ECE GY-6143 ML Homework 1 Solution

Yogya Sharma
(ys5250@nyu.edu)

1.

Polynomial regression with **d** as the degree of the polynomial:

$$f(x;\theta)=\theta_0 +\theta_1 +\theta_2 x^2 +...+\theta_d x^d$$

We minimize the risk to find the optimum value of d:

$$R_{emp}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}(y_i - f(x;\boldsymbol{\theta}))^2$$

So, to minimize the risk we need to find the value of θ (model parameters) which does it.

To do this we differentiate the empirical risk with respect to θ and find θ*:

$$\theta^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathrm{y}$$

To avoid overfitting, we split the data into two equal random halves. Here, we use the training half to train the model and the test half for cross validation.

In the code present below, we find the θ values for each value of **d** using the training data, and then find the minimum risk in that interval of **d**, using the test data.

The d value corresponding to the least test error is the answer to our question.

```
import scipy.io
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

#read the data as csv file
data = pd.read_csv("problem1.txt")
```

```
#extracting values of x and y in a list
x_data = data['x'].values.tolist()
y_data = data['y'].values.tolist()

#splitting the data for test and train
x_train, x_test, y_train, y_test =
train_test_split(x_data, y_data, random_state = 0,
test_size = 0.5)

#initializing the maximum value dor d
d = 40

#creating X zeros array for both test and train data
X_1 = np.zeros((len(x_train), d), dtype = float)
X_2 = np.zeros((len(x_test), d), dtype = float)

#initialising Y_Test and Y_Train arrays
yy_1 = np.asarray(y_train)
Y_Train = np.transpose([yy_1])
yy_2 = np.asarray(y_test)
Y_Test = np.transpose([yy_2])

#initializing a zeros array for model parmeters
T = np.zeros((len(x_train),1))

#updating the training X array till highest power (d)
of x values
for i in range(0, d):
  for j in range(len(x_train)):
    X_1[j][i] = x_train[j]**(i)

#Initializing the X_Train values as a dataframe
X_Train = pd.DataFrame(X_1)

#updating the test X array till highest power (d) of x
values
for k in range(0, d):
  for l in range(len(x_test)):
    X_2[l][k] = x_test[l]**k

#Initializing the X_Test values as a dataframe
X_Test = pd.DataFrame(X_2)

D = []
```

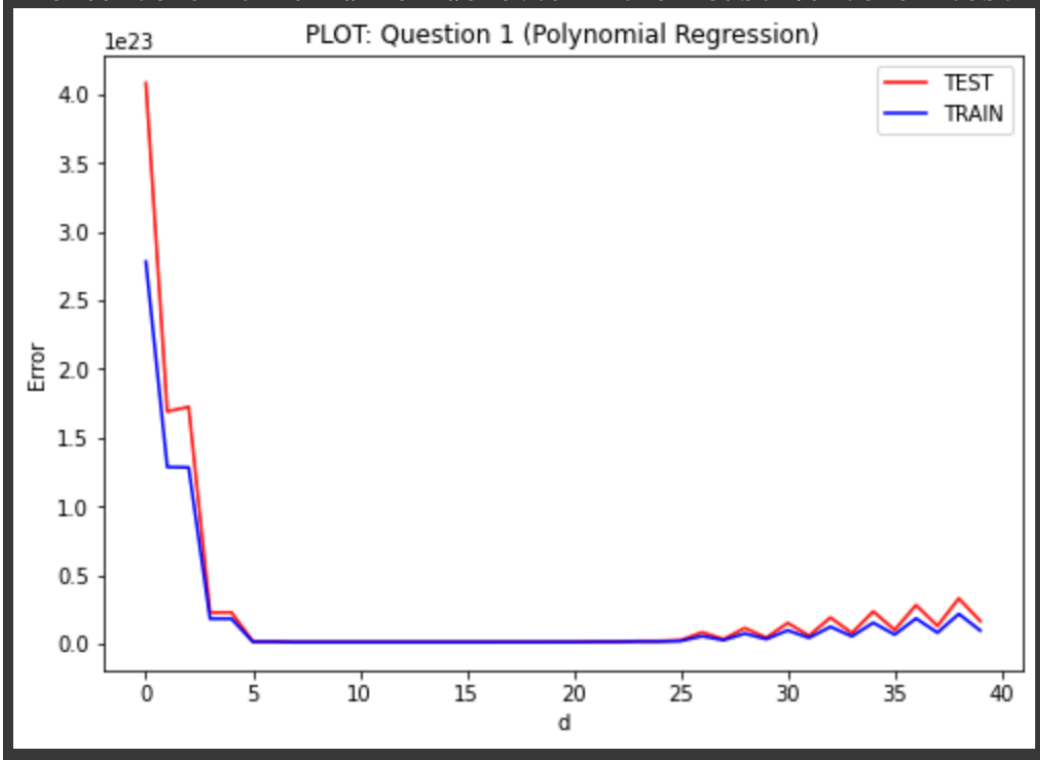
```
for n in range(0,d):
  D.append(n)

E_Test = []
E_Train = []

for p in range(0, d):

  #extracting the subarray from the X_Train dataframe
for each iteration   of p between 1 to d to calculate
model parameters
  X = X_Train.iloc[: , :(p+1)]

  #calculating inverse for calculating model parameters
  X_P = np.linalg.pinv(X)

  #calculating values of model parameters for the
current iteration
  T = np.dot(X_P, Y_Train)

  #extracting the subarray from the X_Test dataframe
for each iteration of p between 1 to d
  X_V = X_Test.iloc[: , :(p+1)]

  #calculating the y values for test data using the
model parameters calculated above to calculate test
error
  Y_V = np.dot(X_V, T)

  #extracting the subarray from the X_Test dataframe
for each iteration of p between 1 to d
  X_W = X_Train.iloc[: , :(p+1)]

  #calculating the y values for train data using the
model parameters calculated above to calculate train
error
  Y_W = np.dot(X_W, T)

  e_sq_1 = 0
  e_sq_2 = 0

  #calculating train and test errors
  for m in range (len(x_test)):
    e_sq_1 = e_sq_1 + (Y_Test[m] - Y_V[m])**2
```

```
  for n in range (len(x_train)):
    e_sq_2 = e_sq_2 + (Y_Train[n] - Y_W[n])**2

  e_sq_test = e_sq_1/(2*len(x_test))
  e_sq_train = e_sq_2/(2*len(x_train))

  E_Test.append(e_sq_test)
  E_Train.append(e_sq_train)

print("The minimum test error is: ", min(E_Test))
print("The value of d for which we obtain the least
value of test error is: ", E_Test.index(min(E_Test)))

#plotting the graph of error test and error train
plt.plot(D, E_Test, 'r')
plt.plot(D, E_Train, 'b')
plt.xlabel('d')
plt.ylabel('Error')
plt.title("PLOT: Question 1 (Polynomial Regression)")
plt.legend(['TEST', 'TRAIN'])

plt.show()
```

(Graph on the next page)

Following is the plot of test error vs. d:

```
The minimum test error is:  [1.22458902e+21]
The value of d for which we obtain the least value of test risk is:  7
```



In this graph we can see that test error (highlighted in red) first reduces and then starts increasing, while the training error remains relatively lower. The value of **d** where this reaches minimum is 7.

2.

Multi-variate regression function with l2 regularized risk minimization.

$$f(x;\boldsymbol{\theta}) \quad = \quad \sum_{i=1}^{k} \theta_i x_i$$

The empirical risk here becomes:

$$R_{\text{reg}}(\theta) \quad = \quad \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}(y_i - f(x;\boldsymbol{\theta}))^2 + \frac{\lambda}{2N}\|\boldsymbol{\theta}\|^2$$

When the gradient of this function becomes 0, for that value of θ we get the minimum risk. For this risk function the θ* comes as:

$$\theta^* = \left(\mathbf{X}^T\mathbf{X} + \lambda I\right)^{-1}\mathbf{X}^T\mathbf{y}$$

In the code below, to avoid overfitting, we split the data into two sections with test data being 30% of the total data. Here, we use the training data to train the model and the test data for cross validation.

We find the values of θ and corresponding test risk values. And the λ for which the test risk is minimum, is the optimal value of λ.

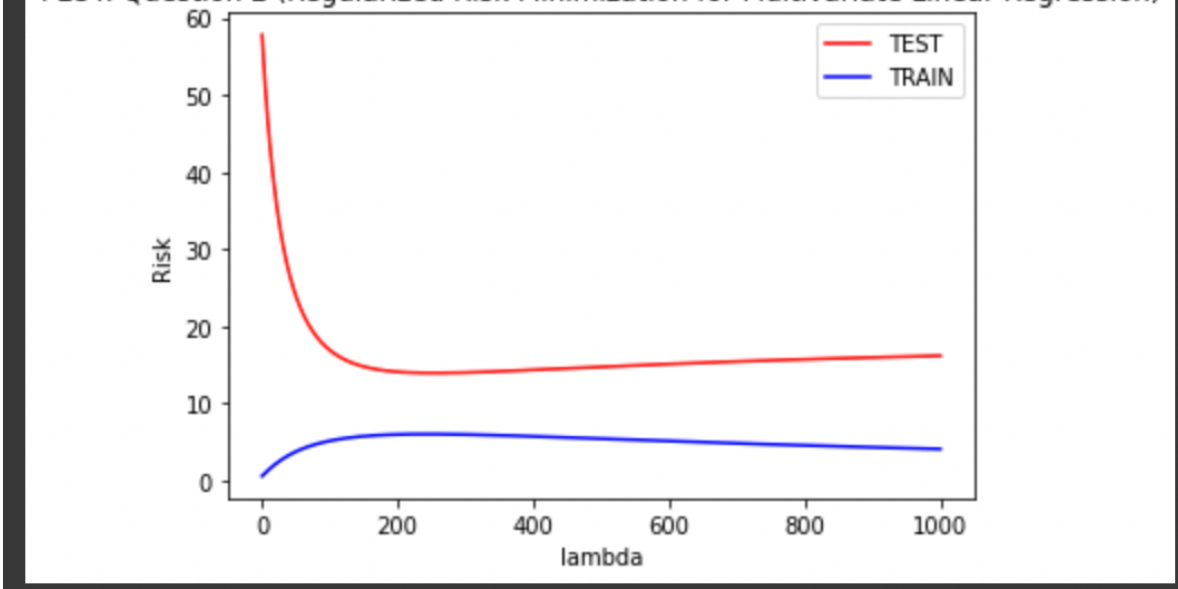
```
from scipy.io import loadmat
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

#reading the csv file
data = pd.read_csv("problem2_updated.txt")

#Getting the number of rows and columns in the data
rows = data.shape[0]
col  = data.shape[1]
```

```
#extracting the values of x parameters
X = data.iloc[ : , :(col-1)]

#extracting the values of y
Y_m = []
Y_m = data['y']
Y_n = np.asarray(Y_m)
Y = np.transpose([Y_m])

#initializing lambda
lamda = 1000

#splitting the data into data for train and test
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X,
Y, random_state = 90, test_size = 0.3)

X_T_Train = np.transpose(X_Train)

#Initializing the Identity matrix
I = np.identity(np.shape(X_Train)[1])

#creating two lists for appending test and train risks
for each value of lambda
E_Test = []
E_Train = []

L = []
for n in range(lamda+1):
  L.append(n)

#calculating model parameters and risks for each value
of lambda
for l in range(lamda+1):
  T = np.dot(np.dot(np.linalg.pinv(np.dot(X_T_Train ,
X_Train) + l*I) , X_T_Train), Y_Train)

  xTheta_Train = np.dot(X_Train, T)
  xTheta_Test = np.dot(X_Test, T)

  e_sq_train = ( np.square(np.sum(xTheta_Train -
Y_Train)) + (l * np.square(np.linalg.norm(T))) )/
(2*len(X_Train))
  e_sq_test = ( np.square(np.sum(xTheta_Test - Y_Test))
+ (l * np.square(np.linalg.norm(T))) ) /
(2*len(X_Test))
```

```
  E_Test.append(e_sq_test)
  E_Train.append(e_sq_train)

#printing the value of lambda for which test risk is
minimum
print("The value of lambda for which test risk is
minimum is:", E_Test.index(min(E_Test)))

#plotting the graph of test and train risk vs lambda
values
plt.plot(L, E_Test, 'r')
plt.plot(L, E_Train, 'b')
plt.xlabel('lambda')
plt.ylabel('Risk')
plt.title("PLOT: Question 2 (Regularized Risk
Minimization for Multivariate Linear Regression)")
plt.legend(['TEST', 'TRAIN'])

plt.show()
```

The plot below shows how the test and train risk vary across different λ values:

```
The value of lambda for which test risk is minimum is: 254
```



According to the graph and the values calculated in the code the test error is minimum for λ = 254

3.

3. $g(z) = 1/(1 + \exp(-z))$

- To show: $g(-z) = 1 - g(z)$

LHS: $g(-z) = 1 / (1 + \exp(-(-z)))$

$= 1 / (1 + \exp(z))$

RHS $= 1 - g(z)$

$= 1 - 1 / (1 + \exp(-z))$

$= \dfrac{(1 + \exp(-z) - 1)}{(1 + \exp(-z))}$

$= \dfrac{\exp(-z)}{(1 + \exp(-z))}$

multiplying both numerator and denominator by $\exp(z)$

$= \dfrac{\exp(z) \times \exp(-z)}{\exp(z) \times (1 + \exp(-z))}$

$$= \frac{1}{\exp(z) + 1}$$

$\because$ LHS = RHS, hence proved that $g(-z) = 1 - g(z)$

- To prove : $g^{-1}(y) = \ln(y/(1-y))$

given $y = g(z) = \frac{1}{1 + \exp(-z)}$

RHS : $\ln\left(\frac{y}{1-y}\right) = \ln(y) - \ln(1-y)$

$$= \ln\left(\frac{1}{1+\exp(-z)}\right) - \ln\left(1 - \frac{1}{1+\exp(-z)}\right)$$

$$= \ln\left(\frac{1}{1+\exp(-z)}\right) - \ln\left(\frac{\exp(-z)}{1+\exp(-z)}\right)$$

$$= \ln\left(\frac{1}{\exp(-z)}\right) = \ln(\exp(z))$$

$$= z$$

LHS : $g^{-1}(y) = g^{-1}(g(z))$

$$= z$$

LHS = RHS, hence proved that

$$g^{-1}(y) = \ln\left(\frac{y}{1-y}\right)$$

4.

For logistic regression the risk is given by:

$$R_{emp}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - 1)\log(1 - f(\mathbf{x}_i;\theta)) - y_i \log(f(\mathbf{x}_i;\theta))$$

To find the minimize the risk we need to find its gradient:

The gradient for R_{emp} is :

$$\nabla_\theta R_{emp} = \frac{-1}{N}\sum_{i=1}^{N}\left(y_i \frac{1}{f(x_i;\theta)} \frac{\partial}{\partial\theta} f(x_i;\theta) - (1-y_i)\frac{1}{1-f(x_i;\theta)}\frac{\partial}{\partial\theta} f(x_i;\theta)\right)$$

$$= -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \frac{1}{f(x_i;\theta)} - (1-y_i)\frac{1}{1-f(x_i;\theta)}\right)\frac{\partial}{\partial\theta} f(x_i;\theta)$$

$$= -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \frac{1}{g(\theta^T x)} - (1-y_i)\frac{1}{1-g(\theta^T x)}\right)\left(g(\theta^T x)\,(1-g(\theta^T x))\frac{\partial \theta^T x}{\partial\theta}\right)$$

$$= -\frac{1}{N}\sum_{i=1}^{N}\left((y_i \frac{1}{g(\theta^T x)} - (1-y_i)\frac{1}{1-g(\theta^T x)}\right)\left(g(\theta^T x)(1-g(\theta^T x))x\right)$$

$$= -\frac{1}{N}\sum_{i=1}^{N}(y_i(1-g(\theta^T x)) - (1-y_i)g(\theta^T x))x$$

$$= -\frac{1}{N}\sum_{i=1}^{N}(y - g(\theta^T x))x$$

To find the optimal value of θ* here, we use batch gradient descent. Here the initial value of θ is a random number. The gradient descent goes as follows:

$$\theta^1 = \theta^0 - \eta \nabla_\theta R_{emp}\Big|_{\theta^0}, \quad t = 1$$

$$\textit{while} \left\| \theta^t - \theta^{t-1} \right\| \geq \in \quad \{$$

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta R_{emp}\Big|_{\theta^t}, \quad t = t+1$$

$$\}$$

Here, η is the learning rate of the gradient descent and ε is the tolerance. The iterations stop once $\theta^t - \theta^{t-1}$ goes below the tolerance ε.

After that we plot the scatter plot of the data and along with the calculated θ* values plot the decision boundary.

```
 from scipy.io import loadmat
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import math

#reading the csv file
data = pd.read_csv("problem4.txt")

#getting the number of rows and columns in the data
rows = data.shape[0]
col  = data.shape[1]

#extracting the value of X parameters
X = data.iloc[ : , :(col-1)]

#extracting the value of y
Y_m = []
Y_m = data['y']
Y_n = np.asarray(Y_m)
Y = np.transpose([Y_m])

#re-structuring the X matrix to get x3 as the first
column (bias term)
X = X.iloc[: , [2,0,1]]
```

```
#initializing tolerance
e = 0.003

#initializing step-size
n = 0.05

#assigning random values for model parameters
theta = [[1],
         [1],
         [1]]

#initialing another matrix with all zero values with
same dimensions as theta
theta_t = np.zeros((3, 1))

#declaring the list to append risk values for each
iteration
J = list()

#declaring the list to append binary classification
error for each iteration
binaryClassificationError = list()

#converting the X dataframe to an array
X_m = np.asarray(X)
diff_max = 1

#initialinzing iteration counter to zero
count = 0

#running the while loop till when the difference
between to concurrent model parameters is higher than
the tolerance
while abs(diff_max) >= e:
  diff = 0

  z = np.dot(X, theta)

  err = 0

  w = pd.Series([z])

  predictions = list()

  #Calculating binary classification error
```

```
  for k in range(200):
    if w[0][k][0]>= 0:
      predictions.append(1)
    else:
      predictions.append(0)

  t = 0
  for g in range(len(Y)):
    if predictions[g] != Y[g]:
      t = t +1

  #calculating and appending the value of binary
classification error
  binaryClassificationError.append(t/len(X))

  #calculating risk
  for i in range(0, len(X)):
    err = err + Y[i] * np.log(1/(1+np.exp(-z[i]))) +
(1-Y[i]) * np.log(1 - (1/(1+np.exp(-z[i]))))

  risk = (-1 * err) / len(X)

  J.append(risk)

  #iterating for gradient descent to obtain successive
values for model parameters
  p = list()
  for j in range(0, 3):
    sum = 0
    for k in range(len(X)):
      sum = sum + ((1/(1+((math.e)**(-z[k])))) - Y[k])
* X_m[k][j]
    p.append((n * sum)[0])
    theta_t[j] = theta[j] - n * sum

  diff_max = min(p, key = abs)
  theta = theta_t
  count = count + 1

iterations  = count

print("Final values of model parameters")
print(theta)
print("Total number of iterations:", count)
```

```
#Plotting scatter plot for y values
figure, hlt = plt.subplots(1,3)
for r in range(0, 2):
    row = np.where(data.y == r)
    hlt[0].scatter( data.iloc[row[0],0],
data.iloc[row[0], 1], s=1 )

hlt[0].set_title("Scatter Plot for Training Data")
hlt[0].set_xlabel("x1")
hlt[0].set_ylabel("x2")
hlt[0].grid()

#plotting decision boundary
decisionBoundary = - (theta[1]/theta[2]) * X.iloc[:,1]
- (theta[0]/theta[2])
hlt[0].plot(X.iloc[:,1], decisionBoundary)

#plotting risk vs. iterations
hlt[1].plot(range(iterations), J, 'r')
hlt[1].set_title("Risk vs. Number of Iterations")
hlt[1].set_xlabel("Number of Iterations")
hlt[1].set_ylabel("Risk")

#plotting binary classification error vs. iterations
hlt[2].plot(range(iterations),
binaryClassificationError, 'b')
hlt[2].set_title("Binary Classification Error vs.
Number of Iterations")
hlt[2].set_xlabel("Number of Iterations")
hlt[2].set_ylabel("Binary Classification Error")

plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=2.0,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)


plt.grid()
plt.show()
```

In the code we set η (denoted by n in the code) as 0.05 and ε (denoted by e in the code) as 0.003.

We calculate θ* vector using these values which comes out to be:

[-19.5882357 49.81816511 27.35693786]

Using these values, we plot the decision boundary as:

$$\Theta^T x = 0$$

Here this will be,

[-19.5882357 49.81816511 27.35693786] [x0 = 0

x1

x2]

as all values in x0 = 1, it is the bias term.

[-19.5882357 49.81816511 27.35693786] [1 = 0

x1

x2]

-19.5882357 + 49.81816511*x1 + 27.35693786*x2 = 0

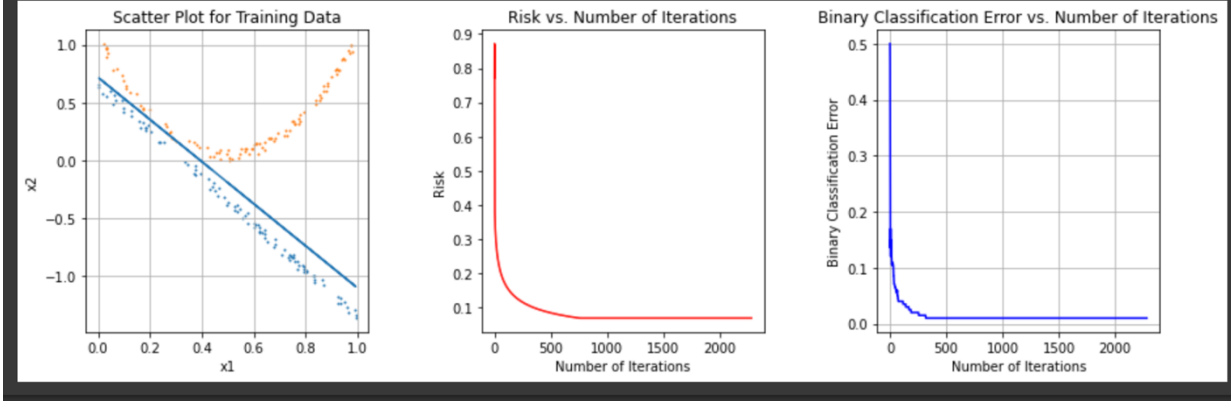
So, this is our decision boundary:

x2 = -1.82104312 * x1 + 0.71602443

(Graphs on the next page)

Following are the plots for decision boundary, risk vs. iterations and binary classification error vs. iterations:

```
Final values of model parameters
[[-19.5882357 ]
 [ 49.81816511]
 [ 27.35693786]]
Total number of iterations: 2285
```



The program takes 2285 iterations to reach convergence with 100% accuracy for which the decision boundary is plotted above. A soon as the binary classification error reaches 0, the iterations stop.