Assignment-3

Git: <https://github.com/iamankan/MA421G.git>

Branch: assignments

File: Copy\_of\_MA421Homework3Problem.ipynb

Colab link: <https://colab.research.google.com/drive/1-VjDdh96zHOXlbzF4jTgMH8rhmKCltTs?usp=sharing>

Code:

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  import matplotlib.pyplot as plt  def sigmoid(t):  return 1/(1+np.exp(-t))  def customloss(y, z):  # loss function for y and yhat = sigmoid(z)  return ((-y\*log\_sig(z)) - ((1-y)\*log\_one\_sig(z)))  def log\_sig(t):  return np.log(sigmoid(t))  def log\_one\_sig(t):  return np.log(1-sigmoid(t))  def model(w,b,X):  # using X as Nxn  print(f'In model, X: {X.shape}, b: {b}, w: {w.shape}')  return (X @ w)+b  def gradients(X, y, y\_hat):  # Using X as Nxn  # print(f'grad: y shape: {y.shape}, X shape: {X.shape}')  return (np.transpose(X) @ (y\_hat - y))/X.shape[0]  def train(w, b, X, y, iter, lr):  print(f'>> {X.shape}')  losslist=list()  for k in range(iter):  z = model(w, b, X)  y\_hat = sigmoid(z)  grad = gradients(X, y, y\_hat)  print(f'gradient shape: {grad.shape}')  w = w - (lr \* grad)  b = np.mean((b\*np.ones(y\_hat.shape)) - (lr \* (y\_hat - y)))  myloss = customloss(y, y\_hat)  losslist.append(np.mean(myloss))  print(f'Iter: {k} Loss: {losslist[-1]}')  return w, b, losslist  def predict(z):  ypred = sigmoid(z)  ypred[ypred<=0.5]=0  ypred[ypred>0.5]=1  ypred = ypred.astype(int)  ypred = np.squeeze(ypred)  # print(f'In pred, {ypred.shape}')  return ypred  def accuracy(y, y\_label):  diff\_bool = (y == y\_label)  diff\_true = diff\_bool[diff\_bool==True]  total\_sample = len(diff\_bool)  return (len(diff\_true)/total\_sample)  from sklearn.datasets import make\_moons  X\_train, y\_train = make\_moons(n\_samples=500, noise=0.1)  X\_test, y\_test = make\_moons(n\_samples=1000, noise=0.1)  print(X\_train.shape)  plt.figure()  plt.plot(X\_train[:, 0][y\_train==0], X\_train[:, 1][y\_train==0], "g^")  plt.plot(X\_train[:, 0][y\_train==1], X\_train[:, 1][y\_train==1], "bs")  from sklearn.datasets import make\_classification  X\_train, y\_train = make\_classification(n\_samples=1000, n\_features=4)  X\_test=X\_train[500:,]  y\_test=y\_train[500:,]  X\_train=X\_train[:500,]  y\_train=y\_train[:500,]  plt.figure()  plt.plot(X\_train[:, 0][y\_train==0], X\_train[:, 1][y\_train==0], "g^")  plt.plot(X\_train[:, 0][y\_train==1], X\_train[:, 1][y\_train==1], "bs")  w = np.random.rand(X\_train.shape[1],1) # assuming X is N-by-n.  # if X is n-by-N, use X\_train.shape[0]  y\_train = y\_train.reshape((-1,1))  y\_test = y\_test.reshape((-1,1))  print(w.shape)  print(X\_train.shape)  print(y\_train.shape)  b = 0  w, b, loss = train(w, b, X\_train, y\_train, iter=300, lr=0.1)  plt.figure()  plt.plot(loss)  #training accuracy  z = model(w,b,X\_train)  print(accuracy(np.squeeze(y\_train), predict(z)))  z = model(w,b,X\_test)  y\_test=np.squeeze(y\_test)  print(y\_test)  print(predict(z))  print(accuracy(y\_test, predict(z)))  url = 'https://raw.githubusercontent.com/madmashup/targeted-marketing-predictive-engine/master/banking.csv'  data = pd.read\_csv(url)  print(data.shape)  print(list(data.columns))  cat\_vars=['default','education','contact','month','day\_of\_week',]  data=data.drop(cat\_vars, axis=1)  print(list(data.columns))  print(data.shape)  cat\_vars=['job','marital','housing','loan','poutcome']  for va in cat\_vars:  #cat\_pre='var'+'\_'+var  print(va)  #print(data[va])  cat\_list = pd.get\_dummies(data[va])  data1=pd.concat([data,cat\_list], axis=1)  data=data1.drop(va, axis=1)  #print(list(cat\_list.columns))  #print(list(data.columns))  #print(data.shape)  print(data.shape)  print(list(data.columns))  X = data.loc[:, data.columns != 'y']  y = data.loc[:, data.columns == 'y']  columns = X.columns  X=X.to\_numpy()  y=y.to\_numpy()  X\_train1, X\_test1, y\_train1, y\_test1 = train\_test\_split(X, y, test\_size=0.3, random\_state=0)  print(X\_train1.shape)  print(X\_test1.shape)  print(columns)  w1 = np.random.rand(X\_train1.shape[1],1) # assuming X is N-by-n.  # if X is n-by-N, use X\_train.shape[0]  y\_train1 = y\_train1.reshape(-1,1)  y\_test1 = y\_test1.reshape(-1,1)  print(w1.shape)  print(X\_train1.shape)  print(y\_train1.shape)  b1 = 0  w1, b1, loss1 = train(w1, b1, X\_train1, y\_train1, iter=300, lr=0.1)  plt.figure()  plt.plot(loss1)  #training accuracy  z1 = model(w1,b1,X\_train1)  print(accuracy(np.squeeze(y\_train1), predict(z1)))  z1 = model(w1,b1,X\_test1)  y\_test1=np.squeeze(y\_test1)  print(accuracy(y\_test1, predict(z1))) |

Output moon dataset:

|  |
| --- |
|  |