CSCI-4/5587, Fall-2023, Machine Learning I
Study Guide for Test#2

(Based on Chapter 2)

(Please don't distribute this study guide.
The guide is provided for your study purpose only)

1. What is supervised learning?
Ans: [do it by yourself]

2. What is an overfitting problem?
Ans: [do it by yourself]

3. What are the differences between 'regression' and 'classification' problems?
Ans: [Chapter 2, do it by yourself]

4. Describe Newton's method for deriving the equation for finding the minimum (or maximum) of a given equation.

Ans: Say, we have an equation $f(x) = 0$

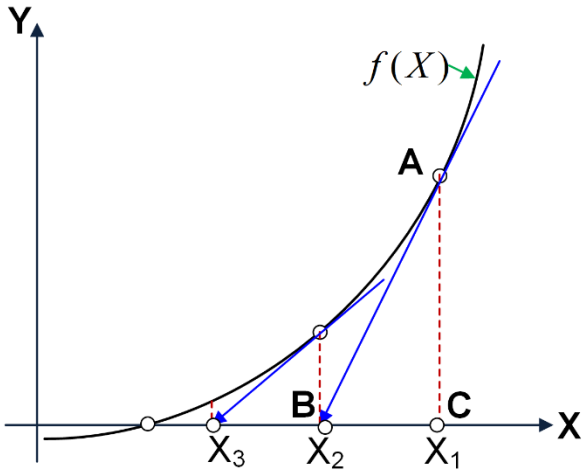


Figure: How Newton's method works in finding the solution.

For the solution of the equation (i.e., to find for what value of x, $f(x) = 0$), assume our initial point is x_1 which intersect the x-axis at C and $f(x)$ at A (see Figure above). Also, consider that the tangent at A intersects the x-axis at B, where the value of x is x_2. From ΔABC and the definition of the slope of an equation, we can write:

$$f'(x_1) = \frac{f(x_1) - 0}{x_1 - x_2} \qquad (i)$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} \qquad (ii)$$

In general, we can write,

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)} \qquad (iii)$$

Equation (*iii*) can be iteratively used to get the solution of the equation.

Now, for a minimization problem, if minimum exists then, we need to find the value of x for which $f'(x) = 0$.

We can think of going for the solution of the equation, $f'(x) = 0$ using (*iii*).

Therefore, we can similarly write,

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)} \qquad (iv)$$

Now, we can use equation (*iv*) to find the minimum (or, maximum) of equation $f(x) = 0$.

5. (a) What is the gradient descent equation? What is the α in that equation used for? (b) How might the overshooting problem occur with a gradient descent approach? (c) How does the gradient ascent equation differ from descent equation?

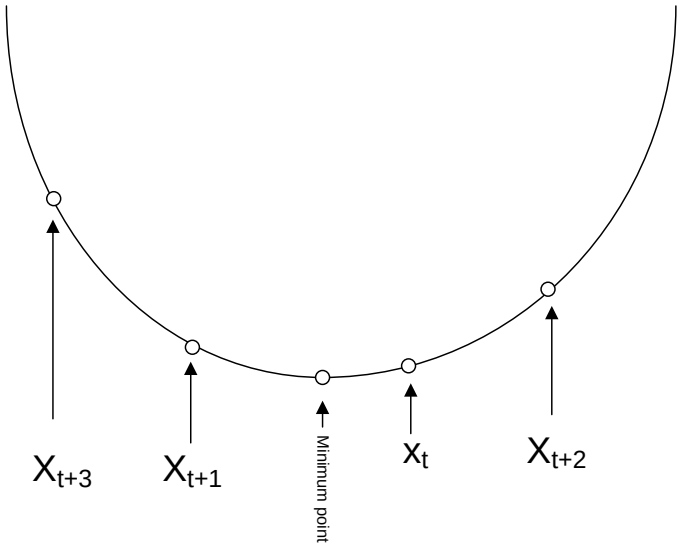
Ans: (a) $x_{t+1} = x_t - \alpha \ \nabla f(x_t)$,

where x is the input variable, t is the time, and f is the function.

Ans: (b) Hints:

If we set the value of α (alpha, the learning rate), to a higher value then overshooting might occur. How?

(A) <---------- (B)



Say we are after the minimum point.
Assume, we are at X(t) now.

We set α a very high value and compute, x(t+1) as:

$$x_{t+1} = x_t - \alpha \ \nabla f(x_t)$$

Note that the α is constant – has a fixed value or it does not change in the iteration but remember slope $\nabla f(x_i)$ dependent on the location of x_i and it changes.

Now, x(t+1) surpasses the minimum point when moving from **(B) to (A) direction** for the minimum point for a very higher deduction (amount: $\alpha \ \nabla f(x_t)$) from x(t).

From the figure, we see a distance of x(t+1) from the minimum point is higher than the distance of x(t) from the minimum point. So, it is obvious that:

$$|\nabla f(x_{t+1})| > |\nabla f(x_t)|$$

[in this case, $\nabla f(x_t)$ is +ve and $\nabla f(x_{t+1})$ is -ve]

$\alpha |\nabla f(x_{t+1})| > \alpha |\nabla f(x_t)|$ will be true.

Based on this information and the figure we can say that the next point x(t+2) would be behind x(t) as we will apply:

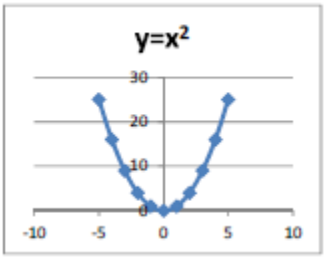
$$x_{t+2} = x_{t+1} - \alpha \ \nabla f(x_{t+1})$$

Again $\alpha |\nabla f(x_{t+2})| > \alpha |\nabla f(x_{t+1})| > \alpha |\nabla f(x_t)|$ will be true, so the next point x(t+3) will be behind x(t+1), etc.

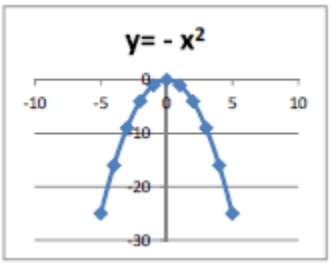
So, we see instead of converging, it is diverging in each iteration => overshooting.

Ans: (c) Hints:
Gradient descent (climbing down) is given by: $x_{t+1} = x_t - \alpha \ \nabla f(x_t)$
Gradient ascent (climbing up) is given by: $x_{t+1} = x_t + \alpha \ \nabla f(x_t)$

(A) <---------- (B) **(A) <---------- (B): direction**



(a)



(b)

Figure: Curve has (a) minimum point and (b) maximum point.

For curve (a) when the iteration goes from **(B) to (A)** to get the minimum point the slope (i.e., ∇f or, $f'(x)$) remains +ve. And we set the α (alpha; the learning rate) as a positive fixed number. So to reduce the x_t in each iteration to move from **(B) to (A)** the term '$\alpha \ \nabla f(x_t)$' must be deducted.

Now, for the curve (b), we cannot find a minimum (the minimum point is not there), rather we have a maximum point. Using the equation, iteratively, if we are moving from **(B) to (A)**, the slope in the case will be –ve, and α is positive, so the term $\alpha \ \nabla f(x_t)$ is now negative. So, we need to have a +ve sign in front of the term to keep it negative so, that the x_t is deducted in each iteration to move it from **(B) to (A) direction**.

6. We can write down the equation for the gradient descent method as:

$$\boldsymbol{\beta_j(t+1) = \beta_j(t) - \frac{2\alpha}{N}\sum_{i=1}^{N}\{x^T(i)\,\beta - y(i)\}.x(i)_j}$$

where j ={0, 1, 2, … , p} and i ={1, 2, … , N}. Convert the equation in terms of vector and matrix to avoid the two loops for indices i and j. Explain your conversion.

Ans: Hints: [The answer is briefly described here, but you will need to explain it, as it was taught in class – to recall, watch the corresponding video.]

Following the given Equation in the question, in terms of MSE(β) we can write the partial derivatives of the cost function:

$$\frac{\partial}{\partial \beta_j} MSE(\beta) = \frac{2}{N}\sum_{i=1}^{N}\{x^T(i)\beta - y(i)\}.x(i)_j \ \ \ldots\ldots\ldots\ldots\ldots \ (1)$$

Instead of computing these partial derivatives individually, we can use the following Equation to calculate them all in one go. The gradient vector noted $\nabla_\beta MSE(\beta)$, contains all the partial derivatives of the cost function (one for each model parameter).

$$\nabla_{\boldsymbol{\beta}} MSE(\boldsymbol{\beta}) = \begin{bmatrix} \frac{\partial}{\partial \beta_0} MSE(\beta) \\ \frac{\partial}{\partial \beta_1} MSE(\beta) \\ \vdots \\ \frac{\partial}{\partial \beta_p} MSE(\beta) \end{bmatrix} = \frac{2}{N} \mathbf{X}^{\mathrm{T}} (\mathbf{X}\boldsymbol{\beta} - \mathbf{Y}) \ \ldots\ldots\ldots\ldots\ldots (2)$$

Thus, we can define the Gradient Descent step as following (including the learning rate $\boldsymbol{\alpha}$):

$$\boldsymbol{\beta}(t+1) = \boldsymbol{\beta}(t) - \alpha \nabla_{\boldsymbol{\beta}} MSE(\boldsymbol{\beta}) \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3)$$

Therefore, finally, we can write:

$$\boldsymbol{\beta}(t+1) = \boldsymbol{\beta}(t) - \frac{2\alpha}{N} \mathbf{X}^{\mathrm{T}} (\mathbf{X}\boldsymbol{\beta} - \mathbf{Y}) \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4)$$

7. We can write the equation, RSS Equation: $RSS(\beta) = \sum_{i=1}^{N} (y_i - x_i^T \beta)^2$ **, in vector form as:** $\mathrm{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta)$ **, where X is an** $N \times p$ **matrix with each row an input vector, and y is an** N**-vector of the outputs in the training set. Differentiating w.r.t. β, we get the *normal equations*:** $\mathbf{X}^T (\mathbf{y} - \mathbf{X}\beta) = 0$ **.**

Show the steps for getting $\mathbf{X}^T (\mathbf{y} - \mathbf{X}\beta) = 0$ **from** $\mathrm{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta)$ **.**

Ans:

$$\begin{aligned} RSS(\beta) &= (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta) \\ &= [\mathbf{y}^T - (\mathbf{X}\beta)^T](\mathbf{y} - \mathbf{X}\beta) \qquad [\because (A \pm B)^T = A^T \pm B^T,\ (AB)^T = B^T A^T] \\ &= (\mathbf{y}^T - \beta^T \mathbf{X}^T)(\mathbf{y} - \mathbf{X}\beta) \\ &= \mathbf{y}^T\mathbf{y} - \beta^T \mathbf{X}^T \mathbf{y} - \mathbf{y}^T \mathbf{X}\beta + \beta^T \mathbf{X}^T \mathbf{X}\beta \qquad \begin{bmatrix} \because a^T b = b^T a \\ \therefore \mathbf{y}^T \mathbf{X}\beta = (\mathbf{X}\beta)^T \mathbf{y} = \beta^T \mathbf{X}^T \mathbf{y} \end{bmatrix} \\ &= \mathbf{y}^T\mathbf{y} - \beta^T \mathbf{X}^T \mathbf{y} - \beta^T \mathbf{X}^T \mathbf{y} + \beta^T \mathbf{X}^T \mathbf{X}\beta \\ &= \mathbf{y}^T\mathbf{y} - 2\beta^T \mathbf{X}^T \mathbf{y} + \beta^T \mathbf{X}^T \mathbf{X}\beta \end{aligned}$$

Therefore, we have

$$RSS(\beta) = \mathbf{y}^T\mathbf{y} - 2\beta^T \mathbf{X}^T \mathbf{y} + \beta^T \mathbf{X}^T \mathbf{X}\beta$$

Now, differentiating w.r.t. β and equating it to zero, we get:

$$\frac{\partial}{\partial \beta}[\mathbf{y}^T\mathbf{y} - 2\beta^T \mathbf{X}^T \mathbf{y} + \beta^T \mathbf{X}^T \mathbf{X}\beta] = 0$$

$$\Rightarrow \quad \frac{\partial}{\partial \beta}(\mathbf{y}^T\mathbf{y}) - 2\frac{\partial}{\partial \beta}(\beta^T)\mathbf{X}^T\mathbf{y} + \frac{\partial}{\partial \beta}(\beta^T \mathbf{X}^T \mathbf{X}\beta) = 0$$

$$\Rightarrow\ 0 - 2\mathbf{X}^T\mathbf{y} + \frac{\partial}{\partial\beta}(\beta^T)\mathbf{X}^T\mathbf{X}\beta + \beta^T\mathbf{X}^T\mathbf{X}\frac{\partial}{\partial\beta}(\beta) = 0$$

$$\Rightarrow\ 0 - 2\mathbf{X}^T\mathbf{y} + \mathbf{X}^T\mathbf{X}\beta + \beta^T\mathbf{X}^T\mathbf{X} = 0 \qquad \begin{bmatrix} \because \beta^T\mathbf{X}^T\mathbf{X} = \beta^T(\mathbf{X}^T\mathbf{X}) \\ = (\mathbf{X}^T\mathbf{X})^T\beta = (\mathbf{X}^T)(\mathbf{X}^T)^T\beta \\ = \mathbf{X}^T\mathbf{X}\beta \end{bmatrix}$$

$$\Rightarrow\ -2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\beta = 0$$

$$\Rightarrow\ -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0$$

$$\therefore \mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0$$

8. Describe the computational complexity of the Normal Equation.

Ans: The Normal Equation computes the inverse of $\mathbf{X}^T\mathbf{X}$, which is a $(p + 1) \times (p + 1)$ matrix (where p is the number of features). The computational complexity of inverting such a matrix is typically about $O(p^{2.4})$ to $O(p^3)$, depending on the implementation. In other words, if you double the number of features, you multiply the computation time by roughly $2^{2.4} = 5.3$ to $2^3 = 8$.

The SVD approach used by Scikit-Learn's **LinearRegression** class is about $O(p^2)$. If you double the number of features, you multiply the computation time by roughly 4.

Both the Normal Equation and the SVD approach get very slow when the number of features grows large (e.g., 100,000). On the positive side, both are linear with regard to the number of instances in the training set (they are $O(N)$, where N is the number of instances or samples), so they handle large training sets efficiently, provided they can fit in memory.

Also, once you have trained your Linear Regression model (using the Normal Equation or any other algorithm), predictions are very fast: the computational complexity is linear with regard to both the number of instances you want to make predictions on and the number of features. In other words, making predictions on twice as many instances (or twice as many features) will take roughly twice as much time.

9. (a) How does the performance of the Gradient Descent vary depending on the learning rate (α)? (b) What are the names of the three variants of the Gradient Descent algorithm?

Ans:
a. The performance of GD dependents on learning rate:

- If the learning rate is too low: the algorithm will eventually reach the solution, but it will take a long time.
- If the learning fits well, then in just a few iterations, GD converges to the solution.
- If the learning rate is too high: the algorithm diverges, jumping all over the place and getting further and further away from the solution at every step.

To find a good learning rate, you can use a grid search. However, you may want to limit the number of iterations so that grid search can eliminate models that take too long to converge. To do that, we can apply a coarse-grain search first and then use the fine-grain search on the potential subpart of the grid found in the coarse-grain search.

b. The three variants are:

- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini-batch Gradient Descent

-- x --