxt('iris.txt', skiprows=1)

X\_data = data[:,:2]

Y\_data = data[:,2:] # Note this is a matrix

# For feeding in data

x = tf.placeholder(tf.float32, [None, 2])

y = tf.placeholder(tf.float32, [None, 1])

# Model parameters

W = tf.Variable(tf.zeros([2, 1]))

b = tf.Variable([0.0])

# Define the model

logits = tf.matmul(x, W) + b

# Loss function

loss = tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=y, logits=logits)

loss = tf.reduce\_mean(loss)

# Accuracy

predict\_op = tf.greater\_equal(logits, tf.zeros\_like(logits))

correct\_op = tf.equal(tf.cast(predict\_op, tf.float32), y)

accuracy\_op = tf.reduce\_mean(tf.cast(correct\_op, tf.float32))

# Hyperparameters

learning\_rate = 0.01

num\_epochs = 100

# Optimizer

optimizer = tf.train.GradientDescentOptimizer(learning\_rate)

train\_op = optimizer.minimize(loss)

# TF session

sess = tf.Session()

sess.run(tf.global\_variables\_initializer())

# Seed the random number generator for reproducibility

np.random.seed(0)

# Minimize the loss function

for epoch in range(num\_epochs):

# Present each data point once in random order

idx = np.random.permutation(data.shape[0])

for i in idx:

feed\_dict = {x: X\_data[i:i+1], y: Y\_data[i:i+1]}

sess.run(train\_op, feed\_dict)

if (epoch+1) % 10 == 0:

feed\_dict = {x: X\_data, y: Y\_data}

accuracy = sess.run(accuracy\_op, feed\_dict)

print("After {} epochs, accuracy = {}".format(epoch+1, accuracy))

# Print the result

W\_val, b\_val = sess.run([W, b])

W\_val = W\_val[:,0]

b\_val = b\_val[0]

print("W =", W\_val)

print("b =", b\_val)

def predict(x\_):

return 1 \* sess.run(predict\_op, {x: x\_})

# Model predictions

labels = predict(X\_data)[:,0]

# Find indices for the two species

idx\_0, = np.where(labels == 0)

idx\_1, = np.where(labels == 1)

# Plot the data

plt.plot(X\_data[idx\_0,0], X\_data[idx\_0,1], 'bo', label='I. versicolor')

plt.plot(X\_data[idx\_1,0], X\_data[idx\_1,1], 'ro', label='I. virginica')

# Plot the separating hyperplane

x\_sep = np.linspace(X\_data[:,0].min(), X\_data[:,0].max())

y\_sep = (-b\_val - W\_val[0]\*x\_sep) / W\_val[1]

plt.plot(x\_sep, y\_sep, 'm', label="Decision boundary")

# Legend

plt.legend()

# Axis labels

plt.xlabel("Sepal length (cm)")

plt.ylabel("Petal legnth (cm)")

#Plotting the figure

plt.show()