ECE M146 Homework 2 Solution
Introduction to Machine Learning
Instructor: Lara Dolecek
TA: Zehui (Alex) Chen, Ruiyi (John) Wu

Please upload your homework to Gradescope by April 16, 11:59 pm.
You can access Gradescope directly or using the link provided on CCLE.
You may type your homework or scan your handwritten version. Make sure all the work is discernible.

1. In class, we see 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 $y \in \mathbb{R}^N$. Prove the following statement:

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

   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.

   Hint: 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 $Az = 0$.

   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 $H = X(X^TX)^{-1}X^T$, where $X \in \mathbb{R}^{N\times M}$ and X^TX is invertible.

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

   (b) For K=2, $H^2 = X(X^TX)^{-1}X^TX(X^TX)^{-1}X^T = X(X^TX)^{-1}X^T = H$. If 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:

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

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

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

Here $\alpha_n > 0$. In class we worked out what happens for the case where all weights (the $\alpha_n' s$) are the same. In this problem, we will generalize some of those ideas to the weighted setting. Calculate the gradient by computing the partial derivative of J with respect to each of the parameters (w_0, w_1). Comment on how the $\alpha_n' s$ affect the linear regression problem. For example, what happens if $\alpha_i = 0$ for some i? Qualitatively describe what will happen in gradient descent if α_j is much greater than other $\alpha_{j'}, j' \neq j$.

Solution:

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

If $\alpha_i = 0$ for some i, the data $\{x_{i,1}, y_i\}$ is irrelevant to this problem. If α_j is much greater than other $\alpha_{j'}$, in gradient descent algorithm, a small difference between $w_0 + w_1 x_{j,1}$ 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 pass through $x_{j,1}$ and y_j.

4. In this exercise, you will develop a stochastic gradient descent (SGD) view of the perceptron algorithm.

   (a) Find the gradient $\frac{\partial J(w)}{dw}$ for the following loss function:

   $$J(w) = -\sum_{i \in \mathcal{M}} w^T x_i y_i,$$

   where $w \in \mathbb{R}^N, x_i \in \mathbb{R}^N$, $i = 1, \cdots, M$ and $y_i \in \{-1, 1\}$. The set $\mathcal{M}$ denote the index set of misclassified data points, i.e., $\mathcal{M} = \{i|\text{sign}(w^T x_i) \neq y_i\}$. You may assume $w^T x_i \neq 0, \forall i$ in this question.
   Solution:

   $$\frac{\partial J(w)}{dw} = -\sum_{i \in \mathcal{M}} x_i y_i.$$

   (b) Find the gradient $\frac{\partial J(w)}{dw}$ for the following loss function:

   $$J(w) = \sum_{i=1}^{M} \max[0, -w^T x_i y_i],$$

   where $w \in \mathbb{R}^N, x_i \in \mathbb{R}^N$, and $y_i \in \{-1, 1\}$. You may assume $w^T x_i \neq 0, \forall i$ in this question.
   Solution: Consider each term of the summation separately, we have

   $$\frac{\partial \max[0, -w^T x_i y_i]}{dw} = \begin{cases} 0, & \text{if} \quad w^T x_i y_i > 0, \\ -x_i y_i, & \text{if} \quad w^T x_i y_i < 0. \end{cases}$$

   Putting these terms together, we get

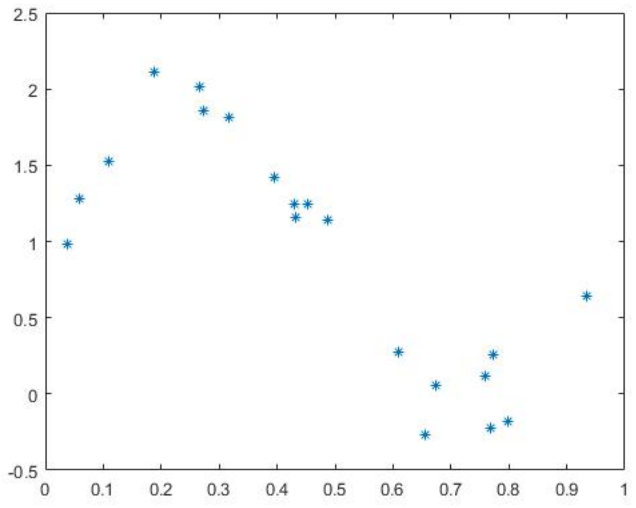
   $$\frac{\partial J(w)}{dw} = -\sum_{i \in \{i|w^T x_i y_i < 0\}} x_i y_i.$$

   (c) Compare your answers from (a) and (b), are they equivalent? Describe the update rule if you use the SGD algorithm to minimize these loss functions. With the learning rate $\eta = 1$, are they equivalent to the perceptron algorithm?
   Solution: The gradients from (a) and (b) are equivalent as $\mathcal{M} \equiv \{i|w^T x_i y_i < 0\}$. Their SGD update rule is then: $w^{k+1} = w^k + \eta x_i y_i$, only for x_i such that $\text{sign}(w^T x_i) \neq y_i$. This is exactly the update rule for the perceptron algorithm with $\eta = 1$. Meanwhile, there is a minor difference between the SGD and Perceptron in this case. The Perceptron algorithm go through the data point in order where the SGD often times randomize the order when it go through all the data points.

5. In this exercise, you will work through linear and polynomial regression. 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 you 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, visualize the training data by plotting the input v.s. 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 will work.



   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

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

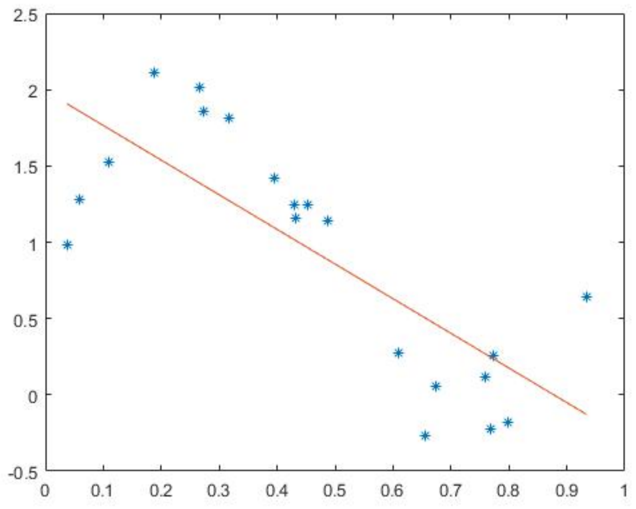
Note that to take into account the intercept term w_0, we can add an additional "feature" to each instance and set it to one, e.g., $x_{i,0} = 1$. This is equivalent to adding an additional first column to X and setting it to all ones.

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 using the training data and report w and $J(w)$. Also generate a plot depicting your training data and the fitted line.

Solution: $w^* = [1.99, -2.27]^T$ and $J(w) = 4.6036$.



(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:

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

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 using all of the following specifications for the gradient descent algorithm:

- Initialize w to be the all 0s 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 $|J(w)_{t-1} - J(w)_t| < 0.0001$.

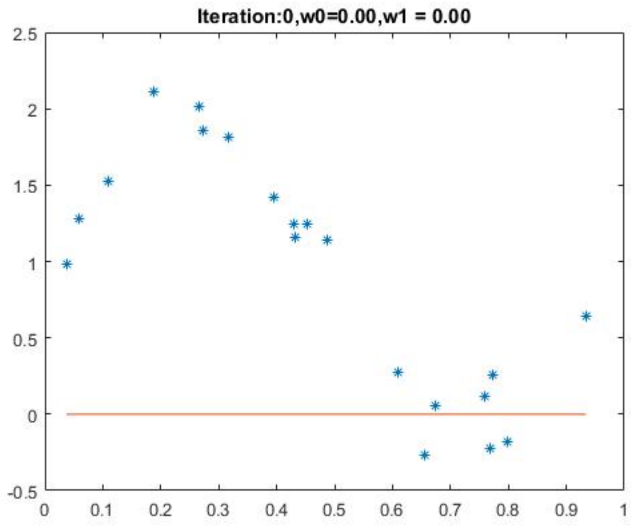
- Use a fixed learning rate η.

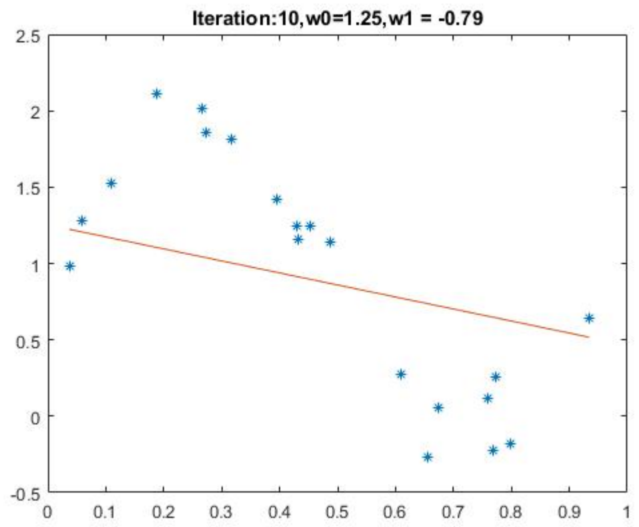
For different $\eta = 0.05, 0.001, 0.0001, 0.00001$, report the final $J(w)$ and the number of iterations until convergence (the number will be 10000 if the algorithm did not converge within 10000 iterations). Discuss how does the learning rate η affect the GD algorithm.

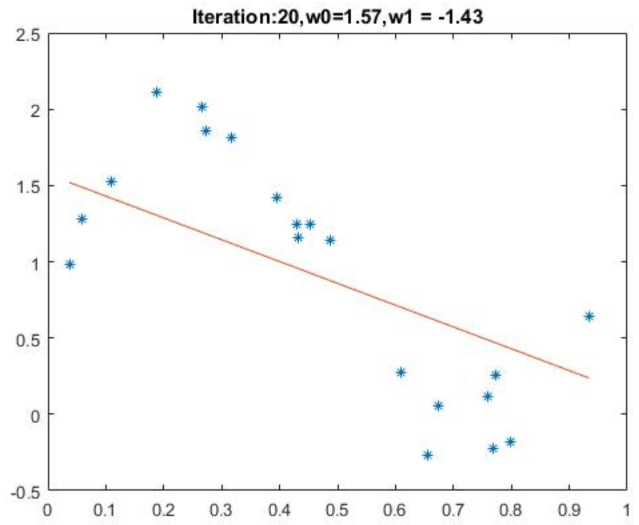
Solution: For $\eta = 0.05, 0.001, 0.0001$ and 0.00001, the # of iterations are $93, 2420$, 10000 and 10000; the $J(w)$ is $4.6044, 4.6489, 5.6408$ and 12.2494 respectively. We observe that if the learning rate is too small, the learning is too slow and the algorithm does not converge (with a large $J(w)$).

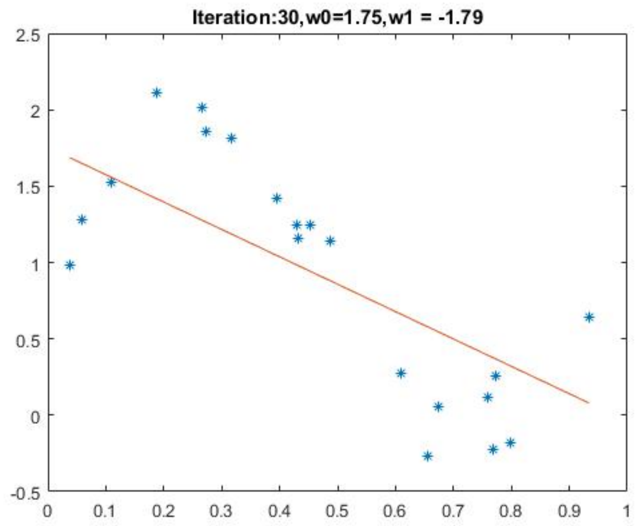
(d) **Gradient Descent Visualization** Repeat the exercise in (c) with only $\eta = 0.05$ for only 40 iterations, with w initialized to the all 0s vector. Report w, $J(w)$ and plot the fitted line (along with the data) for iteration 0, 10, 20, 30 and 40. Compare your fitted lines and $J(w)$(s) to what you get in (b). What do you observe over different iterations?

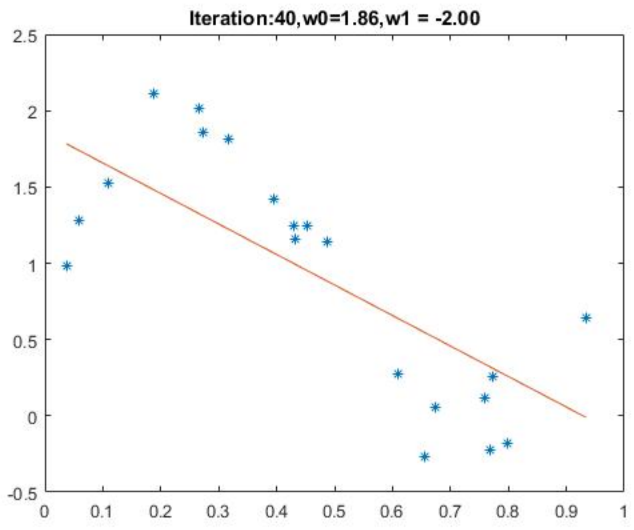
Solution: We observe that with more iterations of GD algorithm, the weight gets closer to the closed form solution. The objective function $J(w)$(s) are $28.6556, 7.6293$, $5.5775, 4.9171$ and 4.7045 for iteration 0, 10, 20, 30 and 40, respectively. The objective function decreases as the GD algorithm runs for more iterations.




Iteration:10,w0=1.25,w1 = -0.79


Iteration:20,w0=1.57,w1 = -1.43


Iteration:30,w0=1.75,w1 = -1.79



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

$$h_w(x) = w^T\phi(x) = w_0 + w_1x + w_2x^2 + \cdots + w_mx^m.$$

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

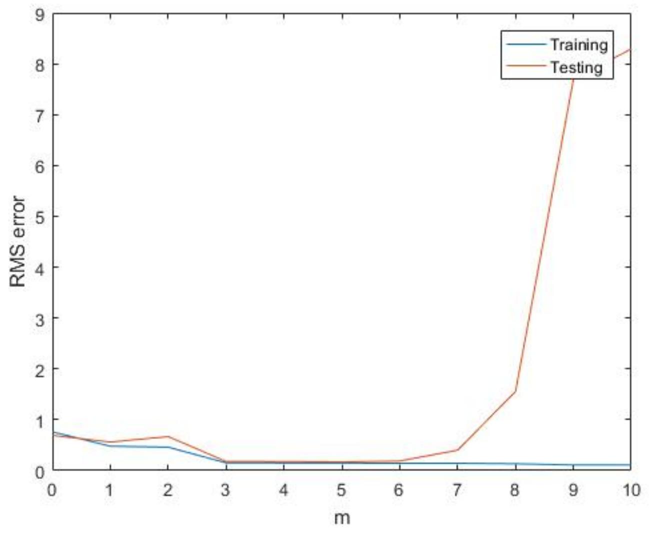
$$\Phi = \begin{bmatrix} \phi(x_1)^T \\ \phi(x_2)^T \\ \vdots \\ \phi(x_N)^T \end{bmatrix}$$

where $\phi(x)$ is a function 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. 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. Generate a plot depicting how RMSE (both training and testing) varies with model complexity (polynomial degree m). Which degree of polynomial would you say best fits the data? Was there evidence of under/over-fitting the data? Use your plot to justify you answer.

Solution:

A polynomial of order (3-6) can fit the data with lowest testing RMSE. Underfitting occurs for polynomial of order (0-2), both the training and testing RMSE are high. Over-fitting occurs for polynomial of order (7-10), while the training RSME is small, the testing RMSE is high.


RMS error
m
Training
Testing
0
1
2
3
4
5
6
7
8
9
10