ECE M146 | Homework 2 Solution
Introduction to Machine Learning | Monday, April 5, 2021
Instructor: Lara Dolecek | Due: Monday, April 12, 2021
TA: Zehui (Alex) Chen | chen1046@ucla.edu

Please upload your homework to Gradescope by April 12, 4:00 pm.
Please submit a single PDF directly on Gradescope
You may type your homework or scan your handwritten version. Make sure all the work is discernible.

1. In class, we learned that the solution of the least square problem satisfies the normal equation $X^T X w = X^T y$, where $X \in \mathbb{R}^{N\times M}$ and $y \in \mathbb{R}^N$. For this normal equation to have a unique solution, $X^T X$ has to be nonsingular. Prove the following statement about the necessary and sufficient conditions of the nonsingularity of $X^T X$:

   The Gram matrix $X^T X$ is nonsingular if and only if X has linearly independent columns.

   Hint: A matrix A has linearly independent columns if $Ax = 0$ only for $x = 0$. You may want to use the following properties of (non)singular matrix. The equation $Ax = 0$ has only the trivial solution $x = 0$ if and only if A is nonsingular. If A is singular, there exist $z \neq 0$ such that $Az = 0$.

   In our set-up, with $N >> M$, there will likely be M linearly independent vectors x_n. Therefore, the Gram matrix $X^T X$ is likely to be invertible.

   Solution:

   - Suppose X has linearly independent columns:

   $$\begin{aligned} X^T X z = 0 &\implies z^T X^T X z = (Xz)^T (Xz) = \|Xz\|^2 = 0 \\ &\implies Xz = 0 \\ &\implies z = 0. \end{aligned}$$

     Therefore $X^T X$ is nonsingular. The last statement comes from X has linearly independent columns.

   - Suppose the columns of X are linearly dependent:

   $$\exists z \neq 0, Xz = 0 \implies \exists z \neq 0, X^T X z = 0.$$

     Therefore $X^T X$ is singular.

2. Consider the hat matrix $H = X(X^T X)^{-1}X^T$, where $X \in \mathbb{R}^{N\times M}$ and $X^T X$ is invertible. We encountered this hat matrix when we solved the linear least squares regression problem. The vector $w = (X^T X)^{-1}X^T y$ led to the prediction $\hat{y} = Xw = X(X^T X)^{-1}X^T y = Hy$. In other words, this hat matrix H describes the linear transformation between the target y (true label) and the estimation $\hat{y}$. We will prove several properties of H in this question.

   (a) Show that H is symmetric.

   (b) Show that $H^K = H$ for any positive integer K.

   (c) If I is the identity matrix of size N, show that $(I - H)^K = I - H$ for any positive integer K.

   (d) Show that $Trace(H) = M$, where the trace is the sum of diagonal elements. Hint:$Trace(AB) = Trace(BA)$.

   Solution:

   (a) $H^T = X[(X^T X)^{-1}]^T X^T = X(X^T X)^{-1}X^T = H$.

   (b) For K=2, $H^2 = X(X^T X)^{-1}X^T X(X^T X)^{-1}X^T = X(X^T X)^{-1}X^T = H$. For K = k-1, we have $H^{k-1} = H$, then $H^k = H^2 = H$. By induction, $H^K = H$ for any positive integer K.

   (c) Similar to (b), it is sufficient to show for $K = 2$, $(I-H)^2 = I^2 - 2H + H^2 = I - H$.

   (d) We use the property of trace in the second step:

$$\begin{aligned} Trace(H) &= Trace(X(X^T X)^{-1}X^T) \\ &= Trace(X^T X(X^T X)^{-1}) \\ &= Trace(I_M) \\ &= M. \end{aligned}$$

3. Consider a linear regression problem in which we want to "weigh" different training instances differently because some of these instances are more important than others. Specifically, suppose we want to minimize

$$J(w_0, w_1) = \sum_{n=1}^{N} \alpha_n (w_0 + w_1 x_n - y_n)^2. \tag{1}$$

Here $\alpha_n > 0$; w_0 and w_1 are weights; x_n's and y_n's are scalars. In class we worked out what happens for the case where all weights (the α_n's) are the same. In this problem, we will generalize some of those ideas to the weighted setting.

(a) Calculate the gradient by computing the partial derivative of J with respect to each of the parameters w_0 and w_1.

(b) Comment on how the $\alpha_n's$ affect the linear regression problem. For example, what happens if $\alpha_i = 0$ for some i, $1 \le i \le N$?

(c) Qualitatively describe what will happen when performing gradient descent if one α_j is much greater than all other $\alpha_{j'}$'s, $j' \neq j$, $1 \le j, j' \le N$.

Solution:

(a)

$$\nabla_{\boldsymbol{w}} J(w_0, w_1) = \begin{bmatrix} \frac{\partial J(w_0,w_1)}{\partial w_0} \\ \frac{\partial J(w_0,w_1)}{\partial w_1} \end{bmatrix} = \begin{bmatrix} \sum_{n=1}^{N} 2\alpha_n (w_0 + w_1 x_n - y_n) \\ \sum_{n=1}^{N} 2\alpha_n x_n (w_0 + w_1 x_n - y_n) \end{bmatrix}.$$

(b) If $\alpha_i = 0$ for some i, the data $\{x_i, y_i\}$ is irrelevant to this problem.

(c) If α_j is much greater than other $\alpha_{j'}$, in gradient descent algorithm, a small difference between $w_0 + w_1 x_j$ and y_j will cause the algorithm to take a large step. As a result, the GD algorithm tends to converge to a line that passes through $\{x_j, y_j\}$.

4. Consider the following objective function:

$$J(w) = \frac{1}{2}\|w\|^2 + C\left[\frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y_i(w^T x_i))\right],$$

where C is a constant, $x_i \in \mathbb{R}^M$ and $y_i \in \{-1, +1\}$. We will encounter a similar loss function when we study support vector machine (SVM). Derive the stochastic gradient descent (SGD) rule to minimize this loss function.
Solution: We first rewrite the objective function as:

$$J(w) = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{2}\|w\|^2 + C \times \max(0, 1 - y_i(w^T x_i))\right].$$

The gradient with respect to w is then calculated:

$$\nabla_w J(w) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases} w, & \text{if} \quad 1 - y_i w^T x_i \le 0, \\ w - C y_i x_i, & \text{if} \quad 1 - y_i w^T x_i > 0. \end{cases}$$

When performing SGD, only a single data point is considered at a time so we update w based on the following rule:

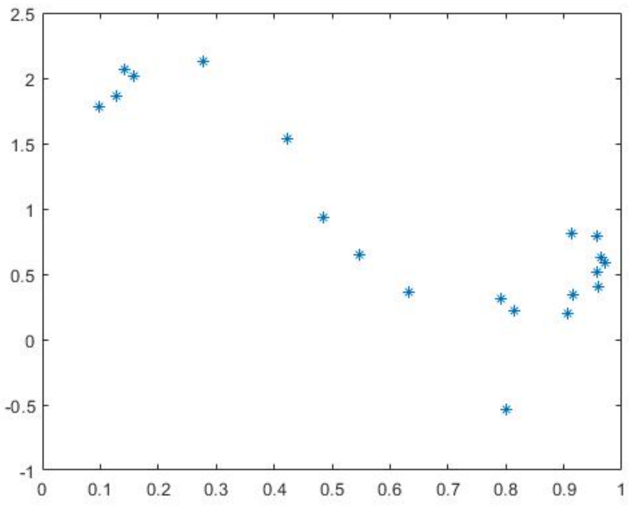
$$w := w - \eta\begin{cases} w, & \text{if} \quad 1 - y_i w^T x_i \le 0, \\ w - C y_i x_i, & \text{if} \quad 1 - y_i w^T x_i > 0, \end{cases}$$

where η is the learning rate. Both answers with and without the normalization by norm are correct.

5. In this exercise, you will implement approaches to solve the linear and polynomial regression problems. Our data consists of inputs $x_n \in \mathbb{R}$ and outputs $y_n \in \mathbb{R}, n \in \{1, \cdots, N\}$, which are related through a target function $f(x)$. Your goal is to learn a predictor $h_w(x)$ that best approximates $f(x)$.

   (a) **Visualization** We provide two sets of data, the training data and the testing data in the two files, *regression_train.csv* and *regression_test.csv*. In each file, the first column is the input and the second column is the output. In MATLAB (or python), visualize the training data by plotting the input vs. the output. What do you observe? For example, can you make an educated guess on the effectiveness of linear regression in predicting the data?

   Solution: We can see a linear trend on the figure so linear regression is likely to



   fit the data well. However, a higher order polynomial may fit the data better.

   (b) **Linear Regression: closed-form solution** Let us start by considering a simple linear regression model:

$$h_w(x) = w^T x = w_0 + w_1 x.$$

   Recall that linear regression attempts to minimize the objective function

$$J(w) = \sum_{n=1}^{N} (h_w(x_n) - y_n)^2 = \|Xw - y\|^2,$$

   where

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, X = \begin{bmatrix} 1, x_1 \\ 1, x_2 \\ \vdots, \vdots \\ 1, x_n \end{bmatrix}, w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}.$$

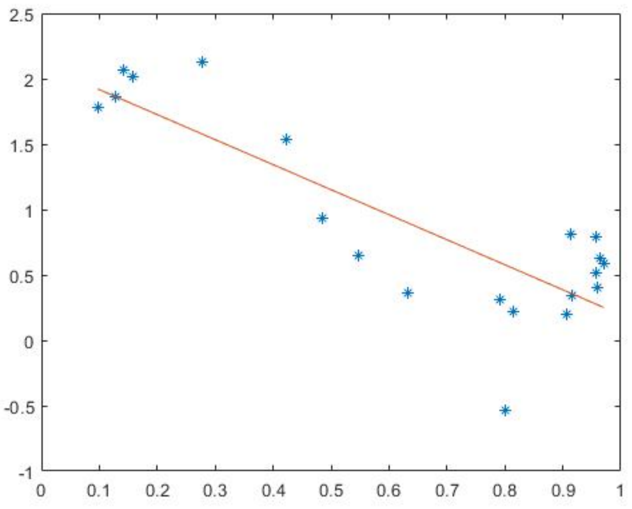
Note that to take into account the intercept term w_0, we can add an additional "feature" to each instance and set it to one, e.g., $x_{i,0} = 1$. This step is equivalent to adding an additional first column to X and setting it to all ones.

In class we learned that the closed-form solution to linear regression is:

$$w^* = (X^T X)^{-1} X^T y.$$

Implement this closed-form solution in MATLAB (or python) using the training data and report w^* and $J(w)$. Also generate a plot depicting your training data and the fitted line.

Solution: $w^* = [2.0034, -1.8052]^T$ and $J(w) = 3.26$.



(c) **Linear Regression: gradient descent** Another way to solve linear regression is through gradient descent (GD). In gradient descent, each iteration performs the following update:

$$w_j := w_j - \eta \sum_{n=1}^{N} (h_w(x_n) - y_n) x_{n,j} \quad \text{(simultaneously update } w_j \text{ for all } j).$$

With each iteration of gradient descent, we expect our updated parameters w to come closer to the optimal w^* and therefore achieve a lower value of $J(w)$.

Implement the gradient descent algorithm in MATLAB (or python) using all of the following specifications for the gradient descent algorithm:

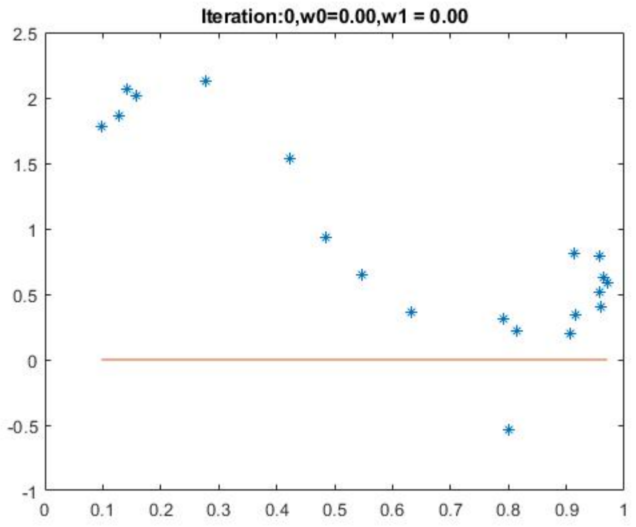
- Initialize w to be the all 0's vector.
- Run the algorithm for 10000 iterations.
- Terminate the algorithm earlier if the value of $J(w)$ is unchanged across consecutive iterations. In this exercise, we say $J(w)$ is unchanged if $|J(w)_{t-1} - J(w)_t| < 0.0001$.

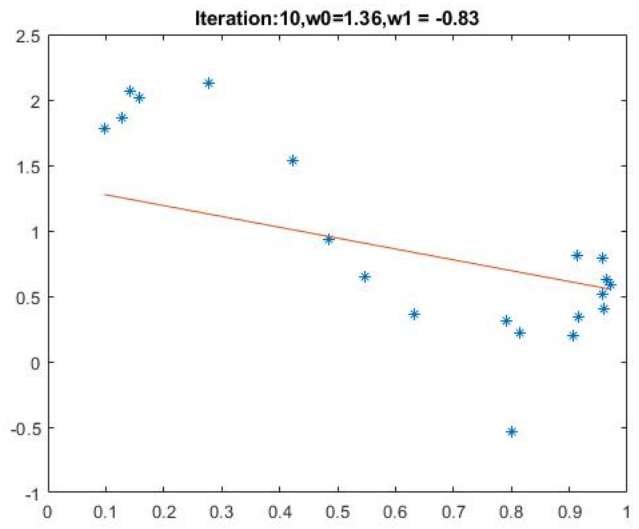
- Use a fixed learning rate η.

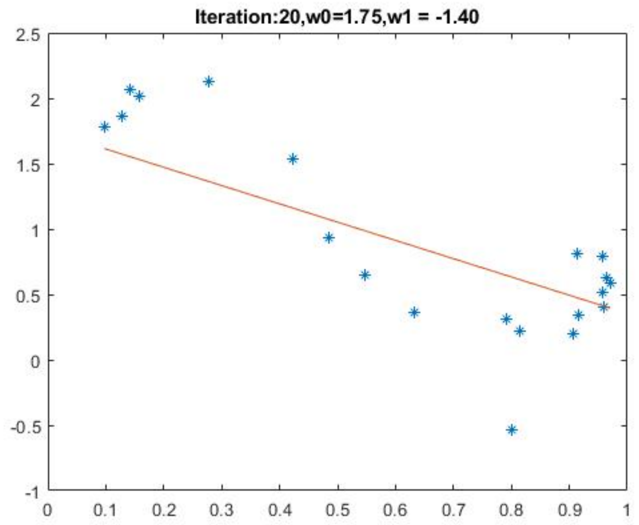
For different $\eta = 0.05, 0.001, 0.0001, 0.00001$, report the final $J(w)$ and the number of iterations until convergence (the number will be 10000 if the algorithm did not converge within 10000 iterations). Discuss how does the learning rate η affect the GD algorithm.

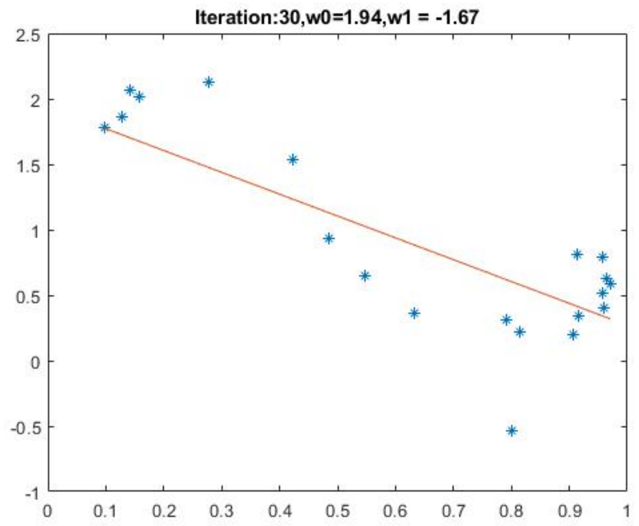
Solution: For $\eta = 0.05, 0.001, 0.0001$ and 0.00001, the # of iterations are $66, 2009$, 10000 and 10000; the $J(w)$ is $3.2605, 3.2947, 3.8878$ and 11.5687, respectively. We observe that if the learning rate is too small, the learning is too slow and the algorithm does not converge (with a large $J(w)$).

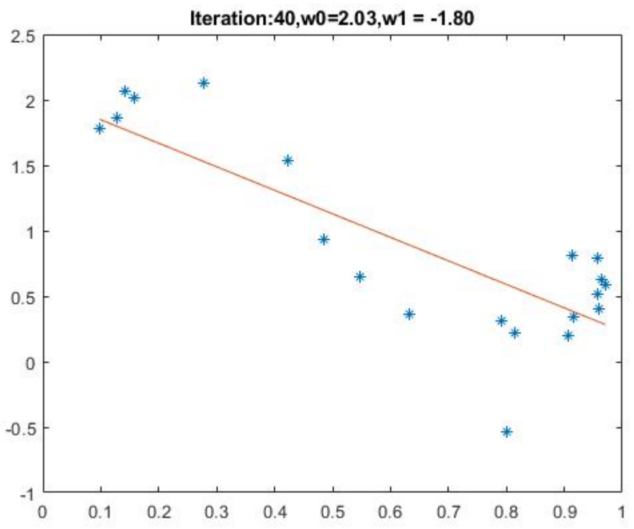
(d) **Gradient Descent Visualization** Repeat the exercise in (c) with only $\eta = 0.05$ for only 40 iterations (updates). Initialize w to be the all 0s vector and treat initialization as iteration 0. Report w, $J(w)$ and plot the fitted line (along with the data) for iteration 0, 10, 20, 30 and 40. Compare your fitted lines and $J(w)$(s) to what you get in (b). What do you observe over different iterations?
Solution: We observe that with more iterations of GD algorithm, the weight gets closer to the closed form solution. The objective function $J(w)$(s) are $26.3134, 5.7519$, $3.8237, 3.3875$ and 3.2888 for iteration 0, 10, 20, 30 and 40, respectively. The objective function decreases as the GD algorithm runs for more iterations.




Iteration:10,w0=1.36,w1 = -0.83


Iteration:20,w0=1.75,w1 = -1.40


Iteration:30,w0=1.94,w1 = -1.67



(e) **Polynomial Regression** Next let us consider the more complicated case of polynomial regression, where our hypothesis is

$$h_w(x) = w^T \phi(x) = w_0 + w_1 x + w_2 x^2 + \cdots + w_m x^m.$$

Note that the function is linear in the vector w of coefficients. We can therefore extend the result of linear regression by replacing the input matrix X with

$$\Phi = \begin{bmatrix} \phi(x_1)^T \\ \phi(x_2)^T \\ \vdots \\ \phi(x_N)^T \end{bmatrix}$$

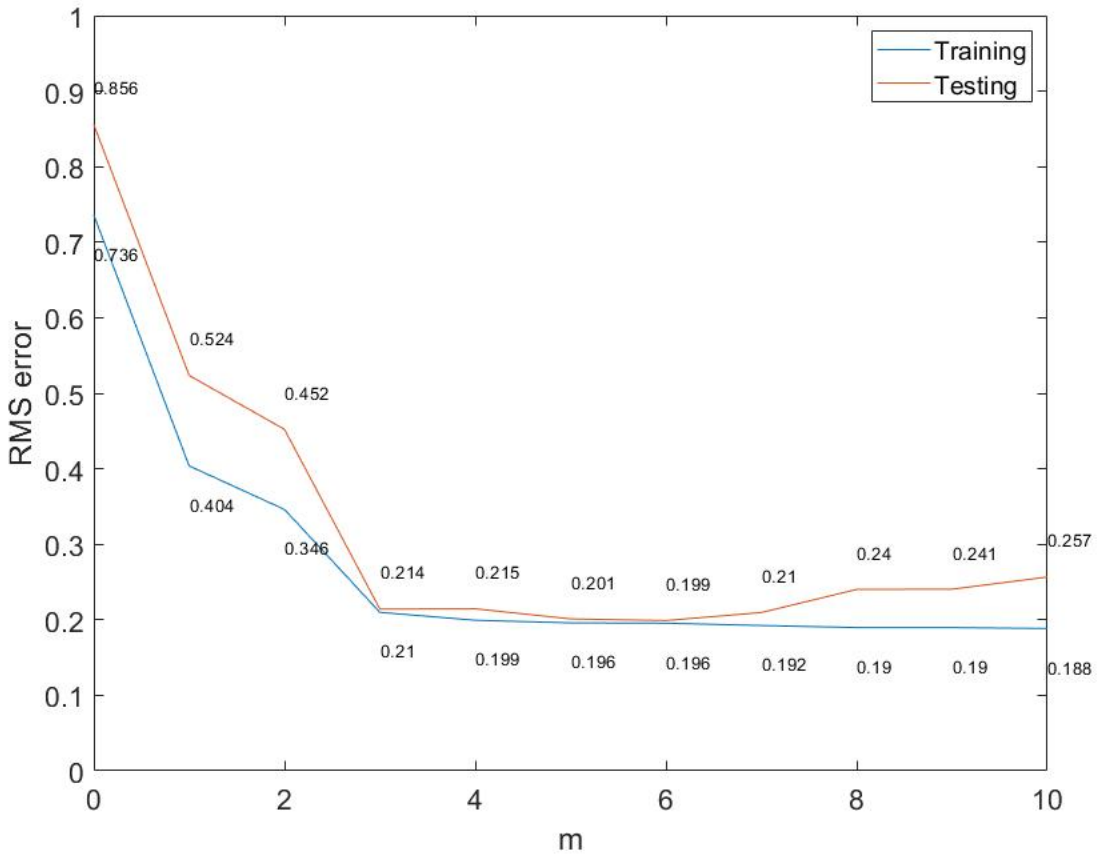
where $\phi(x)$ is a vector such that $\phi_j(x) = x^j$ for $j = 0, \cdots, m$.
For $m = 0, \cdots, 10$, use the closed-form solution to determine the best-fit polynomial regression model on the training data.

i. With this model, calculate the RMSE (Root-Mean-Square error),i.e., $E_{RMS} = \sqrt{J(w)/N}$, on both the training data and the test data.
ii. Generate a plot depicting how RMSE (both training and testing) varies with model complexity (polynomial degree m).
iii. Which degree of the polynomial would you say best fits the data? Was there evidence of under/over-fitting the training data? Use your plot to justify you answers.

Solution:

i. The RMSE values are depicted in Fig. 1.
ii. See Fig. 1 below.
iii. Polynomials of orders 3-6 can fit the data with lowest testing RMSEs. Underfitting occurs for polynomials of orders 0-2, both the training and testing RMSE are high. Over-fitting occurs for polynomials of orders 7-10, while the training RSMEs is small, the testing RMSEs is high.

Figure 1: RMSE plot


Training
Testing
RMS error
m