CS 5350/6350: Machine Learning Fall 2023

Homework 2

Milena Belianovich

- You are welcome to talk to other members of the class about the homework. I am more concerned that you understand the underlying concepts. However, you should write down your own solution. Please keep the class collaboration policy in mind.
- Feel free to discuss the homework with the instructor or the TAs.
- Your written solutions should be brief and clear. You do not need to include original problem descriptions in your solutions. You need to show your work, not just the final answer, but you do *not* need to write it in gory detail. Your assignment should be **no more than 20 pages**. Every extra page will cost a point.
- Handwritten solutions will not be accepted.
- *Your code should run on the CADE machines.* You should include a shell script, `run.sh`, that will execute your code in the CADE environment. Your code should produce similar output to what you include in your report.

  You are responsible for ensuring that the grader can execute the code using only the included script. If you are using an esoteric programming language, you should make sure that its runtime is available on CADE.
- Please do not hand in binary files! We will *not* grade binary submissions.
- The homework is due by **midnight of the due date**. Please submit the homework on Canvas.

1 Paper Problems [40 points + 8 bonus]

1. [5 points] We have derived the PAC guarantee for consistent learners (namely, the learners can produce a hypothesis that can 100% accurately classify the training data). The PAC guarantee is described as follows. Let H be the hypothesis space used by our algorithm. Let C be the concept class we want to apply our learning algorithm to search for a target function in C. We have shown that, with probability at least $1-\delta$, a hypothesis $h \in H$ that is consistent with a training set of m examples will have the generalization error $\text{err}_D(h) < \epsilon$ if

$$m > \frac{1}{\epsilon}\big(\log(|H|) + \log\frac{1}{\delta}\big).$$

   (a) [2 points] Suppose we have two learning algorithms L_1 and L_2, which use hypothesis spaces H_1 and H_2 respectively. We know that H_1 is larger than H_2, i.e., $|H_1| > |H_2|$. For each target function in C, we assume both algorithms can find a hypothesis consistent with the training data.

i. [1 point] According to Occam's Razor principle, which learning algorithm's result hypothesis do you prefer? Why?

ii. [1 point] How is this principle reflected in our PAC guarantee? Please use the above inequality to explain why we will prefer the corresponding result hypothesis.

(b) [3 points] Let us investigate algorithm L_1. Suppose we have n input features, and the size of the hypothesis space used by L_1 is 3^n. Given $n = 10$ features, if we want to guarantee a 95% chance of learning a hypothesis of at least 90% generalization accuracy, how many training examples at least do we need for L_1?

Solution.

a)

i) According to Occam's Razor principle, simpler hypotheses are preferred over more complex ones given that they both explain the data sufficiently well. So if both L_1 and L_2 can find hypotheses consistent with the training data, we would prefer the result hypothesis from the learning algorithm with a smaller hypothesis space. Thus, we'd prefer the result hypothesis from L_2 since H_2 is smaller than H_1.

ii) The PAC guarantee equation reflects the principle of Occam's Razor. If we look at the term $\log(|H|)$, it becomes evident that the size of the required training set m increases logarithmically with the size of the hypothesis space $|H|$. So, if $|H_1| > |H_2|$, then the required m to meet the same ϵ and δ for L_1 will be greater than that for L_2. This implies that for algorithms with larger hypothesis spaces, more data is required to guarantee a certain level of performance, which is a direct consequence of preferring simpler hypotheses.

b)

Given the information:
$n = 10$,
$|H_1| = 3^{10}$,
$\delta = 0.05$ (because there's a 95% chance of learning),
$\epsilon = 0.10$ (because we want at least 90% generalization accuracy).
Plugging these into our PAC guarantee formula:

$$m > \frac{1}{\epsilon}\big(\log(|H|) + \log\frac{1}{\delta}\big).$$

$$m > (1/0.10) * (log(3^{10}) + log(1/0.05))$$

$$m > 10 * (10 * log(3) + log(20))$$

Using logarithm properties:

$$m > 10 * (10 * 1.0986 + 2.9957)$$

$$m > 10 * (10.986 + 2.9957)$$

$$m > 10 * 13.9817$$

$$m > 139.817$$

Rounding up, we'll need at least 140 training examples for L_1 to guarantee a 95% chance of learning a hypothesis with at least 90% generalization accuracy.

2. [5 points] In our lecture about AdaBoost algorithm, we introduced the definition of weighted error in each round t,

$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\big(\sum_{i=1}^{m} D_t(i) y_i h_t(x_i)\big)$$

where $D_t(i)$ is the weight of i-th training example, and $h_t(x_i)$ is the prediction of the weak classifier learned round t. Note that both y_i and $h_t(x_i)$ belong to $\{1, -1\}$. Prove that equivalently,

$$\epsilon_t = \sum_{y_i \neq h_t(x_i)} D_t(i).$$

Solution. Let's break this equation down by considering the two possible cases for each training example:
1) $y_i = h_t(x_i)$
Given y_i and $h_t(x_i)$ belong to $\{1, -1\}$, the products for the two possible values are:
If $y_i = 1$ and $h_t(x_i) = 1$, then $y_i h_t(x_i) = 1$
If $y_i = -1$ and $h_t(x_i) = -1$, then $y_i h_t(x_i) = 1$
2) $y_i \neq h_t(x_i)$
For these misclassified examples:
If $y_i = 1$ and $h_t(x_i) = -1$, then $y_i h_t(x_i) = -1$
If $y_i = -1$ and $h_t(x_i) = 1$, then $y_i h_t(x_i) = -1$
Using the above cases in the given expression:
For correctly classified examples, the term $D_t(i) y_i h_t(x_i)$ is $D_t(i)$
For correctly misclassified examples, the term $D_t(i) y_i h_t(x_i)$ is $-D_t(i)$
Now, we can substitute the values:

$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\big(\sum_{y_i = h_t(x_i)} D_t(i) - \sum_{y_i \neq h_t(x_i)} D_t(i)\big)$$

The first summation $\sum_{y_i = h_t(x_i)} D_t(i)$ represents the total weight of the correctly classified examples. But given that the total weight of all examples is 1 (since D_t is a distribution), the weight of correctly classified examples is $1 - \sum_{y_i \neq h_t(x_i)} D_t(i)$.
We can substitute this into the equation:

$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\big(1 - 2 \sum_{y_i \neq h_t(x_i)} D_t(i)\big)$$

$$\epsilon_t = \frac{1}{2} - \frac{1}{2} + \sum_{y_i \neq h_t(x_i)} D_t(i)$$

$$\epsilon_t = \sum_{y_i \neq h_t(x_i)} D_t(i)$$

Therefore, we proved the need statement.

3. [20 points] Can you figure out an equivalent linear classifier for the following Boolean functions? Please point out what the weight vector, the bias parameter and the hyperplane are. Note that the hyperplane is determined by an equation. If you cannot find out a linear classifier, please explain why, and work out some feature mapping such that, after mapping all the inputs of these functions into a higher dimensional space, there is a hyperplane that well separates the inputs; please write down the separating hyperplane in the new feature space.

   (a) [2 point] $f(x_1, x_2, x_3) = x_1 \wedge \neg x_2 \wedge \neg x_3$

   (b) [2 point] $f(x_1, x_2, x_3) = \neg x_1 \vee \neg x_2 \vee \neg x_3$

   (c) [8 points] $f(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee x_4)$

   (d) [8 points] $f(x_1, x_2) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$

Solution.

a) The only input that results in a 'true' output for this function is $(1, 0, 0)$. All other input combinations will result in 'false'. So, a linear classifier can be constructed for this function.

Linear threshold function: $w_1x_1 + w_2x_2 + w_3x_3 + b \geq 0$

Setting the weights and bias such that: $w = [2, -1, -1]$, $b = -1$, will classify $(1, 0, 0)$ correctly as 'true' (since $2 - 1 \geq 0$) and all other input combinations as 'false'. Therefore, the linear threshold function is: $2x_1 - x_2 - x_3 - 1 \geq 0$ or $2x_1 - x_2 - x_3 \geq 1$.

The hyperplane is determined by the equation: $2x_1 - x_2 - x_3 - 1 = 0$.

b) This function results in 'true' for all input combinations except $(1, 1, 1)$. We can also construct a linear classifier for this function.

Linear threshold function: $w_1x_1 + w_2x_2 + w_3x_3 + b \leq 0$

Setting the weights and bias such that: $w = [-1, -1, -1]$, $b = 2$, will classify $(1, 1, 1)$ correctly as 'false' and all other input combinations as 'true'. Therefore, the linear threshold function is: $-x_1 - x_2 - x_3 + 2 \leq 0$ or $-x_1 - x_2 - x_3 \leq -2$.

The hyperplane is determined by the equation: $-x_1 - x_2 - x_3 + 2 = 0$.

c) This function isn't linearly separable. However, let's perform a feature mapping. In this higher dimensional space, the function can be represented as:

$\phi(x_1, x_2, x_3, x_4) = (x_1x_3, x_1x_4, x_2x_3, x_2x_4)$

For the function to be true, at least one product from each pair (either x_1 or x_2 with x_3 or x_4) should be 1. So, for an input like $(1, 0, 1, 0)$, the mapped output is $(1, 0, 0, 0)$, and the sum of these is 1.

Considering that any single product being 1 is enough for f to be true, the separating hyperplane could be:

$x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 - 0.5 = 0$

Bias b for this would be -0.5.

d) This function results in 'true' for inputs $(0, 0)$ and $(1, 1)$ and 'false' for $(1, 0)$ and $(0, 1)$. This function isn't linearly separable in its current form. However, we can perform a feature mapping:

$\phi(x_1, x_2) = (x_1^2, x_2^2, x_1x_2)$

In the mapped space, $(1, 1)$ maps to $(1, 1, 1)$, $(0, 0)$ maps to $(0, 0, 0)$, $(1, 0)$ maps to $(1, 0, 0)$, and $(0, 1)$ maps to $(0, 1, 0)$.

Considering this, a hyperplane that separates the true values from the false values could be:
$x_1^2 + x_2^2 - x_1 x_2 - 1 = 0$
Bias b for this would be -1.

4. [**Bonus**] [8 points] Given two vectors $\mathbf{x} = [x_1, x_2]$ and $\mathbf{y} = [y_1, y_2]$, find a feature mapping $\phi(\cdot)$ for each of the following functions, such that the function is equal to the inner product between the mapped feature vectors, $\phi(\mathbf{x})$ and $\phi(\mathbf{y})$. For example, $(\mathbf{x}^\top \mathbf{y})^0 = \phi(\mathbf{x})^\top \phi(\mathbf{y})$ where $\phi(\mathbf{x}) = [1]$ and $\phi(\mathbf{y}) = [1]$; $(\mathbf{x}^\top \mathbf{y})^1 = \phi(\mathbf{x})^\top \phi(\mathbf{y})$ where $\phi(\mathbf{x}) = \mathbf{x}$ and $\phi(\mathbf{y}) = \mathbf{y}$.

   (a) [2 points] $(\mathbf{x}^\top \mathbf{y})^2$

   (b) [2 points] $(\mathbf{x}^\top \mathbf{y})^3$

   (c) [4 points] $(\mathbf{x}^\top \mathbf{y})^k$ where k is any positive integer.

   Solution.
   a) $(\mathbf{x}^\top \mathbf{y})^2$
   Expanding the term gives:
   $(\mathbf{x}^\top \mathbf{y})^2 = (x_1 y_1 + x_2 y_2)^2$
   $= x_1^2 y_1^2 + 2 x_1 x_2 y_1 y_2 + x_2^2 y_2^2$
   This suggests that the feature mapping $\phi(\cdot)$ for vector $\mathbf{x}$ and $\mathbf{y}$ should be:
   $\phi(\mathbf{x}) = [x_1^2, \sqrt{2} x_1 x_2, x_2^2]$
   $\phi(\mathbf{y}) = [y_1^2, \sqrt{2} y_1 y_2, y_2^2]$
   b) $(\mathbf{x}^\top \mathbf{y})^3$
   Expanding:
   $(\mathbf{x}^\top \mathbf{y})^3 = (x_1 y_1 + x_2 y_2)^3$
   $= x_1^3 y_1^3 + 3 x_1^2 x_2 y_1^2 y_2 + 3 x_1 x_2^2 y_1 y_2^2 + x_2^3 y_2^3$
   This leads to the feature mapping:
   $\phi(\mathbf{x}) = [x_1^3, \sqrt{3} x_1^2 x_2, \sqrt{3} x_1 x_2^2, x_2^3]$
   $\phi(\mathbf{y}) = [y_1^3, \sqrt{3} y_1^2 y_2, \sqrt{3} y_1 y_2^2, y_2^3]$
   c) $(\mathbf{x}^\top \mathbf{y})^k$ where k is any positive integer
   For general k, the feature mapping $\phi(\cdot)$ would produce all combinations of the terms of $\mathbf{x}$ and $\mathbf{y}$ up to degree k. There would be terms like $x_1^{k-i} x_2^i$ for each i from 0 to k.
   The exact form of $\phi(\mathbf{x})$ and $\phi(\mathbf{y})$ for general k would involve a lot of binomial coefficients (from the binomial theorem). For example, the coefficient of $x_1^{k-2} x_2^2$ when expanding the polynomial would be $\binom{k}{2}$. The form would be cumbersome to write out in full, but conceptually, it's a mapping to all combinations of the terms of the vectors up to degree k.
   In conclusion, the feature mapping $\phi(\cdot)$ essentially captures all the possible interaction terms of a given degree between the elements of vectors $\mathbf{x}$ and $\mathbf{y}$.

5. [10 points] Suppose we have the training data shown in Table 1, from which we want to learn a linear regression model, parameterized by a weight vector $\mathbf{w}$ and a bias parameter b.

x_1	x_2	x_3	y
1	-1	2	1
1	1	3	4
-1	1	0	-1
1	2	-4	-2
3	-1	-1	0

Table 1: Linear regression training data.

(a) [1 point] Write down the LMS (least mean square) cost function $J(\mathbf{w}, b)$.

(b) [3 points] Calculate the gradient $\frac{\nabla J}{\nabla \mathbf{w}}$ and $\frac{\nabla J}{\nabla b}$ when $\mathbf{w} = [-1, 1, -1]^\top$ and $b = -1$.

(c) [3 points] What are the optimal $\mathbf{w}$ and $\mathbf{b}$ that minimize the cost function?

(d) [3 points] Now, we want to use stochastic gradient descent to minimize $J(\mathbf{w}, b)$. We initialize $\mathbf{w} = \mathbf{0}$ and $b = 0$. We set the learning rate $r = 0.1$ and sequentially go through the 5 training examples. Please list the stochastic gradient in each step and the updated $\mathbf{w}$ and b.

Solution.

a) The LMS (least mean square) cost function $J(\mathbf{w}, b)$ is defined as the mean of the squared differences between the predicted and actual values:

$$J(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{w}^T \mathbf{x}^{(i)} + b - y^{(i))^2}$$

Where m is the number of training examples.

b) To calculate the gradient with respect to $\mathbf{w}$ and b, we need to differentiate the cost function. For simplicity, let's denote the error for each example as:

$$e^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)} + b - y^{(i)}$$

The gradient with respect to $\mathbf{w}$ is:

$$\frac{\nabla J}{\nabla \mathbf{w}} = \frac{2}{m} \sum_{i=1}^{m} e^{(i)} \mathbf{x}^{(i)}$$

And the gradient with respect to b is:

$$\frac{\nabla J}{\nabla b} = \frac{2}{m} \sum_{i=1}^{m} e^{(i)}$$

Using the given values $\mathbf{w} = [-1, 1, -1]^T$ and $b = -1$, we'll compute the error $e^{(i)}$ for each example and then compute the gradients.

Using the training data and plugging in the values:

$$e^{(1)} = [-1, 1, -1] \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix} - 1 - 1 = 0$$

$$e^{(2)} = [-1, 1, -1] \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix} - 1 - 4 = -4$$

$$e^{(3)} = [-1, 1, -1] \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} - 1 + 1 = -1$$

$$e^{(4)} = [-1, 1, -1] \begin{bmatrix} 1 \\ 2 \\ -4 \end{bmatrix} - 1 + 2 = 4$$

$$e^{(5)} = [-1, 1, -1] \begin{bmatrix} 3 \\ -1 \\ -1 \end{bmatrix} - 1 - 0 = 2$$

Using the above errors:

$$\frac{\nabla J}{\nabla \mathbf{w}} = \frac{2}{5}[0 + (-4) + 1 + 4 + 6, 0 + (-4) - 1 + 8 - 2, 0 + (-12) + 0 - 8 - 2] = [1.6, 0.2, -2.4]$$

$$\frac{\nabla J}{\nabla b} = \frac{2}{5}(0 - 4 - 1 + 4 + 2) = 0.2$$

c) The optimal parameters $\mathbf{w}$ and b can be determined using the normal equations for linear regression.

Given the training set, we can construct the design matrix X and the target vector y. First, we augment the input data with a column of ones to account for the bias term b:

$$\begin{bmatrix} 1 & 1 & -1 & 2 \\ 1 & 1 & 1 & 3 \\ 1 & -1 & 1 & 0 \\ 1 & 1 & 2 & -4 \\ 1 & 3 & -1 & -1 \end{bmatrix}$$

And the target vector is:

$$y = \begin{bmatrix} 1 \\ 4 \\ -1 \\ -2 \\ 0 \end{bmatrix}$$

Now, the normal equations for linear regression are given by:

$$\mathbf{w} = (X^T X)^{-1} X^T y$$

Where $\mathbf{w}$ will be a vector of coefficients, the first of which will be the bias term b (because we augmented X with a column of ones). By calculating:

$$X^T X = \begin{bmatrix} 5 & 5 & 2 & 0 \\ 5 & 13 & 0 & 0 \\ 2 & 0 & 6 & 8 \\ 0 & 0 & 8 & 30 \end{bmatrix}$$

And the inverse is:

$$(X^T X)^{-1} = \begin{bmatrix} 5.2 & -1.96 & -0.39 & -0.16 \\ -1.96 & 0.77 & 0.15 & 0.06 \\ -0.39 & 0.15 & 0.17 & -0.04 \\ -0.16 & 0.06 & -0.04 & 0.03 \end{bmatrix}$$

And:

$$X^T y = \begin{bmatrix} 2 \\ 5 \\ -2 \\ -3 \end{bmatrix}$$

Finally, we can compute $\mathbf{w}$:

$$\mathbf{w} = \begin{bmatrix} 5.2 & -1.96 & -0.39 & -0.16 \\ -1.96 & 0.77 & 0.15 & 0.06 \\ -0.39 & 0.15 & 0.17 & -0.04 \\ -0.16 & 0.06 & -0.04 & 0.03 \end{bmatrix} \begin{bmatrix} 2 \\ 5 \\ -2 \\ -3 \end{bmatrix} = \begin{bmatrix} 4.44 \\ -5.06 \\ -1.53 \\ 0.49 \end{bmatrix}$$

Thus, from this vector, the bias term b is 4.44 and the weights are $\mathbf{w} = [-5.06, -1.53, 0.49]$.
d) Stochastic Gradient Descent for Linear Regression:
Initialization:
$\mathbf{w} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$
$b = 0$
$r = 0.1$
We will now go through each of the 5 training examples.
1. Using the first data point:
$x^{(1)} = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}$, $y^{(1)} = 1$ Prediction:
$\hat{y}^{(1)} = \mathbf{w}^T x^{(1)} + b = 0 + 0 = 0$
Gradient w.r.t. $\mathbf{w}$ and b:
$\frac{\nabla J}{\nabla \mathbf{w}} = (\hat{y}^{(1)} - y^{(1)}) x^{(1)} = \begin{bmatrix} -1 \\ 1 \\ -2 \end{bmatrix}$
$\frac{\nabla J}{\nabla \mathbf{w}} = (\hat{y}^{(1)} - y^{(1)}) = -1$ Updating $\mathbf{w}$ and b using the gradients:
$\mathbf{w} = \mathbf{w} - r\frac{\nabla J}{\nabla \mathbf{w}} = \begin{bmatrix} 0.1 \\ -0.1 \\ 0.2 \end{bmatrix}$
$b = b - r\frac{\nabla J}{\nabla b} = 0.1$
2. Using the second data point:
$x^{(2)} = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix}$, $y^{(2)} = 4$ Prediction:

$\hat{y}^{(2)} = \mathbf{w}^T x^{(2)} + b = 0.7$
Gradient w.r.t. $\mathbf{w}$ and b:
$\frac{\nabla J}{\nabla \mathbf{w}} = (\hat{y}^{(2)} - y^{(2)})x^{(2)} = \begin{bmatrix} -3.3 \\ -3.3 \\ -9.9 \end{bmatrix}$
$\frac{\nabla J}{\nabla \mathbf{w}} = (\hat{y}^{(2)} - y^{(2)}) = -3.3$ Updating $\mathbf{w}$ and b using the gradients:
$\mathbf{w} = \mathbf{w} - r\frac{\nabla J}{\nabla \mathbf{w}} = \begin{bmatrix} 0.43 \\ 0.23 \\ 1.19 \end{bmatrix}$
$b = 0.43$
3. Using the second data point:
$x^{(3)} = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$, $y^{(3)} = -1$ Prediction:
$\hat{y}^{(3} = \mathbf{w}^T x^{(3)} + b = 0.66$
Gradient w.r.t. $\mathbf{w}$ and b:
$\frac{\nabla J}{\nabla \mathbf{w}} = (\hat{y}^{(3)} - y^{(3)})x^{(3)} = \begin{bmatrix} 2.66 \\ -2.66 \\ 0 \end{bmatrix}$
$\frac{\nabla J}{\nabla \mathbf{w}} = (\hat{y}^{(3)} - y^{(3)}) = 1.66$ Updating $\mathbf{w}$ and b using the gradients:
$\mathbf{w} = \mathbf{w} - r\frac{\nabla J}{\nabla \mathbf{w}} = \begin{bmatrix} 0.264 \\ 0.396 \\ 1.19 \end{bmatrix}$
$b = 0.264$
4. Using the second data point:
$x^{(4)} = \begin{bmatrix} 1 \\ 2 \\ -4 \end{bmatrix}$, $y^{(4)} = -2$ Prediction:
$\hat{y}^{(4} = \mathbf{w}^T x^{(4)} + b = -3.066$
Gradient w.r.t. $\mathbf{w}$ and b:
$\frac{\nabla J}{\nabla \mathbf{w}} = (\hat{y}^{(4)} - y^{(4)})x^{(4)} = \begin{bmatrix} -1.066 \\ -2.132 \\ 4.264 \end{bmatrix}$
$\frac{\nabla J}{\nabla \mathbf{w}} = (\hat{y}^{(4)} - y^{(4)}) = -1.066$ Updating $\mathbf{w}$ and b using the gradients:
$\mathbf{w} = \mathbf{w} - r\frac{\nabla J}{\nabla \mathbf{w}} = \begin{bmatrix} 0.3696 \\ 0.6092 \\ 0.8136 \end{bmatrix}$
$b = 0.3706$
5. Using the second data point:
$x^{(5)} = \begin{bmatrix} 3 \\ -1 \\ -1 \end{bmatrix}$, $y^{(5)} = 0$ Prediction:
$\hat{y}^{(5} = \mathbf{w}^T x^{(5)} + b = 0.5626$
Gradient w.r.t. $\mathbf{w}$ and b:

$\frac{\nabla J}{\nabla \mathbf{w}} = (\hat{y}^{(5)} - y^{(5)})x^{(5)} = \begin{bmatrix} 1.6878 \\ -0.5626 \\ -0.5626 \end{bmatrix}$
$\frac{\nabla J}{\nabla \mathbf{w}} = (\hat{y}^{(5)} - y^{(5)}) = 0.5626$ Updating $\mathbf{w}$ and b using the gradients:
$\mathbf{w} = \mathbf{w} - r\frac{\nabla J}{\nabla \mathbf{w}} = \begin{bmatrix} 0.2005 \\ 0.6654 \\ 0.8689 \end{bmatrix}$
$b = 0.31434$

Therefore, at the end of the first iteration through the dataset, we have the following $\mathbf{w}$ and b:
$\mathbf{w} = \begin{bmatrix} 0.2005 \\ 0.6654 \\ 0.8689 \end{bmatrix}$
$b = 0.31434$

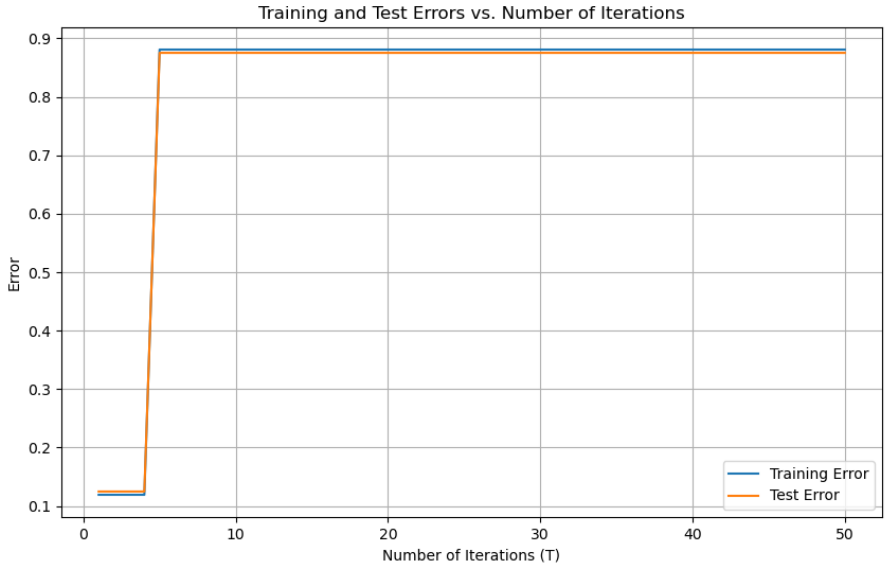
2 Practice [60 points + 10 bonus]

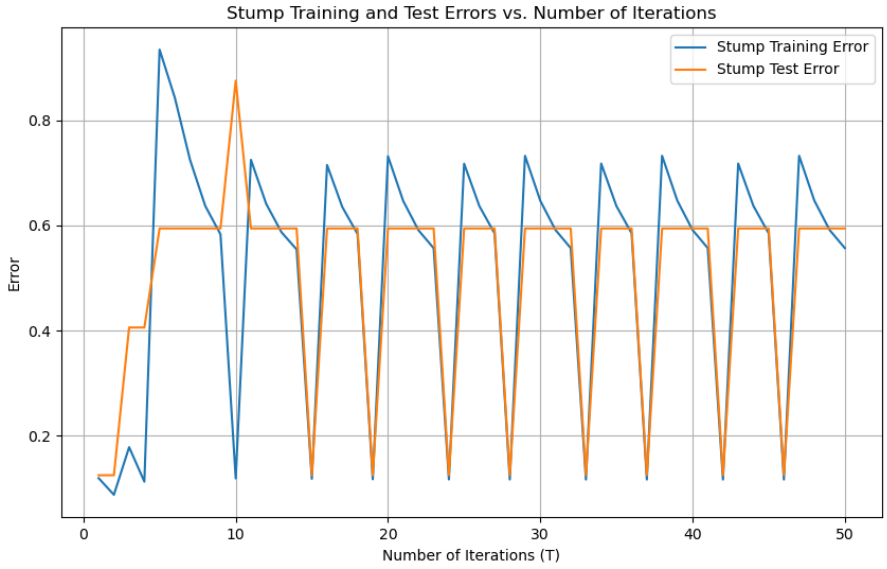
1. [2 Points] Update your machine learning library. Please check in your implementation of decision trees in HW1 to your GitHub repository. Remember last time you created a folder "Decision Tree". You can commit your code into that folder. Please also supplement README.md with concise descriptions about how to use your code to learn decision trees (how to call the command, set the parameters, etc). Please create two folders "Ensemble Learning" and "Linear Regression" in the same level as the folder "Decision Tree".

2. [36 points] We will implement the boosting and bagging algorithms based on decision trees. Let us test them on the bank marketing dataset in HW1 (bank.zip in Canvas). We use the same approach to convert the numerical features into binary ones. That is, we choose the media (NOT the average) of the attribute values (in the training set) as the threshold, and examine if the feature is bigger (or less) than the threshold. For simplicity, we treat "unknown" as a particular attribute value, and hence we do not have any missing attributes for both training and test.

   (a) [8 points] Modify your decision tree learning algorithm to learn decision stumps — trees with only two levels. Specifically, compute the information gain to select the best feature to split the data. Then for each subset, create a leaf node. Note that your decision stumps must support weighted training examples. Based on your decision stump learning algorithm, implement AdaBoost algorithm. Vary the number of iterations T from 1 to 500, and examine the training and test errors. You should report the results in two figures. The first figure shows how the training and test errors vary along with T. The second figure shows the training and test errors of all the decision stumps learned in each iteration. What can you observe and conclude? You have had the results for a fully expanded decision tree

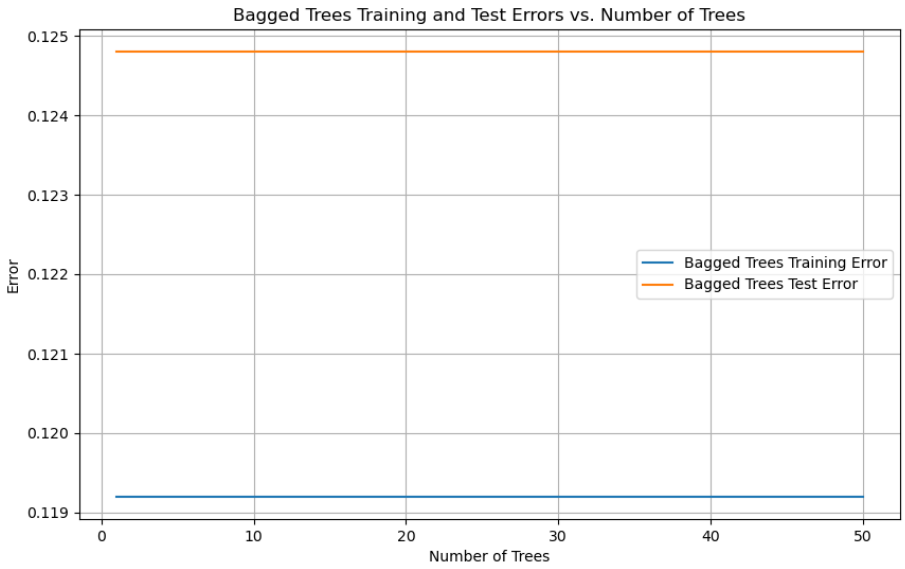
in HW1. Comparing them with Adaboost, what can you observe and conclude?
Solution. The code is implemented in GitHub (https://github.com/milenabel/CS6350-ML2023/tree/main/EnsembleLearning).
Due to the long runtime of the algorithms, I only managed to do 1 to 50 iterations.





(b) [8 points] Based on your code of the decision tree learning algorithm (with information gain), implement a Bagged trees learning algorithm. Note that each tree should be fully expanded — no early stopping or post pruning. Vary the number of trees from 1 to 500, report how the training and test errors vary along with the tree number in a figure. Overall, are bagged trees better than a single tree? Are bagged trees better than Adaboost?
Solution. The code is implemented in GitHub (https://github.com/milenabel/CS6350-

ML2023/tree/main/EnsembleLearning).
Due to the long runtime of the algorithms, I only managed to do 1 to 50 iterations.



Bagged trees, in general, tend to perform better than a single decision tree because they reduce variance and help to avoid overfitting. They are also better than AdaBoost according to the figure provided.

(c) [6 points] Through the bias and variance decomposition, we have justified why the bagging approach is more effective than a single classifier/predictor. Let us verify it in real data. Experiment with the following procedure.

- REPEAT for 100 times
- [STEP 1] Sample $1,000$ examples *uniformly without replacement* from the training datset
- [STEP 2] Run your bagged trees learning algorithm based on the $1,000$ training examples and learn 500 trees.
- END REPEAT
- Now you have 100 bagged predictors in hand. For comparison, pick the first tree in each run to get 100 fully expanded trees (i.e. single trees).
- For each of the test example, compute the predictions of the 100 single trees. Take the average, subtract the ground-truth label, and take square to compute the bias term (see the lecture slides). Use all the predictions to compute the sample variance as the approximation to the variance term (if you forget what the sample variance is, check it out here). You now obtain the bias and variance terms of a single tree learner for one test example. You will need to compute them for all the test examples and then take average as your final estimate of the bias and variance terms for the single decision tree learner. You can add the two terms to obtain the estimate of the general squared error (that is, expected error w.r.t test examples). Now use your 100 bagged predictors to do the same thing and estimate the general bias and

variance terms, as well as the general squared error. Comparing the results of the single tree learner and the bagged trees, what can you conclude? What causes the difference?

(d) [8 points] Implement the random forest algorithm as we discussed in our lecture. Vary the number of random trees from 1 to 500. Note that you need to modify your tree learning algorithm to randomly select a subset of features before each split. Then use the information gain to select the best feature to split. Vary the size of the feature subset from $\{2, 4, 6\}$. Report in a figure how the training and test errors vary along with the number of random trees for each feature subset size setting. How does the performance compare with bagged trees?

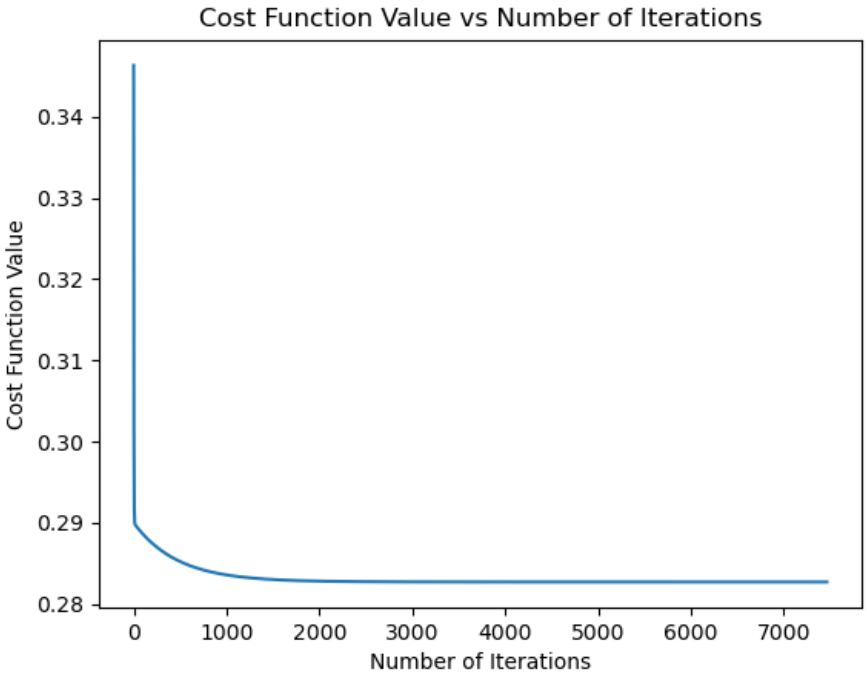
(e) [6 points] Following (c), estimate the bias and variance terms, and the squared error for a single random tree and the whole forest. Comparing with the bagged trees, what do you observe? What can you conclude?

3. [**Bonus**][10 points] In practice, to confirm the performance of your algorithm, you need to find multiple datasets for test (rather than one). You need to extract and process data by yourself. Now please use the credit default dataset in UCI repository https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients. Randomly choose 24000 examples for training and the remaining 6000 for test. Feel free to deal with continuous features. Run bagged trees, random forest, and Adaboost with decision stumps algorithms for 500 iterations. Report in a figure how the training and test errors vary along with the number of iterations, as compared with a fully expanded single decision tree. Are the results consistent with the results you obtained from the bank dataset?

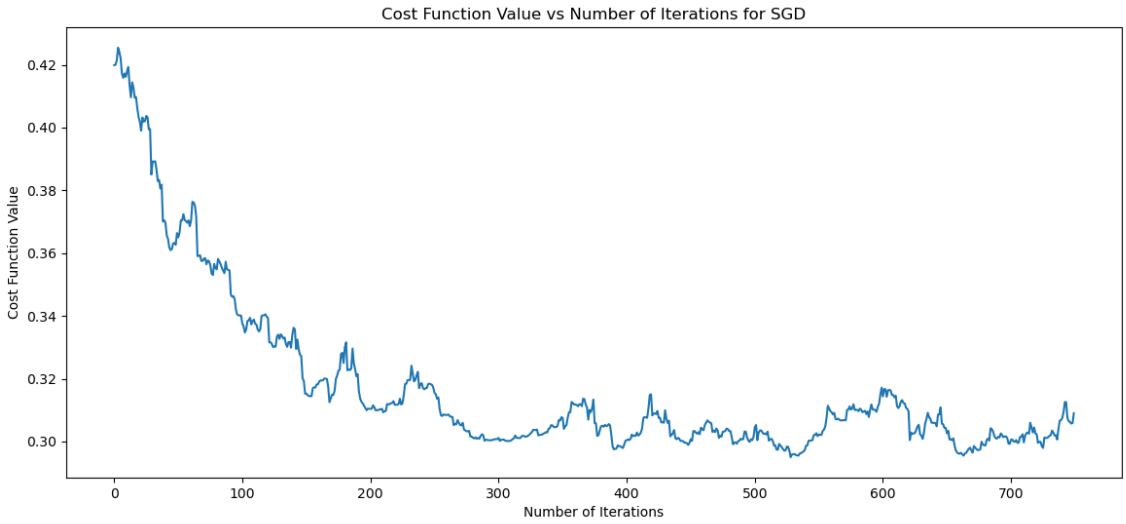
4. [22 points] We will implement the LMS method for a linear regression task. The dataset is from UCI repository (`https://archive.ics.uci.edu/ml/datasets/Concrete+Slump+Test`). The task is to predict the real-valued SLUMP of the concrete, with 7 features. The features and output are listed in the file "concrete/data-desc.txt". The training data are stored in the file "concrete/train.csv", consisting of 53 examples. The test data are stored in "concrete/test.csv", and comprise of 50 examples. In both the training and testing datasets, feature values and outputs are separated by commas.

   (a) [8 points] Implement the batch gradient descent algorithm, and tune the learning rate r to ensure the algorithm converges. To examine convergence, you can watch the norm of the weight vector difference, $\|w_t - w_{t-1}\|$, at each step t. if $\|w_t - w_{t-1}\|$ is less than a tolerance level, say, 10^{-6}, you can conclude that it converges. You can initialize your weight vector to be $\mathbf{0}$. Please find an appropriate r such that the algorithm converges. To tune r, you can start with a relatively big value, say, $r = 1$, and then gradually decrease r, say $r = 0.5, 0.25, 0.125, \ldots$, until you see the convergence. Report the learned weight vector, and the learning rate r. Meanwhile, please record the cost function value of the training data at each step, and then draw a figure shows how the cost function changes along with steps. Use your final weight vector to calculate the cost function value of the test data.

Solution. The code is implemented in GitHub (https://github.com/milenabel/CS6350-ML2023/tree/main/LinearRegression).
Learned **GDA** Weight Vector: [−0.01520404 0.90020422 0.78592199 0.85064194 1.29860639 0.12983046 1.57176489 0.99832589].
Learning rate: 0.5.
Test Cost: 0.4672255434765307



(b) [8 points] Implement the stochastic gradient descent (SGD) algorithm. You can initialize your weight vector to be **0**. Each step, you randomly sample a training example, and then calculate the stochastic gradient to update the weight vector. Tune the learning rate r to ensure your SGD converges. To check convergence, you can calculate the cost function of the training data after each stochastic gradient update, and draw a figure showing how the cost function values vary along with the number of updates. At the beginning, your curve will oscillate a lot. However, with an appropriate r, as more and more updates are finished, you will see the cost function tends to converge. Please report the learned weight vector, and the learning rate you chose, and the cost function value of the test data with your learned weight vector.
Solution. The code is implemented in GitHub (https://github.com/milenabel/CS6350-ML2023/tree/main/LinearRegression).
Learned **SGD** Weight Vector: [−0.03969494 − 0.0385621 − 0.24910744 − 0.10948452 0.39582832 − 0.09827536 0.1331324 − 0.03406326].
Learning rate: 0.25.
Test Cost: 0.4202370852021848



(c) [6 points] We have discussed how to calculate the optimal weight vector with an analytical form. Please calculate the optimal weight vector in this way. Comparing with the weight vectors learned by batch gradient descent and stochastic gradient descent, what can you conclude? Why?

Solution.

Optimal Weight Vector (Analytical): [−0.01519667 0.90056451 0.78629331 0.85104314 1.29889413 0.12989067 1.57224887 0.99869359]

Observations:

- Similarity between Analytical and GDA Solutions:

- - The weight vector obtained from the Gradient Descent Algorithm (GDA) is very close to the optimal weight vector obtained analytically.

- - This indicates that GDA has converged to a solution that is almost identical to the true optimal solution.

- Dissimilarity between Analytical and SGD Solutions:

- - The weight vector obtained from the Stochastic Gradient Descent (SGD) is different from the optimal weight vector.

- - This is typical for SGD, as it is a stochastic algorithm and may not converge to the exact optimal solution. However, the cost function value indicates that the solution is reasonable.

Conclusion:

- Gradient Descent Algorithm (GDA) is Very Effective:

- - The GDA has effectively converged to a solution that is almost identical to the true optimal solution obtained analytically. This is a strong indication that GDA is a powerful method for this particular problem.

- Stochastic Gradient Descent (SGD) is Effective but Less Precise:

- - The SGD has found a reasonable solution that, while not identical to the op-

timal solution, still provides a good approximation. The nature of SGD means it is less precise than GDA, but it can be more efficient for very large datasets.

- Learning Rate Tuning is Crucial:
- - The best learning rate for GDA was 0.5, while for SGD it was 0.25. This emphasizes the importance of tuning the learning rate to find the most effective solution.
Overall, both gradient descent algorithms have proven to be effective methods for solving this problem, with GDA providing a more precise solution that is closer to the true optimal weight vector.