1072 Deep Learning – Homework 1

Due: April 12, 2019, 11:55pm

1. (10%) (Maximum Likelihood Estimation) Please set the derivatives of the log likelihood function of a Gaussian Probability Distribution Function to zero with respect to and and verify the following results below:

對做偏微分:

=

= = 0 //依題意將derivative設為0，可得極值，求得的

=>左右移項 => = = =>

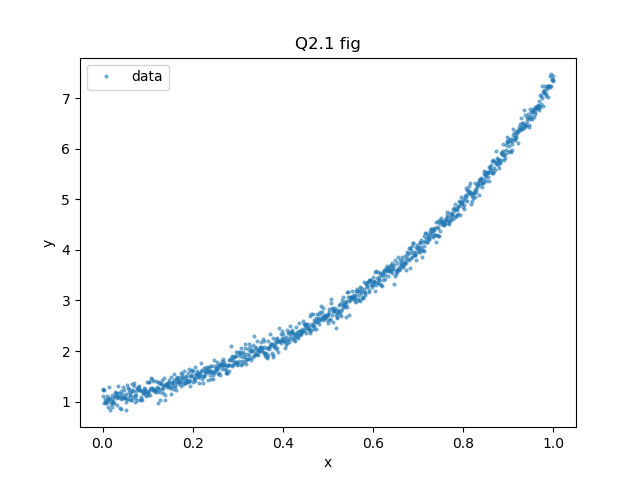
對做偏微分:

= = 0 //依題意將derivative設為0，可得極值，求得的

=>左右移項=>=>

=> => =

1. (10%) Please load ‘data.mat’ into your Matlab or Python code, where you will find . Now do the following procedures, paste your source code and show the results in your report.
   1. Plot the data using plot function.



import scipy.io

from matplotlib import pyplot as plt

mat = scipy.io.loadmat('data.mat')

plt.plot(mat['x'], mat['y'], "o", ms=2, alpha=0.5, label='data')

plt.legend(loc='best')

plt.xlabel("x")

plt.ylabel("y")

Title="Q2.1 fig"

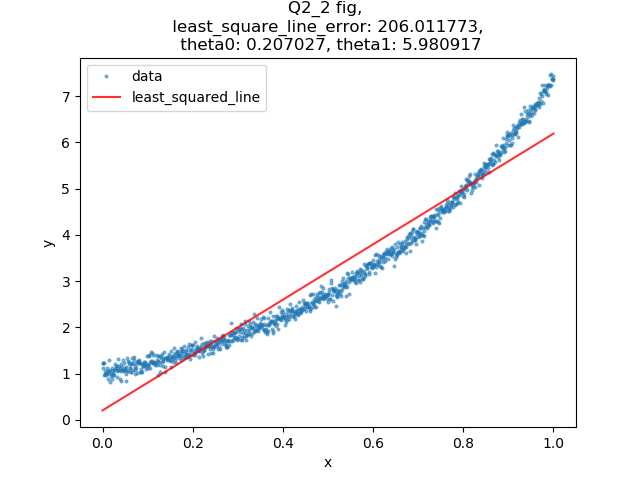
plt.title(Title)

plt.savefig("./picture/hw1q2\_1.png")

plt.grid()

plt.show()

* 1. Compute the least square line using the given data and overlay the line over the given data.



import scipy.io

import numpy as np

from numpy import ones

from numpy.linalg import inv

from matplotlib import pyplot as plt

mat = scipy.io.loadmat('data.mat')

# At\*A\*x = At\*b, x= [theta0, theta1] is the least square solution to ||Ax-b||

A = np.concatenate((ones([len(mat['x']),1]), mat['x']), axis=1)

At = A.transpose()

At\_dot\_A = np.dot(At, A)

inverse\_At\_dot\_A = inv(At\_dot\_A)

At\_dot\_b = np.dot(At, mat['y'])

x = np.dot(inverse\_At\_dot\_A, At\_dot\_b)

theta0 = x[0][0]

theta1 = x[1][0]

print(theta0, theta1)

# compute least\_square\_line\_error

least\_square\_line\_error = 0

for i in range(len(mat['x'])):

least\_square\_line\_error += pow(abs(mat['y'][i][0] - (theta0 + mat['x'][i][0]\*theta1)), 2)

print("least\_square\_line\_error: ")

print(least\_square\_line\_error)

# y = theta0 + x\*theta1, let's plot

plt.plot(mat['x'], mat['y'], 'o', ms=2, alpha=0.5, label = 'data')

plt.plot(mat['x'], theta0 + theta1\*mat['x'], 'r', ms=1, alpha=0.8, label = 'least\_squared\_line')

plt.legend(loc='best')

plt.xlabel("x")

plt.ylabel("y")

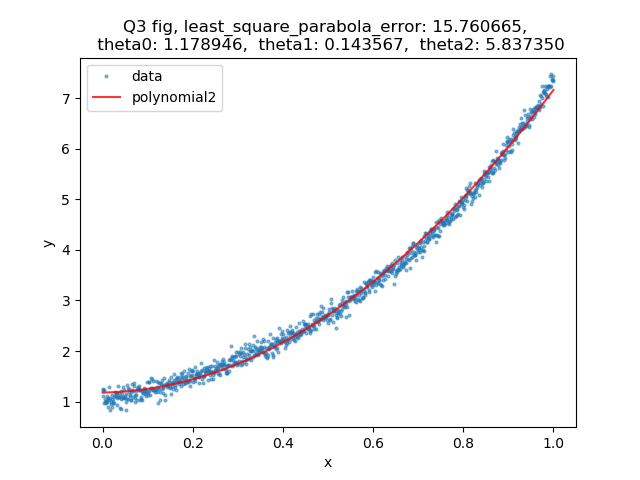
plt.title("Q2\_2 fig, \n least\_square\_line\_error: %f, \n theta0: %f, theta1: %f" %(least\_square\_line\_error, theta0, theta1))

plt.savefig("./picture/hw1q2\_2.png")

plt.grid()

plt.show()

1. (10%) Using the same data from Question 2, compute the least square parabola (i.e. second order polynomial ) to fit the data. (5%) Explain which formulation (line or parabola) is more suitable for this dataset and why? (paste your source code and show the results in your report)



import scipy.io

import numpy as np

from numpy import ones

from numpy.linalg import inv

from matplotlib import pyplot as plt

mat = scipy.io.loadmat('data.mat')

# At\*A\*x = At\*b, x= [theta0, theta1] is the least square solution to ||Ax-b||

A = np.concatenate((ones([len(mat['x']),1]), mat['x'], mat['x']\*mat['x']), axis=1)

At = A.transpose()

At\_dot\_A = np.dot(At, A)

inverse\_At\_dot\_A = inv(At\_dot\_A)

At\_dot\_b = np.dot(At, mat['y'])

x = np.dot(inverse\_At\_dot\_A, At\_dot\_b)

theta0 = x[0][0]

theta1 = x[1][0]

theta2 = x[2][0]

print(theta0, theta1,theta2)

# compute least\_square\_parabola\_error

least\_square\_parabola\_error = 0

for i in range(len(mat['x'])):

least\_square\_parabola\_error += pow(abs(mat['y'][i][0] - (theta0 + mat['x'][i][0]\*theta1 + mat['x'][i][0]\*mat['x'][i][0]\*theta2)), 2)

print("least\_square\_parabola\_error: ")

print(least\_square\_parabola\_error)

# y = theta0 + x\*theta1 + x^2\*theta2, let's plot

plt.plot(mat['x'], mat['y'], 'o', ms=2, alpha=0.5, label='data')

plt.plot(mat['x'], theta0 + theta1\*mat['x'] + theta2\*mat['x']\*mat['x'], 'r', ms=1, alpha=0.8, label = 'polynomial2')

plt.legend(loc='best')

plt.xlabel("x")

plt.ylabel("y")

plt.title("Q3 fig, least\_square\_parabola\_error: %f, \n theta0: %f, theta1: %f, theta2: %f"%(least\_square\_parabola\_error, theta0, theta1, theta2))

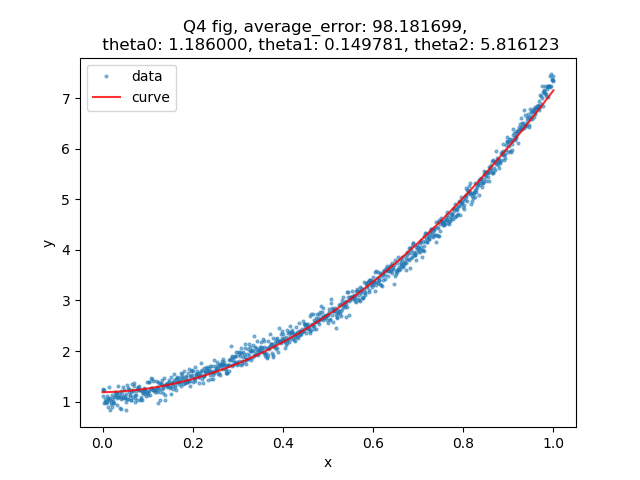
plt.savefig("./picture/hw1q3.png")

plt.grid()

plt.show()

ANS: parabola，because it presents lower error(15.760665<206.011773).

1. (20%) Using the same data from Question 2, now we use the loss function (L1 Norm) below instead of least square based methods. (paste your source code and show the results in your report) Hint: use a gradient descent approach.



import scipy.io

from matplotlib import pyplot as plt

mat = scipy.io.loadmat('data.mat')

# randomly picked initial value

theta0 = 2

theta1 = -1

theta2 = 4

lr = 0.0001

iteration = 10000

# record theta0, theta1, theta2 initial value for plotting

theta0\_history = [theta0]

theta1\_history = [theta1]

theta2\_history = [theta2]

# iteration

for i in range(iteration):

theta0\_gradient = 0.0

theta1\_gradient = 0.0

theta2\_gradient = 0.0

for n in range(len(mat['x'])):

if (mat['y'][n][0] - (theta0 + mat['x'][n][0] \* theta1 + mat['x'][n][0] \* mat['x'][n][0] \* theta2)) > 0:

theta0\_gradient -= 1

theta1\_gradient -= mat['x'][n][0]

theta2\_gradient -= mat['x'][n][0]\*mat['x'][n][0]

elif (mat['y'][n][0] - (theta0 + mat['x'][n][0] \* theta1 + mat['x'][n][0] \* mat['x'][n][0] \* theta2)) < 0:

theta0\_gradient += 1

theta1\_gradient += mat['x'][n][0]

theta2\_gradient += mat['x'][n][0]\*mat['x'][n][0]

else:

pass

# update theta0, theta1, theta2

theta0 -= lr\*theta0\_gradient

theta1 -= lr\*theta1\_gradient

theta2 -= lr\*theta2\_gradient

# record theta0, theta1, theta2 for plotting

theta0\_history.append(theta0)

theta1\_history.append(theta1)

theta2\_history.append(theta2)

print(theta0, theta1, theta2)

# compute least\_square\_parabola\_error

average\_error = 0.0

for i in range(len(mat['x'])):

average\_error += abs(mat['y'][i][0] - (theta0 + mat['x'][i][0]\*theta1 + mat['x'][i][0]\*mat['x'][i][0]\*theta2))

print("average\_error: ")

print(average\_error)

# plot

plt.plot(mat['x'], mat['y'], 'o', ms=2, alpha=0.5, label='data')

plt.plot(mat['x'], theta0 + mat['x']\*theta1 + mat['x']\*mat['x']\*theta2, 'r', ms=1, alpha=0.8, label='curve')

plt.legend(loc='best')

plt.xlabel("x")

plt.ylabel("y")

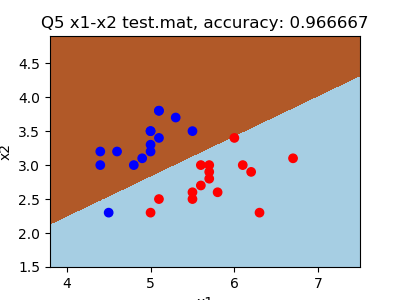
plt.title("Q4 fig, average\_error: %f, \n theta0: %f, theta1: %f, theta2: %f"%(average\_error, theta0, theta1, theta2))

plt.savefig("./picture/hw1q4.png")

plt.grid()

plt.show()

1. (25%) In ‘train.mat,’ you can find 2-D points X=[x1, x2] and their corresponding labels Y=y. Please use logistic regression to find the decision boundary (optimal ) based on ‘train.mat.” Report the test error on the test dataset ‘test.mat.’ (percentage of misclassified test samples) Hint: you can use “mnrfit” in Matlab or “LogisticRegression” in Python.



import matplotlib

import scipy.io

import numpy as np

from matplotlib import pyplot as plt

from sklearn.linear\_model import LogisticRegression

train = scipy.io.loadmat('train.mat')

test = scipy.io.loadmat('test.mat')

X\_train = np.concatenate((train['x1'], train['x2']), axis=1)

X\_test = np.concatenate((test['x1'], test['x2']), axis=1)

logreg = LogisticRegression()

logreg.fit(X\_train, train['y'])

x\_min, x\_max = train['x1'][:, 0].min() - 0.5, train['x1'][:, 0].max() + 0.5

y\_min, y\_max = train['x2'][:, 0].min() - 0.5, train['x2'][:, 0].max() + 0.5

h = 0.001

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))

Z = logreg.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.figure(1, figsize=(4,3))

plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

predict\_testY = logreg.predict(X\_test)

# count accuracy

correct\_count = 0

for i in range(len(test['y'])):

if test['y'][i][0] == predict\_testY[i]:

correct\_count += 1

accuracy = correct\_count/len(test['y'])

# plot x1,x2-y

colors=['red', 'blue']

plt.scatter(test['x1'], test['x2'], c=test['y'], cmap=matplotlib.colors.ListedColormap(colors))

# plt.plot(clf.predict(X\_test), ms=2, label="line")

plt.xlabel("x1")

plt.ylabel("x2")

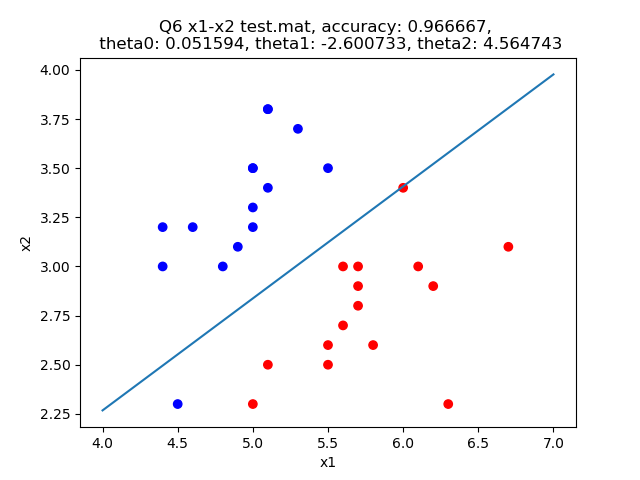
plt.title("Q5 x1-x2 test.mat, accuracy: %f"%(accuracy))

plt.savefig('./picture/hw1q5.png')

plt.grid()

plt.show()

1. (20%) Please use a gradient descent method to solve Question 5. (show your code, decision boundary, and test error on the test dataset)



import matplotlib

import scipy.io

from math import exp

from matplotlib import pyplot as plt

import numpy as np

train = scipy.io.loadmat('train.mat')

test = scipy.io.loadmat('test.mat')

def sigmoid(z):

return 1 / (1 + exp(-z))

### gradient descent

# initial value

theta0 = 0.0

theta1 = 0.0

theta2 = 0.0

lr = 0.0001

iteration = 10000

# iteration

for i in range(iteration):

theta0\_gradient = 0.0

theta1\_gradient = 0.0

theta2\_gradient = 0.0

for n in range(len(train['x1'])):

theta0\_gradient += (train['y'][n] - sigmoid(theta0 + theta1 \* train['x1'][n][0] + theta2 \* train['x2'][n][0]))

theta1\_gradient += (train['y'][n] - sigmoid(theta1 \* train['x1'][n][0] + theta2 \* train['x2'][n][0])) \* \

train['x1'][n]

theta2\_gradient += (train['y'][n] - sigmoid(theta1 \* train['x1'][n][0] + theta2 \* train['x2'][n][0])) \* \

train['x2'][n]

theta0 += lr \* theta0\_gradient

theta1 += lr \* theta1\_gradient

theta2 += lr \* theta2\_gradient

# accuracy

correct\_count=0

c1=0

c2=0

for i in range(len(test['x1'])):

if(theta0 + theta1\*test['x1'][i] + theta2\*test['x2'][i] > 0 ):

c1+=1

if test['y'][i] == 1:

correct\_count+=1

elif(theta0 + theta1\*test['x1'][i] + theta2\*test['x2'][i] < 0):

c2+=1

if test['y'][i] == 0:

correct\_count+=1

print(c1, c2, correct\_count)

print(theta0[0], theta1[0], theta2[0])

X1 = [4,7]

X2 = []

for i in range(len(X1)):

X2.append( ((-1)\*theta1\*X1[i] - theta0) / theta2 )

# plot x1,x2-y

colors=['red', 'blue']

plt.scatter(test['x1'], test['x2'], c=test['y'], cmap=matplotlib.colors.ListedColormap(colors))

plt.plot(X1, X2, ms=2, label="line")

plt.xlabel("x1")

plt.ylabel("x2")

plt.title("Q6 x1-x2 test.mat, accuracy: %f, \n theta0: %f, theta1: %f, theta2: %f" %(correct\_count/len(test['y']), theta0, theta1, theta2))

plt.savefig('./picture/hw1q6.png')

plt.grid()

plt.show()