-CSE 242 Homework 4-

Siman Wang swang324@ucsc.edu
Kejun Chen kchen158@ucsc.edu
November 2019

Problem 1)

Prove that for an arbitrary number of examples N and number of features D, and set of N examples with D (real-valued) features, that the Least Squares cost function $J(\theta)$ is a convex function of the D-dimensional parameter vector θ. Recall that

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{N}(h_\theta(\mathbf{x_i}) - t_\mathbf{i})^2$$

and the hypothesis $h_\theta(\mathbf{x})$ is $\theta \cdot \mathbf{x}$. You may either show that the Hessian H is positive semi-definite (easier, but requires more linear algebra knowledge), or that for any $\alpha \in [0, 1]$ and any parameter vectors v and w, we have $\alpha J(v) + (1-\alpha)J(w) \geq J(\alpha v + (1-\alpha)w)$ (a little longer, but simpler algebra). It may be helpful to prove it first for single examples and then use the property that the sum of convex functions is also a convex function. It may be helpful to know that for any real vector v, we have $v_T v = \sum_i v_i^2 \geq 0$. (Hint: to gain intuition, you might want to examine the single example case with $D = 1$ and $D = 2$ first.)

According to the convex function property, we know the sum of convex function is still convex, so we can first simplify every single item denoted $J_i(\theta), i = 1, 2, \ldots N$, for convenience, in our proof, θ and x are all column vectors with D dimension:

$$\begin{aligned}
J_i(\theta) &= \frac{1}{2}(h_\theta(x_i) - t_i)^2 \\
&= \frac{1}{2}(\theta^T x_i - t_i)^2 \\
&= \frac{1}{2}(\theta^T x_i - t_i)(\theta^T x_i - t_i) \\
&= \frac{1}{2}((\theta^T x_i)(\theta^T x_i) - \theta^T x_i t_i - t_i \theta^T x_i + t_i^2)
\end{aligned}$$

From the inner product property, we can easily know $\theta^T x_i = x_i^T \theta$. So we can rewrite the equation as:

$$\begin{aligned}
J_i(\theta) &= \frac{1}{2}(\theta^T x_i x_i^T \theta - 2t_i \theta^T x_i + t_i^2) \\
&= \frac{1}{2}\theta^T (x_i x_i^T)\theta - t_i \theta^T x_i + \frac{1}{2}t_i^2
\end{aligned}$$

The outer product matrix $A = x_i x_i^T$ is symmetric positive semi-definite. For any given non-zero vector v, we need prove:

$$\begin{aligned}
v^T A v &\geq 0 \\
v^T x_i x_i^T v &\geq 0 \\
(v^T x_i)(x_i^T v) &\geq 0 \\
(v^T x_i)(v^T x_i) &\geq 0
\end{aligned}$$

So we have known A is positive semi-definite, and $x_i x_i^T = (x_i x_i^T)^T$, so it is obvious that A is symmetric positive semi-definite.

In fact, we can see $J_i(\theta)$ is written as a general quadratic form. We only need to prove that $\theta^T A \theta$ is convex, and note that, when α is equal to 0 or 1, we get the equality. In order to prove $\theta^T A \theta$ is convex, we need prove the

following equation:

$$\alpha(v^TAv) + (1-\alpha)v^TAv \geq (\alpha w + (1-\alpha)w)^TA(\alpha v + (1-\alpha)w)$$
$$\alpha(v^TAv) + (1-\alpha)v^TAv \geq \alpha^2 v^TAv + (1-\alpha)^2 w^TAw + \alpha(1-\alpha)v^TAw + \alpha(1-\alpha)w^TAv$$
$$v^TAv + w^TAw - v^TAw - w^TAv \geq 0$$
$$(v-w)^TA(v-w) \geq 0$$

Until now, we have proven that $\theta^T(x_i x_i^T)\theta$ is convex. $-t_i\theta^T x_i + \frac{1}{2}t_i^2$ is a simple and special case of "convex"(linear), so I skip this proof. $J_i(\theta)$ is convex. $J(\theta)$ is the sum of $J_i(\theta)$, so we can say $J(\theta)$ is convex.

Here is another proof by proving Hessian matrix of $J(\theta)$ is positive semi-definite.

For a single example $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n]^T$, we have loss function $J(\theta) = \frac{1}{2}(h_\theta(\mathbf{x}) - t)^2$. $\mathbf{H}_{i,j}$ represents the item of i raw and j column in its Hessian matrix $\mathbf{H}$.

$$\mathbf{H}_{i,j} = \frac{\partial^2 J(\theta)}{\partial\theta_i\theta_j} = \frac{\partial^2 \frac{1}{2}(\theta^T\cdot\mathbf{x}-t)^2}{\partial\theta_i\theta_j} = \frac{\partial(\theta^T\mathbf{x}-t)\mathbf{x}_i}{\partial\theta_j} = \mathbf{x}_i\mathbf{x}_j.(1 \leq i \leq j \leq n)$$

$$\text{Hessian } \mathbf{H} = \begin{bmatrix} \mathbf{x}_1\mathbf{x}_1 & \mathbf{x}_1\mathbf{x}_2 & ... & \mathbf{x}_1\mathbf{x}_n \\ \mathbf{x}_2\mathbf{x}_1 & \mathbf{x}_2\mathbf{x}_2 & ... & \mathbf{x}_2\mathbf{x}_n \\ ... & ... & ... & ... \\ \mathbf{x}_n\mathbf{x}_1 & \mathbf{x}_n\mathbf{x}_2 & ... & \mathbf{x}_n\mathbf{x}_n \end{bmatrix} = \mathbf{x}^T\mathbf{x}$$

$\forall \mathbf{A} \in R^{n\times n}, \mathbf{A}^T\mathbf{H}\mathbf{A} = \mathbf{A}^T\mathbf{x}^T\mathbf{x}\mathbf{A} = (\mathbf{x}\mathbf{A})^T\mathbf{x}\mathbf{A} \geq 0.$

Thus, hessian H is positive semi-definite. Hence, least square is a convex function for a single example. It is safe to say that $J(\theta)$ is a convex function for each example by the same proof. Since the sum of convex functions is also a convex function, we have $J(\theta) = \frac{1}{2}\sum_{i=1}^{N}(h_\theta(\mathbf{x_i}) - t_i)^2$ is a convex function.

Problem 2)

The following table describes the training data for this problem:

x_1	x_2	x_3	t
3	9	2	19
6	9	1	19
7	7	7	10
8	6	4	11
1	0	8	-3

Note: This is artificial and slightly cherry picked data. For each instance, the features x_i were generated randomly and then noise was added. The targets were generated by the following function before the noise: $t = 0 * x_{1(clean)} + 2 * x_{2(clean)} - 1 * x_{3(clean)} + 3$. Your computed weight vector should be somewhat similar to this,but with so few examples and so much relative noise you shouldn't expect to get really close.

1. Save the data and run a linear regression algorithm on the full training set without a bias component.(Scikit learn has a **fit intercept** parameter that can be set to false to do this.) Report the model (weights) and "Root mean squared error". Note that testing on the training set means we will may have relatively small error. Now add an $x0$ feature that is always 1 to each example, and re-run the regression. Did the model or accuracy change? We will use this augmented 4-feature training set for the rest of the problem.

   Without bias, the weighted parameter is [-0.18715635; 2.24806365; -0.4395789], and the root mean square error is 0.7818754163281899.
   When we add another feature $x_0 = 1$, we re-run the code, and find the weighted parameter is

[4.36080292; −0.13429383; 1.84768377; −0.89658481], and the root mean square error is 0.18970922173315738. The accuracy changes a lot.

2. Suppose you had an unlabeled instance $x = [3, 3, 5]$. What t prediction for t would the second model from part (a) give?

   In the second model, we get the argument $x = [1, 3, 3, 5]$ first. Then we use the
   $w = [4.36080292; -0.13429383; 1.84768377; -0.89658481]$, and then we get the prediction $y = 5.01804869$

3. The "ridge parameter" is another name for the $L2$ regularization coefficient (often written λ). Run the linear regression (ridge regression) with small and large regularization parameters, say 0.0001 and 0.3. What are learned weights? Is the change qualitatively what you would expect?

   Note that we use bias component for this part. When the regularization coefficient is 0.0001, we get can learned weighs is
   $[4.3579732, -0.13432373, 1.8479395, -0.89628787]$

   Note that we use bias component for this part. When the regularization coefficient is 0.3, we get can learned weighs is
   $[1.54871459, -0.15170183, 2.09151721, -0.60239432]$

   In $L2$ regularization, increasing the *lambda* value strengthens the regularization effect, which has a trend to make the model simple and avoid over-fitting. But if the regularization coefficient is too big, the regularization effect will play the main role in learning process, and it has the trend to encourage weight value to 0, not exactly 0, especially for those non-informative features. In the example, feature x_0 doesn't contain much information, because this feature is always equal to 0 for every training data point. We cans see clearly that when λ is 0.3, the first weight is much lower than the non-regularization function. As for $\lambda = 0.0001$, we cannot see much difference with the non-regularization function, because it is close to 0. Beside, when I set $\lambda = 100, 200$, I can see clearly the first weight becomes much lower and lower, but still not 0.

4. Stochastic Gradient Descent. Start with the weight vector $w = [1, 1, 1, 1]$ and step through stochastic gradient descent on the first two instances: $x = [1, 3, 9, 2]$, $t = 19$ and $x = [1, 6, 9, 1]$, $t = 19$ (note that these include the bias component). For your learning rate parameter, use $\eta = 0.01$, For your error function use squared error. Report the updated weights after each instance. Show your work for at least the first instance.

   First, we can define the square loss function is $J(\theta) = \frac{1}{2}\sum_i^n (h_\theta(x_i) - t_i)^2$ θ is the weights, in this problem description, it is totally same with w. According to SGD method, if we want to minimize the loss function, we can calculate its gradient first, and update the weights. Sometimes it is difficult to directly calculate the weights when the gradient is equal to 0, but we can update the weights step by step to approach the optimal solution. When the gradient is negative (positive), we need to increase (decrease) the weights. Note that here x_i represent different examples instead of features, sorry for using the same denotations, so I hope it is understandable.

$$\begin{aligned}
\frac{\partial J}{\partial \theta} &= \frac{\partial(\frac{1}{2}\sum_i^n (h_\theta(x_i) - t_i)^2)}{\partial \theta} \\
&= (h_\theta(x_i) - t_i)) \frac{\partial(\sum_i^n (\theta x_i - t_i))}{\partial \theta} \\
&= (h_\theta(x_i) - t_i)) x_i \\
&= (\theta x_i - t_i) x_i
\end{aligned}$$

   Then, we define the learning rate is η, then the updated weights should be written as:

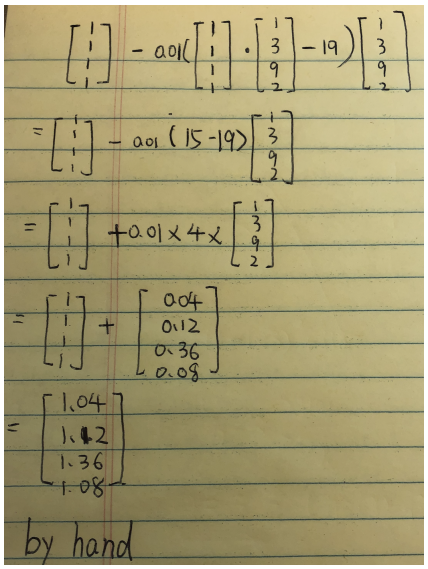
$$\theta_j = \theta_{j-1} - \eta(\theta_{j-1} x_i - t_i) x_i$$

   Note that θx is the inner product, so we need get a specific number of this part, so sometimes we need to transpose one of them to make sure we can get a number, according to the θ or x are row vector or

column vector. For convenience, we replace θ in the previous proof with w. The first time we use SGD starting with the initial weights $w_0 = [1, 1, 1, 1]$, and the first example denoted x_1 is $x_1 = [1, 3, 9, 2]$ and label is $t_1 = 19$, $\eta = 0.01$. Replace the learning method we have got with this real number, we can easily get $w_1 = [1.04, 1.12, 1.36, 1.08]$.

After the first example learning, we have got the updated weight w_1, and we can start with w_1 and continue to study weights $w_2 = w_1 - \eta(\theta x_2 - t_i)x_2$, now $x_2 = [1, 6, 9, 1]$, label $t_2 = 19$, then we can get updated weight $w = [1.0192, 0.9952, 1.1728, 1.0592]$

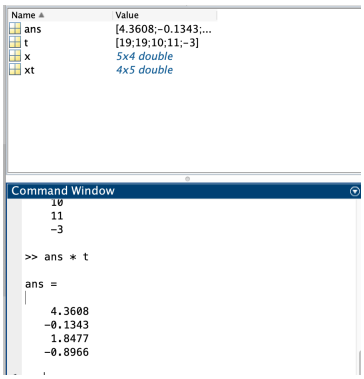
I will show how can I got it by hand:



5. Compute regression weights using eq.3.34 from Bishop. In our notation that is: $w = (X^T X)^{-1} X^T t$, here X is the (5-row, 4-column) matrix whose rows are the unlabeled instances and whose columns are each of the features for the instance augmented by the fixed feature $x0 = 1$ which will lead to the bias. How does this w compare to the weights from part a? If you like, you may use your favorite math software for the matrix arithmetic but it is quite feasible to do directly, except for the inverse. You should not compute the inverse by hand (Google "inverse matrix 4x4 calculator").

   I use Matlab to calculate w, I will show the results step by step:

   First, I type matrix X which is 5x4, and then I use transpose function in Matlab to get X^T, which is 4x5 columns. Then $(X^T X)$ is 4x4 columns, and we use inv function in Matlab to get $(X^T X)^{-1}$. Later,$(X^T X)^{-1}$ multiply X^T, we get a 4x5 columns matrix. Last, label t is a 5x1 column, so we can get a 4x1 column matrix, which is exactly our weighted vector $w = [4.3608; -0.1343; 1.8477; -0.8966]$.It seems we get the same weighted parameters as part (a), under the argument examples.



6. If the examples are re-ordered (so t are permuted the same way), what happens to the learned w vector and why?

First, we need to answer what is permutation matrix. In a permutation matrix, one entry of 1 in each row and each column and 0s elsewhere. The following is a 5-dimensional matrix denoted A:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

We can see what will happen if we use AX, note X is our training data matrix, and each row represents a data point, and each column is a feature. In this problem augmented X is:

$$\begin{bmatrix} 1 & 3 & 9 & 2 & 19 \\ 1 & 6 & 9 & 1 & 19 \\ 1 & 7 & 7 & 7 & 10 \\ 1 & 8 & 6 & 4 & 11 \\ 1 & 1 & 0 & 8 & -3 \end{bmatrix}$$

Then we can get a AX is:

$$\begin{bmatrix} 1 & 3 & 9 & 2 & 19 \\ 1 & 8 & 6 & 4 & 11 \\ 1 & 6 & 9 & 1 & 19 \\ 1 & 1 & 0 & 8 & -3 \\ 1 & 7 & 7 & 7 & 10 \end{bmatrix}$$

It is very clear that AX is reordered examples.

From Wikipedia, we can know given permutation matrix, denoted A_i, A_iX can represent an arbitrary reordered training examples that of X, and A_i is $N * N$, N is the equal to the number of our training data, in this problem, $N = 5$. In addition, a permutation matrix A_i satisfies $A_i^T A_i = I$.

Then, we will show the strict proof:

$$\begin{aligned} w &= ((A_iX)^T(A_iX))^{-1}(A_iX)^T(A_it) \\ &= (X^TA_i^TA_iX)^{-1}X^TA_i^TA_it \\ &= (X^TIX)^{-1}X^TIt \\ &= (X^TX)^{-1}X^Tt \end{aligned}$$

Thus, we get the same learned w vector regardless of order of examples.

Problem 3)

For the purpose to comparing the results, in this problem we use the sum-square-loss. In our notation, it will be Let N be the number of data points of the data set,and ϵ^2 be the variance of the noise.

1. Closed form solution for Linear Regression Generate two different data sets with $N = 10000$ examples and $\sigma^2 = 0.01$. Select the first k examples from the first data set as the training data set and use the second data set as the test data set. Use the closed form solution to minimize the MSE on the training data. Report the training error and test error for $k = 10, 100, 1000, 10000$. And explain the results you get.

   First, we know the closed form of Linear Regression is $w = (X^TX)^{-1}X^Tt$. Given k we choose different training size, and get the corresponding w, and use the estimated w on the test set with $w \cdot x$, and then we can predict the output labels of the whole N test set. According to our true label, we can get the MSE error based on k training model. It is obvious that with the increase of k, our error should be lower. Besides,the error in the training set should be lower than the test set. Our results act as this and I keep the 7 digital after point:

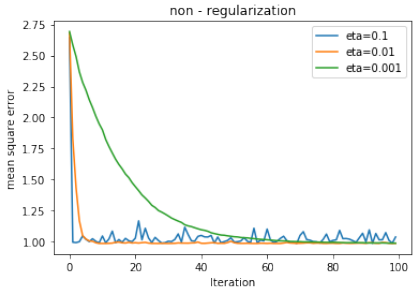
	train error	test error
k = 10	0.0162657	0.0160602
k = 100	0.0105210	0.0103739
k = 1000	0.0100487	0.0099546
k = 10000	0.0100249	0.0099462

2. Stochastic Gradient Descent Generate a training data set with $N = 10000, \sigma^2 = 1$. And use the Stochastic Gradient Descent algorithm: Run the algorithm with $T = 10000$ and $\eta = 0.1, 0.01, 0.001$. Plot the MSE curves (say with a data point every 100 updates) for these three cases. Why would the different η's lead to different curves?

   The learning process has been shown in problem 2 and I will use its conclusion directly.

$$\theta_j = \theta_{j-1} - \eta(\theta_{j-1}x_i - t_i)x_i$$

   Starting with the initial weight, every time we add a new training point, we can update the weight one time. Using the updated weight, we can calculate the error. The new training point is randomly chosen and the iteration 10000, which means we update the weights 10000 times. With different learning rate, we can draw the following picture. Note the axis represents iteration times. Although I have recorded all the error about each update, but I choose equal interval 100 to draw the picture. Seeing from the picture, the learning rate is lower, the curse is more smooth.



   Explanation:

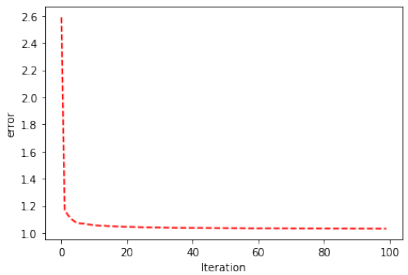
   When $\eta = 0.1$, the curve rapidly declines to a low level but sharply fluctuates during following training steps. With a large learning rate, updated difference $|\theta_t - \theta_{t-1}|$ is large; hence, model can quickly approach to the optimal solution but probably skip over the optimal solution. Thus, learned w varies around the optimal solution.

   When $\eta = 0.001$, the curve is smooth, slowly declines, and asymptotically approaches to the optimal solution. With a small learning rate, updated difference $|\theta_t - \theta_{t-1}|$ is small; hence, model can gradually approach to the optimal solution with a slow pace. In each training step, current learned w is probably fitter than previous learned w.

   When $\eta = 0.01$, it shows a trade-off between learning speed and learning performance.

3. Repeat 3b, but anneal the learning rate η to be something like $1/i$ in each iteration i.

   When the learning rate satisfies the previous requirements, we can get the curve. I calculate the error of each update, but I only choose 100 data points to draw the picture:



4. Include regularization in problem 3

First we use regularization form of Closed form solution for Linear Regression. Considering the regularization the estimated weight should be $w = (\lambda I + X^T X)^{-1} X^T t$, which is quite similar to the previous method, we can easily get the error with different λ. When the $\lambda = 0.5$, we can get:

	train error	test error
k = 10	0.1320137	0.1313284
k = 100	0.0118789	0.0117515
k = 1000	0.0101874	0.0101008
k = 10000	0.0101644	0.0100685

When the $\lambda = 0.01$, we can get:

	train error	test error
k = 10	0.0753149	0.0746523
k = 100	0.0116370	0.0115900
k = 1000	0.0101137	0.0101256
k = 10000	0.0099921	0.0100243

When we add regularization in our SGD method, we can get the updated weight should satisfy:

$$\theta_j = \theta_{j-1} - \eta(\theta_{j-1} x_i - t_i) x_i - \lambda \theta_{j-1}$$

We can use the similar method to draw the MSE figure, using different λ: