CS 613 - Machine Learning

Assignment 1 - Regression

John Obuch

Introduction

In this assignment you will perform linear regression on a dataset and using cross-validation to analyze your results. In addition to computing and applying the close-form solution, you will also implement from scratch a gradient descent algorithm for linear regression.

As with all homeworks, you cannot use any functions that are against the "spirit" of the assignment, unless explicitly told to do so. For this assignment that would mean any linear regression functions. You *may* use statistical and linear algebra functions to do things like:

- mean
- std
- cov
- inverse
- matrix multiplication
- transpose
- etc...

Grading

Part 1 (Theory)	19pts
Part 2 (Closed-form LR)	25pts
Part 3 (S-folds LR)	10pts
Part 4 (Local LR)	10pts
Part 5 (GD LR)	25pts
Report	11pts
TOTAL	100

Table 1: Grading Rubric

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).

Response:

We start by referencing the general form:

$$Y = mx + b + \epsilon$$

.

Utilizing matrix notation, we know that:

$$\vec{\theta} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y}$$

.

Given the data above, let:

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

We can write $\mathbf{X}$ as follows (including bias term):

$$\mathbf{X} = \begin{bmatrix} 1 & -2 \\ 1 & -5 \\ 1 & -3 \\ 1 & 0 \\ 1 & -6 \\ 1 & -2 \\ 1 & 1 \\ 1 & 5 \\ 1 & 1 \\ 1 & 3 \end{bmatrix}$$ Similarly, $\mathbf{Y} = \begin{bmatrix} 1 \\ -4 \\ 1 \\ 3 \\ 11 \\ 5 \\ 0 \\ -1 \\ -3 \\ 1 \end{bmatrix}$, and $\mathbf{X}^{\mathrm{T}} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -2 & -5 & -3 & 0 & -6 & -2 & 1 & 5 & -1 & 3 \end{bmatrix}$.

Next, we need to standardize $\mathbf{X}$. We do this with the following transformation:

$$\widetilde{\mathbf{X}} = \frac{\mathbf{X} - \bar{\mathbf{X}}}{\mathbf{S}}$$

.

Where $\mathbf{X}$ is the feature vector, $\mathbf{S} = \sqrt{\frac{\sum(\mathbf{X}-\bar{\mathbf{X}})^2}{N-1}}$ is the sample standard deviation, $\bar{\mathbf{X}}$ is a vector where all the elements are the average value of the feature vector, and $N = 10$ is the total number of records.

After some computation, we yield the following:

$\bar{\mathbf{X}} = -1$, $\mathbf{S} = 3.39934$, $\widetilde{\mathbf{X}} = \begin{bmatrix} 1 & -0.294 \\ 1 & -1.177 \\ 1 & -0.588 \\ 1 & 0.294 \\ 1 & -1.471 \\ 1 & -0.294 \\ 1 & 0.588 \\ 1 & 1.765 \\ 1 & 0 \\ 1 & 1.177 \end{bmatrix}$, and

$\widetilde{\mathbf{X}}^{\mathrm{T}} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -0.294 & -1.177 & -0.588 & 0.294 & -1.471 & -0.294 & 0.588 & 1.765 & 0 & 1.177 \end{bmatrix}$.

Performing matrix multiplication, we get that:

$$\widetilde{\mathbf{X}}^{\mathrm{T}}\widetilde{\mathbf{X}} = \begin{bmatrix} 10 & 0 \\ 0 & 9 \end{bmatrix}$$

.

Next, we desire to obtain the computational result of $(\widetilde{\mathbf{X}}^{\mathrm{T}}\widetilde{\mathbf{X}})^{-1}$. **Note:** If $\mathbf{A}$ is a 2 x 2 matrix in the form $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$, then $\mathbf{A}^{-1} = \frac{1}{\det(\mathbf{A})}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$, where $\det(\mathbf{A}) = ad - bc$.

Computing $(\widetilde{\mathbf{X}}^{\mathrm{T}}\widetilde{\mathbf{X}})^{-1}$, we yield the following:

$$\det(\widetilde{\mathbf{X}}^{\mathrm{T}}\widetilde{\mathbf{X}}) = 90$$

. Thus, leveraging the trick defined above, we have that:

$$(\widetilde{\mathbf{X}}^{\mathrm{T}}\widetilde{\mathbf{X}})^{-1} = \frac{1}{90}\begin{bmatrix}9 & 0\\ 0 & 10\end{bmatrix} = \begin{bmatrix}\frac{1}{10} & 0\\ 0 & \frac{1}{9}\end{bmatrix}$$

Next, we compute $\widetilde{\mathbf{X}}^{\mathrm{T}}\mathbf{Y} = \begin{bmatrix}14\\ -13.531\end{bmatrix}$.

We now have everything we need to compute the parameters θ_i for $i = 0, 1$.

Recall that:

$$\vec{\theta} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y}$$

In our case, using our standardized notation, we have that:

$$\vec{\theta} = (\widetilde{\mathbf{X}}^{\mathrm{T}}\widetilde{\mathbf{X}})^{-1}\widetilde{\mathbf{X}}^{\mathrm{T}}\mathbf{Y} = \begin{bmatrix}0.10 & 0\\ 0 & 0.111\end{bmatrix}\begin{bmatrix}14\\ -13.531\end{bmatrix} \approx \begin{bmatrix}1.40\\ -1.502\end{bmatrix} = \begin{bmatrix}\theta_0\\ \theta_1\end{bmatrix}$$

Therefore, utilizing least squares estimate (LSE), our coefficients for the linear regression equation are:

$\theta_0 \approx 1.40$ and $\theta_1 \approx -1.50$

Thus, we have that:

$$\hat{Y} = -1.50X + 1.40$$

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

Utilizing the sklearn.linear_model Linear Regression function, the results of the intercept and coefficient are as follows (*See source code for further detail*):

$\theta_0 = 1.4000$, and $\theta_1 = -1.503557$.

Note: Comparing these results to the mathematical results outlined above in (a), we observe equivalent results.

■

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.

Given that:

$$g(x) = (x-1)^4$$

Using the power rule and chain rule we have:

$$g'(x) = 4(x-1)^3$$

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

$$g'(x) = 4(x-1)^3 = 0$$
$$\implies (x-1)^3 = 0$$
$$\implies x = 1$$

Plugging this result into our original equation $g(x)$, we obtain our y coordinate:

$$g(1) = (1-1)^4 = 0$$

.

Thus, the global minima (or critical point) coordinate is at the point $(x, y) = (1, 0)$.

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

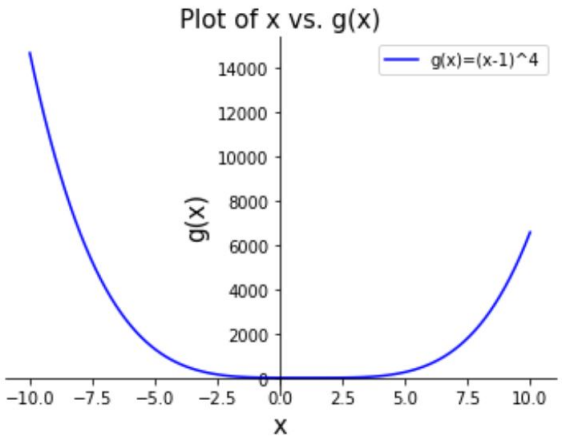


Figure 1: Plot of x vs $g(x)$.

See source code for further detail

■

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.

Results:

1. $\hat{\mathbf{Y}} = 3275.6667 + 1097.6031\mathbf{X_1} - 259.3279\mathbf{X_2}$
2. RMSE = 624.6526

See source code for further detail

■

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.

Results: For each fold S, we obtain an average Root Mean Squared Error (RMSE) notated $\text{RMSE}_{\bar{X}}$ and a Standard Deviation of the RMSE notated RMSE_{σ}. What follows are the results for each fold:

1. $S = 3$, $\text{RMSE}_{\bar{X}} = 637.89999$, and $\text{RMSE}_{\sigma} = 30.55483$.

2. $S = 5$, $\text{RMSE}_{\bar{X}} = 627.15291$, and $\text{RMSE}_{\sigma} = 14.42182$.

3. $S = 20$, $\text{RMSE}_{\bar{X}} = 622.32346$, and $\text{RMSE}_{\sigma} = 6.37933$.

4. $S = N = 44$, $\text{RMSE}_{\bar{X}} = 623.40514$, and $\text{RMSE}_{\sigma} = 3.595\,09 \times 10^{-14}$

See source code for further detail

■

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.

Result:

1. RMSE = 521.20822

See source code for further detail

■

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.

Results:

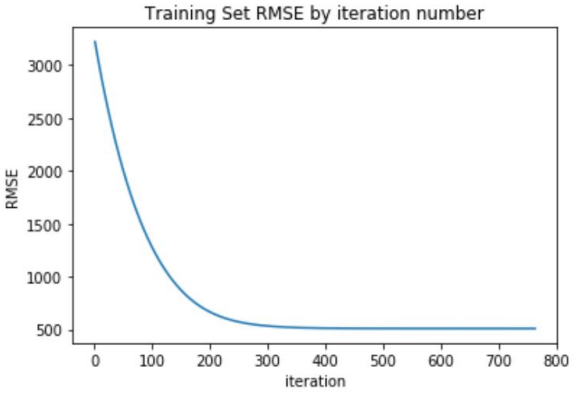
1. $\hat{\mathbf{Y}} = 2917.56239 + 1221.23801\mathbf{X_1} + -278.77198\mathbf{X_2}$
2. 

Figure 2: Plot of iteration number vs RMSE for training data.

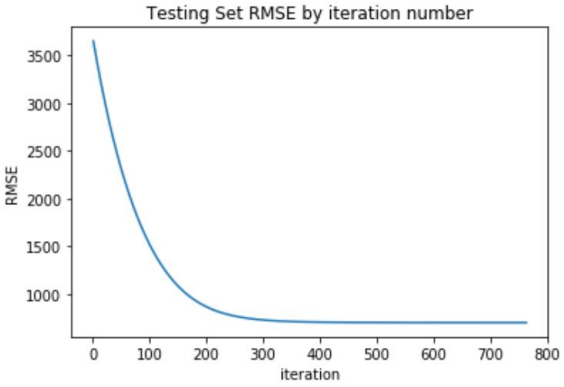


Figure 3: Plot of iteration number vs RMSE for testing data.

3. $\text{RMSE}_{testing} = 703.60948$

See source code for further detail

■

Submission

For your submission, upload to Blackboard a single zip file containing:

1. PDF Writeup
2. Source Code Jupyter Notebook

The PDF document should contain the following:

1. Part 1:
   (a) Your solutions to the theory question
2. Part 2:
   (a) Final Model
   (b) RMSE
3. Part 3:
   (a) Mean and Standard Deviations of RMSE for different values of S.
4. Part 4:
   (a) RMSE
5. Part 5:
   (a) Final Model
   (b) RMSE
   (c) Plot of RMSE for Training and Testing Data vs Gradient Descent iteration number