```
"""CSE 6363 - PROJ01 - Part 1
"""


from pprint import pprint

from numpy.lib.function_base import iterable
from sklearn.preprocessing import StandardScaler
import pdb
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np


#import pdb;

D = [ ((6.4432, 9.6309), 50.9155), ((3.7861, 5.4681),  29.9852),
  ((8.1158, 5.2114),  42.9626), ((5.3283, 2.3159),  24.7445),
  ((3.5073, 4.8890),  27.3704), ((9.3900, 6.2406), 51.1350),
  ((8.7594, 6.7914), 50.5774), ((5.5016, 3.9552), 30.5206),
  ((6.2248, 3.6744), 31.7380), ((5.8704, 9.8798), 49.6374),
  ((2.0774, 0.3774), 10.0634), ((3.0125, 8.8517), 38.0517),
  ((4.7092, 9.1329), 43.5320), ((2.3049, 7.9618), 33.2198),
  ((8.4431, 0.9871), 31.1220), ((1.9476, 2.6187), 16.2934),
  ((2.2592, 3.3536), 19.3899), ((1.7071, 6.7973), 28.4807),
  ((2.2766, 1.3655), 13.6945), ((4.3570, 7.2123), 36.9220),
  ((3.1110, 1.0676), 14.9160), ((9.2338, 6.5376), 51.2371),
  ((4.3021, 4.9417), 29.8112), ((1.8482, 7.7905), 32.0336),
  ((9.0488, 7.1504), 52.5188), ((9.7975, 9.0372), 61.6658),
  ((4.3887, 8.9092), 42.2733), ((1.1112, 3.3416), 16.5052),
  ((2.5806, 6.9875), 31.3369), ((4.0872, 1.9781), 19.9475),
  ((5.9490, 0.3054), 20.4239), ((2.6221, 7.4407), 32.6062),
  ((6.0284, 5.0002), 35.1676), ((7.1122, 4.7992), 38.2211),
  ((2.2175, 9.0472), 36.4109), ((1.1742, 6.0987), 25.0108),
  ((2.9668, 6.1767), 29.8861), ((3.1878, 8.5944), 37.9213),
  ((4.2417, 8.0549), 38.8327), ((5.0786, 5.7672), 34.4707)]

class PolyReg:
  def __init__(self, degree=1, learningRate=0.01, iterations=100):
    self.degree = degree
    self.learningRate = learningRate
    self.iterations = iterations

  def train(self, X, Y, theta, show_prog=True):
    iterations = self.iterations

    m, n = X.shape
    theta_hist = np.ndarray((iterations, m, n))
    cost_hist = np.ndarray((iterations, 1))
    #loss_hist = np.ndarray((iterations, m,n))
    Xtrans = X.transpose()
    for i in range(0, iterations):
        hypothesis = np.dot(X, theta)
        #pdb.set_trace()
```

```
        loss = hypothesis - Y
        J = np.sum(loss ** 2) / (2 * m)  # cost
        if show_prog == True:
          print(" cost(",i,"): {:.3f}".format(J))
        gradient = np.dot(Xtrans, loss) / m
        #pdb.set_trace()
        theta = theta - self.learningRate * gradient
        #pdb.set_trace()
        #print("loss: ", loss)
        #print("Cost: ", J)
        #print("hypothesis")
        #pprint(hypothesis)
        #print()
        theta_hist[i,:,:] = theta.T
        cost_hist[i] = J
        #loss_hist[i] = loss
    return theta, theta_hist, cost_hist


  def init_theta(self, shape):
        # initialize weights (theta)
    theta = np.random.uniform(low=0.9, high=1.1, size=shape)
    return theta

def predict(X,theta):
  return np.dot(X, theta)

def prepare_data(dataset, deg=4):
  X = list()
  Y = list()
  for datum in dataset:
    X.append(np.transpose(np.asarray(datum[0], dtype=np.float)))
    Y.append(datum[1])
  X = np.asarray(X)
  Y = np.asarray(Y)
  X.reshape(-1,X.ndim)
  Y = np.expand_dims(Y, axis=1)
  m, _ = X.shape
  stdScaler = StandardScaler()
  X = stdScaler.fit_transform(X)
  X = np.concatenate((np.ones((len(dataset),1)), X), axis=1)

  if deg == 1:
    X = np.hstack((
      (X[:,0]     ).reshape((m,1)), # 0
      (X[:,1] ** 1).reshape((m,1)),
      (X[:,2] ** 1).reshape((m,1)),
    ))
  elif deg == 2:
    X = np.hstack((
      (X[:,0]     ).reshape((m,1)), # 0
      (X[:,1] ** 1).reshape((m,1)),
      (X[:,1] ** 2).reshape((m,1)),
      (X[:,2] ** 1).reshape((m,1)),
```

This is where we update theta and learn nonlinear features.

```
      (X[:,2] ** 2).reshape((m,1)),
    ))
  elif deg == 3:
    X = np.hstack((
      (X[:,0]     ).reshape((m,1)), # 0
      (X[:,1] ** 1).reshape((m,1)),
      (X[:,1] ** 2).reshape((m,1)),
      (X[:,1] ** 3).reshape((m,1)),
      (X[:,2] ** 1).reshape((m,1)),
      (X[:,2] ** 2).reshape((m,1)),
      (X[:,2] ** 3).reshape((m,1)),
    ))
  elif deg == 4:
    X = np.hstack((
      (X[:,0]     ).reshape((m,1)), # 0
      (X[:,1] ** 1).reshape((m,1)),
      (X[:,1] ** 2).reshape((m,1)),
      (X[:,1] ** 3).reshape((m,1)),
      (X[:,1] ** 4).reshape((m,1)),
      (X[:,2] ** 1).reshape((m,1)),
      (X[:,2] ** 2).reshape((m,1)),
      (X[:,2] ** 3).reshape((m,1)),
      (X[:,2] ** 4).reshape((m,1)), # 9
    ))

  return X, Y

  """hypothesis function, predicts Y_est based on X and theta
  """
def hypothesis(X, theta):
  return X @ theta

def cost(theta, X, Y):
  m = len(Y)
  J = np.mean(np.square(hypothesis(X,theta) - Y))/(2*m)
  return J

def plot_cost(cost_hist):
  x = np.arange(0, len(cost_hist), 1)
  #ymin = np.min(cost_hist)
  #ymax = np.max(cost_hist)

  plt.plot(x, cost_hist, color='red')
  plt.xlabel('iterations')
  plt.ylabel('cost')
  plt.show()
  return

def getR2(Y_, Y):
  exp_mean = np.sum((Y-Y.mean())**2)
  mean = np.sum((Y_-Y)**2)
  return (1 - (mean/exp_mean))

def getX1X2(X, deg):
```

```
  if deg == 1:
    X1 = X[:,1]
    X2 = X[:,2]
  elif deg == 2:
    X1 = X[:,1]
    X2 = X[:,3]
  elif deg == 3:
    X1 = X[:,1]
    X2 = X[:,3]
  elif deg == 4:
    X1 = X[:,1]
    X2 = X[:,4]
  return X1, X2

def prepare_testdata(testset, deg=1):
  testset = np.asarray(testset)
  X = list()
  Y = list()
  for i in range(len(testset)):
    X.append(np.asarray((testset[i][0],testset[i][1]) , dtype=np.float))
    Y.append(testset[i][2])
  X = np.asarray(X)
  Y = np.asarray(Y)
  Y = np.expand_dims(Y, axis=1)
  stdScaler = StandardScaler()
  X = stdScaler.fit_transform(X)
  X = np.concatenate((np.ones((len(testset),1)), X), axis=1)
  m, _ = X.shape
  if deg == 1:
    X = np.hstack((
      (X[:,0]     ).reshape((m,1)), # 0
      (X[:,1] ** 1).reshape((m,1)),
      (X[:,2] ** 1).reshape((m,1)),
    ))
  elif deg == 2:
    X = np.hstack((
      (X[:,0]     ).reshape((m,1)), # 0
      (X[:,1] ** 1).reshape((m,1)),
      (X[:,1] ** 2).reshape((m,1)),
      (X[:,2] ** 1).reshape((m,1)),
      (X[:,2] ** 2).reshape((m,1)),
    ))
  elif deg == 3:
    X = np.hstack((
      (X[:,0]     ).reshape((m,1)), # 0
      (X[:,1] ** 1).reshape((m,1)),
      (X[:,1] ** 2).reshape((m,1)),
      (X[:,1] ** 3).reshape((m,1)),
      (X[:,2] ** 1).reshape((m,1)),
      (X[:,2] ** 2).reshape((m,1)),
      (X[:,2] ** 3).reshape((m,1)),
    ))
  elif deg == 4:
    X = np.hstack((
```

```
      (X[:,0]     ).reshape((m,1)), # 0
      (X[:,1] ** 1).reshape((m,1)),
      (X[:,1] ** 2).reshape((m,1)),
      (X[:,1] ** 3).reshape((m,1)),
      (X[:,1] ** 4).reshape((m,1)),
      (X[:,2] ** 1).reshape((m,1)),
      (X[:,2] ** 2).reshape((m,1)),
      (X[:,2] ** 3).reshape((m,1)),
      (X[:,2] ** 4).reshape((m,1)), # 9
    ))
  return X, Y


def runtest(testset, deg, theta):
  Xt, Yt = prepare_testdata(testset, deg)
  Yt_ = predict(Xt, theta)
  #Yt = Yt/np.mean(Yt)
  #Yt_ = Yt_/np.mean(Yt_)
  R2t = getR2(Yt_, Yt)
  print(deg," deg model accuracy for test data (R2): {:.3f}".format(R2t))
  return

if __name__ == '__main__':

  """ Complete Dataset --- Part a and b """
  file = open("../PolyTest.txt", 'rt')
  data = file.read()
  testset = np.fromstring(data, dtype=np.float, count=-1, sep="\n")
  testset =  testset.reshape(10,3)
  #pprint(testset)
  stdsc = StandardScaler()
  testset = stdsc.fit_transform(testset)

  #pprint(testset)

  """for i in np.arange(len(testset[0])):
    max_ = np.max(testset[:,i])
    min_ = np.min(testset[:,i])
    if min_ < .001: min_ = .001
    mean = np.mean(testset[:, i])
    print(mean)
    print(i)
    for j in np.arange(len(testset)):
      testset[j, i] = max_ - (((max_ - min_) * (max_ - testset[j, i])) / (max_-min_))

  pprint(testset)
  """
  #pdb.set_trace()


  # First degree
  model = PolyReg(degree=1, learningRate=0.01, iterations=180)
  X, Y = prepare_data(D,deg=model.degree)
  theta = model.init_theta(shape=((model.degree*2)+1,1))
```

```
theta, theta_hist, cost_hist = model.train(X,Y, theta, show_prog=False)
Y_ = predict(X, theta)
R2 = getR2(Y_, Y)
print("1st deg model accuracy for training(R2): {:.3f}".format(R2))
#plot_cost(cost_hist)
# test data
runtest(testset, model.degree, theta)

# Second degree
model = PolyReg(degree=2, learningRate=0.01, iterations=200)
X, Y = prepare_data(D,deg=model.degree)
theta = model.init_theta(shape=((model.degree*2)+1,1))
theta, theta_hist, cost_hist = model.train(X,Y, theta, show_prog=False)
Y_ = predict(X, theta)
R2 = getR2(Y_, Y)
print("2nd deg model accuracy (R2): {:.3f}".format(R2))
#plot_cost(cost_hist)
runtest(testset, model.degree, theta)


# Third degree
model = PolyReg(degree=3, learningRate=0.01, iterations=200)
X, Y = prepare_data(D,deg=model.degree)
theta = model.init_theta(shape=((model.degree*2)+1,1))
theta, theta_hist, cost_hist = model.train(X,Y, theta, show_prog=False)
Y_ = predict(X, theta)
R2 = getR2(Y_, Y)
print("3rd deg model accuracy (R2): {:.3f}".format(R2))
#plot_cost(cost_hist)
runtest(testset, model.degree, theta)


# Fourth degree
model = PolyReg(degree=4, learningRate=0.01, iterations=200)
X, Y = prepare_data(D,deg=model.degree)
theta = model.init_theta(shape=((model.degree*2)+1,1))
theta, theta_hist, cost_hist = model.train(X,Y, theta, show_prog=False)
Y_ = predict(X, theta)
R2 = getR2(Y_, Y)
print("4th deg model accuracy (R2): {:.3f}".format(R2))
#plot_cost(cost_hist)
runtest(testset, model.degree, theta)




#pdb.set_trace()

#x = np.linspace()

#pprint(theta)
# print("Const_hist")
#pprint(cost_hist)
```

```
 # ax = plt.axes(projection='3d')
  #ax.plot3D(X[:,1], X[:,6], Y,  'rx', label='XY')
  #ax.plot_surface(model.X[:,1], model.X[:,2], hypothesis(model.X, model.theta), rstride=1, cstride=1, cmap='viridis
', label='Y_est')
  #plt.show()




'''
  plt.scatter(model.X, model.Y, color='green')
  plt.plot(model.X, Y_est, color='red')
  plt.title('Polynomial Fit - Linear Regression')
  plt.xlabel('X')
  plt.ylabel('Y')
  plt.show()
'''

""" end of linReg.py """
```