CS 613 - Machine Learning

Assignment 1 - Regression - Abishek S Kumar

Solutions

This is the solutions pdf file of the Assignment 1, all solutions are provided after each question, NOTE: the source code is also listed with the answers, along with the source code from jupyter-notebook in the assoziated zip file

Datasets

Fish Length Dataset (x06Simple.csv) This dataset consists of 44 rows of data each of the form:

1. Index
2. Age (days)
3. Temperature of Water (degrees Celsius)
4. Length of Fish

The first row of the data contains header information.

Data obtained from: http://people.sc.fsu.edu/ jburkardt/datasets/regression/regression.html

1 Theory

1. (10pts) Consider the following data:

$$\begin{bmatrix} -2 & 1 \\ -5 & -4 \\ -3 & 1 \\ 0 & 3 \\ -6 & 11 \\ -2 & 5 \\ 1 & 0 \\ 5 & -1 \\ -1 & -3 \\ 3 & 1 \end{bmatrix}$$

   (a) Compute the coefficients for the linear regression using least squares estimate (LSE), where the second value (column) is the dependent variable (the value to be predicted) and the first column is the sole feature. Show your work and remember to add a bias feature and to standardize the features. Compute this model using **all** of the data (don't worry about separating into training and testing sets).

   (b) Confirm your coefficient and intercept term using the sklearn.linear_model LinearRegression function.

2. For the function $g(x) = (x-1)^4$, where x is a single value (not a vector or matrix):

   (a) (3pts) What is the gradient with respect to x? Show your work to support your answer.

   (b) (3pts) What is the global minima for $g(x)$? Show your work to support your answer.

   (c) (3pts) Plot x vs $g(x)$ using matplotlib and use this image in your report.

Answers to Question 1

1. The following are the values of coefficients and intercept calculated for the given data, for both parts 'a' and 'b':

$$\theta = \begin{bmatrix} -1.426399 & 1.4 \end{bmatrix}$$

   (a) Working: The mean and standard deviation of the data is as follows:

$$X(mean) = \text{-1.0},\ X(standarddeviation) = 3.224903$$

   The standardized data set based on the function

$$X(standard(i)) = X(i) - X(mean)/X(standarddeviation)$$

   The standardized data set and appending a bias feature is as follows:

$$\begin{bmatrix} -0.31008684 & 1. \\ -1.24034735 & 1. \\ -0.62017367 & 1. \\ 0.31008684 & 1. \\ -1.55043418 & 1. \\ -0.31008684 & 1. \\ 0.62017367 & 1. \\ 1.86052102 & 1. \\ 0. & 1. \\ 1.24034735 & 1. \end{bmatrix}$$

   Next calculate regression parameters using the equation:

$$\theta = (X^T X)^{-}1X^T Y \text{ where; } \theta = \begin{bmatrix} \theta_1 & \theta_0 \end{bmatrix} \text{ therefore } \theta = \begin{bmatrix} -1.426399 & 1.4 \end{bmatrix}$$

   The values calculated from the inbuilt function *sklearn.linear_model* is; coeff: -1.4263995 and intercept: 1.4

2. The following is the work related to calculating the gradient and global minima:

   (a) Gradient of $g(x) = (x-1)^4$ is $g'(x) = 4(x-1)^3 * (1)$

   (b) Calculating global minima : set gradient to zero i.e. $g'(x) = 0$ and $4(x-1)^3 * (1) = 0$ the value of x at $g'(x) = 0$ is 1 which at the global minima.

   (c) Gradient plot $g(x)$ vs. x:

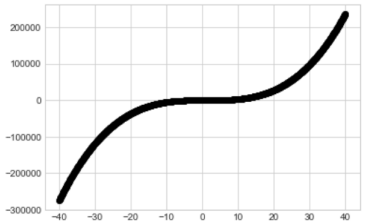

Figure 1: Gradient plot

Source Code from Jupyter Notebook for Question 1:

```
import numpy as np
from sklearn.linear_model import LinearRegression as linreg
X = [-2, -5, -3, 0, -6, -2, 1, 5, -1, 3]
Y = [1, -4, 1, 3, 11, 5, 0, -1, -3, 1]

X_mean = np.mean(W)
X_std = np.std(W)
print(X_std)
print(X_mean)

W = np.mat(X)
Z = np.mat(Y)
W = W.reshape(10,1)
Z = Z.reshape(10,1)
W_standard = (W - X_mean)/X_std
print(W)
print(Z)

W_standard = np.mat(W_standard)
W_standard = W_standard.reshape(10,1)
print(W_standard.shape)

ones = np.ones(W_standard.shape[0])
ones_append = (np.reshape(np.asarray(ones), (ones.shape[0], 1)))
print(ones.shape)
W_append = np.append( W_standard, ones_append, axis = 1)
print(W_append)

W_stand_tr = np.transpose(W_append)
print(W_stand_tr)

first_dot_inv = np.linalg.inv(np.dot(W_stand_tr , W_append))
print(first_dot_inv)
second_dot = np.dot(W_stand_tr, Z)
print(second_dot)
third_dot = np.dot(first_dot_inv, second_dot)
print(third_dot)

verify = linreg().fit(W_standard, Z)
print(verify.coef_)
print(verify.intercept_)

x = np.linspace(-40, 40, 1000)
y = 4*((x-1)**3)
plt.plot(x,y,'o',color='black')
```

2 Closed Form Linear Regression

Download the dataset *x06Simple.csv* from Blackboard. This dataset has header information in its first row and then all subsequent rows are in the format:

$$ROWId, x_{i,1}, x_{i,2}, y_i$$

Your code should work on any CSV data set that has the first column be header information, the first column be some integer index, then D columns of real-valued features, and then ending with a target value.

Write a script that:

1. Reads in the data, ignoring the first row (header) and first column (index).
2. Randomizes the data
3. Selects the first 2/3 (round up) of the data for training and the remaining for testing
4. Standardizes the data (except for the last column of course) using the training data
5. Computes the closed-form solution of linear regression
6. Applies the solution to the testing samples
7. Computes the *root mean squared error* (RMSE): $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2}$. where $\hat{Y}_i$ is the predicted value for observation X_i.

Implementation Details

1. Seed the random number generate with zero prior to randomizing the data
2. Don't forget to add in a bias feature!

In your report you will need:

1. The final model in the form $y = \theta_0 + \theta_1 x_{:,1} + ...$
2. The root mean squared error.

Answers to Question 2

1. The final model along with coefficients is as follows: $y = 3343.375x_0 + 1018.600x_{:,1} - 290.526x_{:,2}$ wherein the values of theta are:
   (a) $\theta_0 = 3343.275$
   (b) $\theta_1 = 1018.600$
   (c) $\theta_2 = -290.526$
2. The RMSE calculated is; RMSE = 663.58478

Source Code from Jupyter Notebook for Question 2:

```
import csv
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression as linreg
new_csv = pd.read_csv('x06Simple.csv')

x = new_csv.values.tolist()

y = np.asarray(x)

temp = np.transpose(y)

temp = np.transpose(temp [1:])

np.random.seed(0)
np.random.shuffle(temp)


temp_tr = np.transpose(temp)

##print(temp_first2)
temp_first2_tr = np.transpose(temp_tr [0:2])
temp_last_col_tr = np.transpose(temp_tr[2])
temp_first2_tr = np.reshape(temp_first2_tr , (temp_first2_tr.shape[0],temp_first2_tr.shape[1
temp_last_col_tr = np.reshape(temp_last_col_tr, (temp_last_col_tr.shape[0], 1))

training_data = temp_first2_tr[0:(np.int((0.66)*temp_first2_tr.shape[0]))][:]
print(training_data.shape)

testing_data = temp_first2_tr[(np.int((0.66)*temp_first2_tr.shape[0])):][:]
#print(testing_data)

training_last_col = temp_last_col_tr[0:(np.int((0.66)*temp_last_col_tr.shape[0]))]
testing_last_col  = temp_last_col_tr[(np.int((0.66)*temp_first2_tr.shape[0])):]
```

```
training_data_temp = np.transpose(training_data)
##print(training_data_temp)
testing_data_temp = np.transpose(testing_data)


training_mean = np.transpose(np.apply_along_axis(np.mean, axis=1, arr=training_data_temp))
training_std = np.transpose(np.apply_along_axis(np.std, axis=1, arr=training_data_temp))

testing_mean = np.transpose(np.apply_along_axis(np.mean, axis=1, arr=testing_data_temp))
testing_std = np.transpose(np.apply_along_axis(np.std, axis=1, arr=testing_data_temp))

training_data_temp_1 = [(training_data_temp[i] - training_mean[i])/training_std[i] for i in
testing_data_temp_1 =  [(testing_data_temp[i] - testing_mean[i])/testing_std[i] for i in ra

training_data_temp_1 = np.transpose(np.asarray(training_data_temp_1))
training_for_verify = training_data_temp_1
testing_data_temp_1 = np.transpose(np.asarray(testing_data_temp_1))

ones = np.ones(training_data_temp_1.shape[0])
ones_append = (np.reshape(np.asarray(ones), (ones.shape[0], 1)))

ones_test = np.ones(testing_data_temp_1.shape[0])
ones_append_test = (np.reshape(np.asarray(ones_test), (ones_test.shape[0], 1)))
training_data_temp_1 = np.append( training_data_temp_1, ones_append, axis = 1)
testing_data_temp_1  = np.append( testing_data_temp_1, ones_append_test, axis = 1)
#print(training_data_temp_1)
print(testing_data_temp_1)

training_data_temp_tr = np.transpose(training_data_temp_1)
#print(training_data_temp_tr)

first_dot_inv = np.linalg.inv(np.dot(training_data_temp_tr, training_data_temp_1))
#print(first_dot_inv)
second_dot = np.dot(training_data_temp_tr, training_last_col)
#print(second_dot)
third_dot = np.dot(first_dot_inv, second_dot)
print(third_dot)

verify = linreg().fit(training_for_verify, training_last_col)
print(verify.coef_)
print(verify.intercept_)

print(testing_data_temp_1.shape)
print(third_dot.shape)

predicted = (testing_data_temp_1.dot(third_dot))
print(predicted.shape)
```

```
print(testing_last_col.shape)

sigma = np.sum((testing_last_col - predicted)**2)

RMSE = np.sqrt(sigma/training_data.shape[0])
print(RMSE)
```

3 S-Folds Cross-Validation

Cross-Validation is a technique used to get reliable evaluation results when we don't have that much data (and it is therefore difficult to train and/or test a model reliably).

In this section you will do S-Folds Cross-Validation for a few different values of S. For each run you will divide your data up into S parts (folds) and test S different models using S-Folds Cross-Validation and evaluate via root mean squared error. In addition, to observe the affect of system variance, we will repeat these experiments several times (shuffling the data each time prior to creating the folds). We will again be doing our experiment on the provided fish dataset. **You may use sklearn KFold** to perform this task.

Write a script that:

1. Reads in the data, ignoring the first row (header) and first column (index).
2. 20 times does the following:
   (a) Randomizes the data
   (b) Creates S folds.
   (c) For $i = 1$ to S
      i. Select fold i as your testing data and the remaining $(S-1)$ folds as your training data
      ii. Standardizes the data (except for the last column of course) based on the training data
      iii. Train a closed-form linear regression model
      iv. Compute the squared error for each sample in the current testing fold
   (d) You should now have N squared errors. Compute the RMSE for these.
3. You should now have 20 RMSE values. Compute the mean and standard deviation of these. The former should give us a better "overall" mean, whereas the latter should give us feel for the variance of the models that were created.

Implementation Details

1. Don't forget to add a bias feature!
2. Set your seed value at the very beginning of your script (if you set it within the 20 tests, each test will have the same randomly shuffled data!).

In your report you will need:

1. The average and standard deviation of the root mean squared error for $S = 3$ over the 20 different seed values..
2. The average and standard deviation of the root mean squared error for $S = 5$ over the 20 different seed values.

3. The average and standard deviation of the root mean squared error for $S = 20$ over 20 different seed values.

4. The average and standard deviation of the root mean squared error for $S = N$ (where N is the number of samples) over 20 different seed values. This is basically *leave-one-out* cross-validation.

Answers to Question 3

1. The average and standard deviation of the root mean squared error for $S = 3$ over the 20 different seed values: *MEAN*= 626.4075 and *STD. DEVIATION*= 21.8106.

2. The average and standard deviation of the root mean squared error for $S = 5$ over the 20 different seed values: *MEAN*= 631.76942106 and *STD. DEVIATION*= 21.899.

3. The average and standard deviation of the root mean squared error for $S = 20$ over 20 different seed values: *MEAN*= 623.3760 and *STD. DEVIATION*= 8.90240.

4. The average and standard deviation of the root mean squared error for $S = N$ (where N is the number of samples) over 20 different seed values: *MEAN*= 623.40513 and *STD. DEVIATION*= 1.28779139e-11.

Source Code from Jupyter Notebook for Question 3:

```
import csv
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression as linreg
from sklearn.model_selection import KFold
new_csv = pd.read_csv('x06Simple.csv')
#print(new_csv)
x = new_csv.values.tolist()

y = np.asarray(x)
#print(y)
temp = np.transpose(y)

temp = np.transpose(temp [1:])
#print(temp)

X = []

for i in range(20):
    np.random.seed(i)
    np.random.shuffle(temp)
    #print(temp)
    kf = KFold(n_splits = 44)
    kf.get_n_splits(temp)
    #print(kf)
    sigma_sq_error = []
    for train_index, test_index in kf.split(temp):

        temp_train, temp_test = temp[train_index], temp[test_index]
        #print("TRAIN:", temp_train, "TEST:", temp_test)
        #y_train, y_test = y[train_index], y[test_index]
        #temp_train = np.asarray(temp[train])
```

```
#temp_test = np.asarray(temp[test])
temp_train_tr = np.transpose(temp_train)
temp_test_tr  = np.transpose(temp_test)

#print(temp_train_tr.shape)
#print(temp_test_tr.shape)
temp_first2_train_tr = np.transpose(temp_train_tr [0:2])
temp_last_col_train_tr = np.transpose(temp_train_tr[2])
temp_first2_test_tr = np.transpose(temp_test_tr [0:2])
temp_last_col_test_tr = np.transpose(temp_test_tr[2])
#print(temp_first2_test_tr.shape)
#print(temp_last_col_test_tr.shape)

##training samples matrix creation x & y
temp_first2_train_tr = np.reshape(temp_first2_train_tr , (temp_first2_train_tr.shape
temp_last_col_train_tr = np.reshape(temp_last_col_train_tr, (temp_last_col_train_tr
#print(temp_first2_train_tr)
#print(temp_last_col_train_tr.shape)
training_data_temp = np.transpose(temp_first2_train_tr)
training_mean = np.transpose(np.apply_along_axis(np.mean, axis=0, arr=training_data_
training_std = np.transpose(np.apply_along_axis(np.std, axis=0, arr=training_data_te
#print(training_mean)
#print(training_std)
training_data_temp_1 = [(training_data_temp[i] - training_mean[i])/training_std[i] f
training_data_temp_1 = np.transpose(np.asarray(training_data_temp_1))
training_for_verify = training_data_temp_1
#print(training_for_verify)

ones_train = np.ones(training_data_temp_1.shape[0])
ones_train_append = (np.reshape(np.asarray(ones_train), (ones_train.shape[0], 1)))
training_data_temp_1 = np.append( training_data_temp_1, ones_train_append, axis = 1)
#print(training_data_temp_1)
training_data_temp_tr = np.transpose(training_data_temp_1)
first_dot_inv = np.linalg.inv(np.dot(training_data_temp_tr, training_data_temp_1))
#print(first_dot_inv)
second_dot = np.dot(training_data_temp_tr, temp_last_col_train_tr)
#print(second_dot)
third_dot = np.dot(first_dot_inv, second_dot)
#print(third_dot)
verify = linreg().fit(training_for_verify, temp_last_col_train_tr)
#print(verify.coef_)
#print(verify.intercept_)

##testing samples matrix creation x & y
temp_first2_test_tr = np.reshape(temp_first2_test_tr , (temp_first2_test_tr.shape[0]
temp_last_col_test_tr = np.reshape(temp_last_col_test_tr, (temp_last_col_test_tr.sha
#print(temp_first2_test_tr)
#print(temp_last_col_test_tr.shape)
```

```
        testing_data_temp = np.transpose(temp_first2_test_tr)
        #print(testing_data_temp)
        ##check the line below
        testing_mean = np.transpose(np.apply_along_axis(np.mean, axis=0, arr=testing_data_te
        testing_std = np.transpose(np.apply_along_axis(np.std, axis=0, arr=testing_data_temp
        #print(testing_mean)
        #print(testing_std)
        testing_data_temp_1 =  [(testing_data_temp[i] - training_mean[i])/training_std[i] fo

        #print(testing_data_temp_1)
        testing_data_temp_1 = np.transpose(np.asarray(testing_data_temp_1))
        #print(testing_data_temp_1)
        ones_test = np.ones(testing_data_temp_1.shape[0])
        ones_append_test = (np.reshape(np.asarray(ones_test), (ones_test.shape[0], 1)))
        testing_data_temp_1  = np.append( testing_data_temp_1, ones_append_test, axis = 1)
        #print(testing_data_temp_1)

        ##calculate RMSE
        predicted = (testing_data_temp_1.dot(third_dot))
        #print(predicted)

        #print(temp_last_col_test_tr.shape)

        sq_errors = (temp_last_col_test_tr - predicted)**2
        #print(sq_errors.shape)
        sigma_sq_error = sigma_sq_error + sq_errors.tolist()
        #print(sigma_sq_error.shape)
    RMSE = np.sqrt(np.sum(sigma_sq_error)/len(sigma_sq_error))
    X.append(RMSE)


#print(X)
X = np.asarray(X)

X = np.reshape(X, (X.shape[0], 1))
#print(X.shape)
X_mean = np.transpose(np.apply_along_axis(np.mean, axis=0, arr=X))
X_std = np.transpose(np.apply_along_axis(np.std, axis=0, arr=X))
print(X_mean)
print(X_std)
```

4 Locally-Weighted Linear Regression

Next we'll do locally-weighted closed-form linear regression. You may use **sklearn train_test_split** for this part.

Write a script to:

1. Read in the data, ignoring the first row (header) and first column (index).
2. Randomize the data
3. Select the first 2/3 of the data for training and the remaining for testing
4. Standardize the data (except for the last column of course) using the training data
5. Then for each *testing sample*
   (a) Compute the necessary distance matrices relative to the training data in order to compute a local model.
   (b) Evaluate the testing sample using the local model.
   (c) Compute the squared error of the testing sample.
6. Computes the *root mean squared error* (RMSE): $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2}$. where $\hat{Y}_i$ is the predicted value for observation X_i.

Implementation Details

1. Seed the random number generate with zero prior to randomizing the data
2. Don't forget to add in the bias feature!
3. Use the L1 distance when computing the distances $d(a, b)$.
4. Let $k = 1$ in the similarity function $\beta(a, b) = e^{-d(a,b)/k^2}$.
5. Use *all* training instances when computing the local model.

In your report you will need:

1. The root mean squared error.

Answers to Question 4

1. The root mean squared error; $RMSE = 320.480679$.

Source Code from Jupyter Notebook for Question 3:

```
import csv
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression as linreg
new_csv = pd.read_csv('x06Simple.csv')

x = new_csv.values.tolist()

y = np.asarray(x)

temp = np.transpose(y)

temp = np.transpose(temp [1:])

np.random.seed(0)
np.random.shuffle(temp)


temp_tr = np.transpose(temp)

##print(temp_first2)
temp_first2_tr = np.transpose(temp_tr [0:2])
temp_last_col_tr = np.transpose(temp_tr[2])
temp_first2_tr = np.reshape(temp_first2_tr , (temp_first2_tr.shape[0],temp_first2_tr.shape[1
temp_last_col_tr = np.reshape(temp_last_col_tr, (temp_last_col_tr.shape[0], 1))

training_data = temp_first2_tr[0:(np.int((0.66)*temp_first2_tr.shape[0]))][:]
#print(training_data.shape)

testing_data = temp_first2_tr[(np.int((0.66)*temp_first2_tr.shape[0])):][:]
#print(testing_data)

training_last_col = temp_last_col_tr[0:(np.int((0.66)*temp_last_col_tr.shape[0]))]
testing_last_col  = temp_last_col_tr[(np.int((0.66)*temp_first2_tr.shape[0])):]
#print(testing_last_col)

training_data_temp = np.transpose(training_data)
##print(training_data_temp)
testing_data_temp = np.transpose(testing_data)
#print(testing_data_temp)

training_mean = np.transpose(np.apply_along_axis(np.mean, axis=1, arr=training_data_temp))
```

```
#print(training_mean)
training_std = np.transpose(np.apply_along_axis(np.std, axis=1, arr=training_data_temp))

testing_mean = np.transpose(np.apply_along_axis(np.mean, axis=1, arr=testing_data_temp))
testing_std = np.transpose(np.apply_along_axis(np.std, axis=1, arr=testing_data_temp))

training_data_temp_std = [(training_data_temp[i] - training_mean[i])/training_std[i] for i i
testing_data_temp_std =  [(testing_data_temp[i] - training_mean[i])/training_std[i] for i in



training_data_temp_std = np.transpose(np.asarray(training_data_temp_std))
#print(training_data_temp_std)

testing_data_temp_std = np.transpose(np.asarray(testing_data_temp_std))
#print(testing_data_temp_std)

ones = np.ones(training_data_temp_std.shape[0])
ones_append = (np.reshape(np.asarray(ones), (ones.shape[0], 1)))

ones_test = np.ones(testing_data_temp_std.shape[0])
ones_append_test = (np.reshape(np.asarray(ones_test), (ones_test.shape[0], 1)))
training_data_appendOnes = np.append( training_data_temp_std, ones_append, axis = 1)
testing_data_appendOnes  = np.append( testing_data_temp_std, ones_append_test, axis = 1)
#print(training_data_temp_std)
#print(testing_data_temp_std)
#print(training_data_appendOnes)
#print(testing_data_appendOnes)

##calculate distance matrices for each feature's test samples

#print(training_data_temp[0])
#print(training_data_temp[1])
#print(training_data_temp.shape)
#print(testing_data_temp[0])
#print(testing_data_temp[1])
#print(testing_data_temp.shape)
##for every feature set in the testing sample, i.e. two rows
RMSE = []

for i in range (testing_data_appendOnes.shape[0]):## along each test sample of each feature
        test_cap =  testing_data_appendOnes[i,:]

        test_cap = np.reshape(test_cap , (test_cap.shape[0],1))
        #print(y_cap)
        distance_matrix = np.zeros([training_data_temp.shape[1]])
        identity_matrix = np.identity(training_data_temp.shape[1])
        #print(distance_matrix.shape)
```

```
        for j in range (training_data_appendOnes.shape[0]):##get every training sample from

            #print(i, "test", testing_data_temp[x][i])
            #print(j, "train", training_data_temp[x][i])
            temp_dist = np.linalg.norm(testing_data_appendOnes[i] - training_data_appendOnes
            #print(x, i, j, temp_dist)
            #print(np.exp(-temp_dist))
            distance_matrix[j] = np.exp(-temp_dist)
            #print(distance_matrix)
        Weight_matrix = distance_matrix * identity_matrix
        #print(Weight_matrix)
        ##calculate theta value for each test sample in each feature set i.e. x = 0 -> 1, i
        first_dot_inv = np.linalg.inv(np.dot(np.dot((training_data_appendOnes.T), Weight_mat
        #first_dot_inv = np.linalg.inv((training_data_appendOnes.T)*Weight_matrix*(training_
        #print(first_dot_inv)
        second_dot = np.dot(np.dot((training_data_appendOnes.T), Weight_matrix), training_la
        #print(second_dot)
        third_dot = np.dot(first_dot_inv, second_dot)
        print(third_dot.shape)
        y_cap = np.dot(testing_data_appendOnes[i], third_dot)
        sq_error = (testing_last_col[i] - y_cap)**2
        RMSE.append(sq_error)

RMSE = np.sqrt(np.sum(RMSE)/len(RMSE))
print(RMSE)
```

5 Gradient Descent

As discussed in class Gradient Descent (Ascent) is a general algorithm that allows us to converge on local minima (maxima) when a closed-form solution is not available or is not feasible to compute.

In this section you are going to implement a gradient descent algorithm to find the parameters for linear regression on the same data used for the previous sections. You may **NOT** use any function for a ML library to do this for you, except **sklearn train_test_split** for the data.

Implementation Details

1. Seed the random number generator prior to your algorithm.
2. Don't forget to a bias feature!
3. Initialize the parameters of θ using random values in the range [-1, 1]
4. Do **batch** gradient descent
5. Terminate when absolute value of the percent change in the RMSE on the **training** data is less than 2^{-23}, or after $1,000$ iterations have passed (whichever occurs first).
6. Use a learning rate $\eta = 0.01$.
7. Make sure that you code can work for an arbitrary number of observations and an arbitrary number of features.

Write a script that:

1. Reads in the data, ignoring the first row (header) and first column (index).
2. Randomizes the data
3. Selects the first 2/3 (round up) of the data for training and the remaining for testing
4. Standardizes the data (except for the last column of course) base on the training data
5. While the termination criteria (mentioned above in the implementation details) hasn't been met
   (a) Compute the RMSE of the *training* data
   (b) While we can't let the testing set affect our training process, also compute the RMSE of the testing error at each iteration of the algorithm (it'll be interesting to see).
   (c) Update each parameter using *batch* gradient descent
6. Compute the RMSE of the testing data.

What you will need for your report

1. Final model
2. A graph of the RMSE if the *training* and *testing* sets as a function of the iteration
3. The final RMSE *testing* error.

Answers to Question 5

1. The randomized values of test theta values in the range [-1,+1] as as follows: $\theta_0 = 0.96193473$ $\theta_1 = -0.17339805$ $\theta_2 = 0.67247727$,

   The final model along with coefficients is as follows: $y = 3343.375x_0 + 1018.600x_{:,1} - 290.526x_{:,2}$ wherein the values of theta are:

   (a) $\theta_0 = 3343.13154718$

   (b) $\theta_1 = 1018.55711808$

   (c) $\theta_2 = -290.51610259$

2. Insert plot of RMSE oftraining and testing sets here:

3. The RMSE calculated is; RMSE = 653.7026937

4. The graphs for Training data vs. RMSE and Testing data vs. RMSE

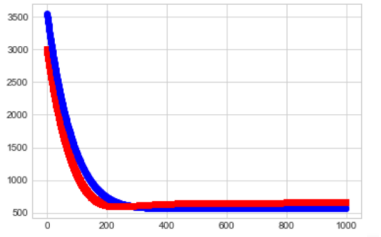

Figure 2: Training data vs. RMSE (blue) and Testing data vs. RMSE (red)

Source Code from Jupyter Notebook for Question 3:

```
import csv
import pandas as pd
import numpy as np
import random
from sklearn.linear_model import LinearRegression as linreg
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn-whitegrid')

new_csv = pd.read_csv('x06Simple.csv')

x = new_csv.values.tolist()

y = np.asarray(x)

temp = np.transpose(y)

temp = np.transpose(temp [1:])
```

```
np.random.seed(0)
np.random.shuffle(temp)


temp_tr = np.transpose(temp)

##print(temp_first2)
temp_first2_tr = np.transpose(temp_tr [0:2])
temp_last_col_tr = np.transpose(temp_tr[2])
temp_first2_tr = np.reshape(temp_first2_tr , (temp_first2_tr.shape[0],temp_first2_tr.shape[1
temp_last_col_tr = np.reshape(temp_last_col_tr, (temp_last_col_tr.shape[0], 1))

training_data = temp_first2_tr[0:(np.int((0.66)*temp_first2_tr.shape[0]))][:]
#print(training_data.shape)

testing_data = temp_first2_tr[(np.int((0.66)*temp_first2_tr.shape[0])):][:]
#print(testing_data)

training_last_col = temp_last_col_tr[0:(np.int((0.66)*temp_last_col_tr.shape[0]))]
testing_last_col  = temp_last_col_tr[(np.int((0.66)*temp_first2_tr.shape[0])):]
#print(training_last_col)

training_data_temp = np.transpose(training_data)
##print(training_data_temp)
testing_data_temp = np.transpose(testing_data)
#print(testing_data_temp)

training_mean = np.transpose(np.apply_along_axis(np.mean, axis=1, arr=training_data_temp))
#print(training_mean)
training_std = np.transpose(np.apply_along_axis(np.std, axis=1, arr=training_data_temp))

testing_mean = np.transpose(np.apply_along_axis(np.mean, axis=1, arr=testing_data_temp))
testing_std = np.transpose(np.apply_along_axis(np.std, axis=1, arr=testing_data_temp))

training_data_temp_std = [(training_data_temp[i] - training_mean[i])/training_std[i] for i i
testing_data_temp_std =  [(testing_data_temp[i] - training_mean[i])/training_std[i] for i in

training_data_temp_std = np.transpose(np.asarray(training_data_temp_std))
#print(training_data_temp_std)

testing_data_temp_std = np.transpose(np.asarray(testing_data_temp_std))
#print(testing_data_temp_std)

ones = np.ones(training_data_temp_std.shape[0])
ones_append = (np.reshape(np.asarray(ones), (ones.shape[0], 1)))

ones_test = np.ones(testing_data_temp_std.shape[0])
ones_append_test = (np.reshape(np.asarray(ones_test), (ones_test.shape[0], 1)))
```

```
training_data_appendOnes = np.append( training_data_temp_std, ones_append, axis = 1)
testing_data_appendOnes  = np.append( testing_data_temp_std, ones_append_test, axis = 1)
#print(training_data_temp_std)
#print(testing_data_temp_std)
print(training_data_appendOnes.shape)
print(testing_data_appendOnes.shape)

##set random values for theta values between -1 and 1
#theta_temp = np.asarray([np.random.uniform(-1, 1) for i in range(training_data_appendOnes.s
#theta_parameters = np.reshape(theta_temp, training_data_appendOnes.shape[1],1)
theta_parameters = np.reshape(np.asarray([random.uniform(-1, 1) for i in range(training_data
print(theta_parameters)

eta_value = 0.01
for i in range(1000):
    #print(i)
    #predicted_values = np.dot(training_data_appendOnes, theta_parameters)
    #predicted_values = np.reshape(np.asarray(predicted_values), (training_data_appendOnes.s
    predicted_values = training_data_appendOnes.dot(theta_parameters)
    sq_error = (training_last_col - predicted_values)**2
    predicted_values_test = testing_data_appendOnes.dot(theta_parameters)
    sq_error_test = (testing_last_col - predicted_values_test)**2
    #print(predicted_values.shape)
    #print(sq_error.shape)
    sigma = np.sum(sq_error)
    RMSE = np.sqrt(sigma/len(sq_error))
    sigma_test = np.sum(sq_error_test)
    RMSE_test = np.sqrt(sigma_test/len(sq_error_test))
    #print(RMSE)
    plt.plot(i, RMSE, 'bo')
    plt.plot(i, RMSE_test, 'rs')
    if RMSE <= (2**(-23)):
        break
    #Grad_temp = (predicted_values - training_last_col)
    #print(Grad_temp)
    #Gradient_values = np.dot(training_data_appendOnes, (Grad_temp)*(eta_value/training_data
    Gradient_values = training_data_appendOnes.T.dot(predicted_values - training_last_col)*(
    theta_parameters = theta_parameters - Gradient_values
    print(theta_parameters)
    prediction_gradients = np.dot(training_data_appendOnes, theta_parameters)
    #print(prediction_gradients)
    sq_error_2 = (training_last_col - prediction_gradients)**2
    sigma_2 = np.sum(sq_error_2)
    RMSE_2 = np.sqrt(sigma_2/len(sq_error_2))
    plt.plot(i, RMSE_2, 'rs')


predicted_values_test = np.dot(testing_data_appendOnes, theta_parameters)
```

```
sq_error_3 = (testing_last_col - predicted_values_test)**2
sigma_3 = np.sum(sq_error_3)
test_RMSE = np.sqrt(sigma_3/len(sq_error_3))
print(test_RMSE)
```