CSE 417T: Homework 4

Hangxiao Zhu

December 2, 2022

Problem 1.

Results:
For each of the binary classification problems, graphically report the training set error and the test set error as a function of the number of weak hypotheses.

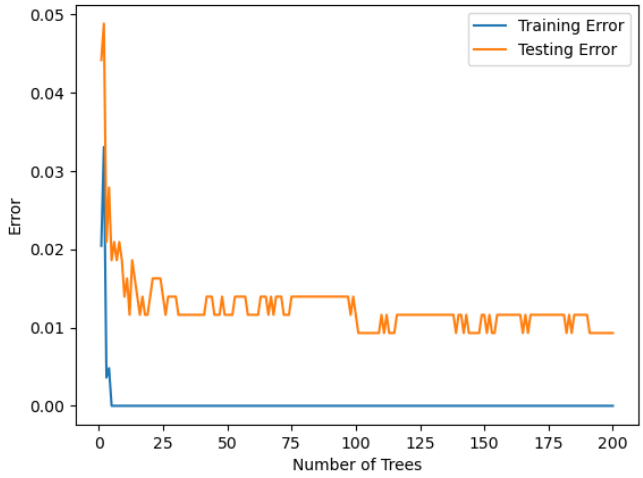


Figure 1: "1-vs-3" Problem

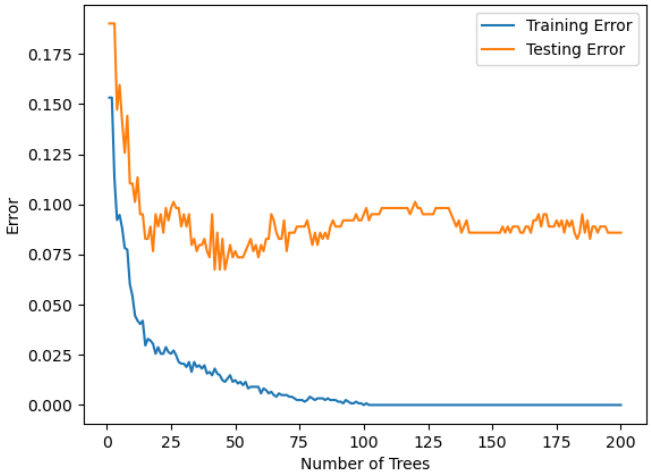


Figure 2: "3-vs-5" Problem

For each of the binary classification problems, report the final training set error and test set error.

	Training error	Test error
"1-vs-3" Problem	0.0	0.009302325581395349
"3-vs-5" Problem	0.0	0.08588957055214724

Summary and interpretation of the results:

- Compare the test error of AdaBoost to the test error of Bagging, it's clear that AdaBoost performs better.
- As the number of weak hypotheses increases, both the training and test error decrease.
- According to the test erros to each problem, AdaBoost performs better on "1-vs-3" Problem. I think it's because 1 and 3 have more distinct characteristic differences, such as symmetries.
- According to the curves, AdaBoost shows robustness to overfitting. I think it's because we used the very simple weak learner 'decision stumps'. However, for "3-vs-5" Problem, AdaBoost shows signs of overfitting. I think it's because the data of number 5 is more noisy.

Problem 2.

With the test example $x = 3.2$, we can its determine the 3 nearest neighbors are $(3, 5)$, $(2, 11)$, $(3, 8)$. Therefore we can predict this test example's y value as:

$$\begin{aligned} g(x) &= \frac{1}{k}\sum_{i=1}^{k} y_{[i]}(x) \\ &= \frac{1}{3}\sum_{i=1}^{3} y_{[i]}(3.2) \\ &= \frac{1}{3}(5 + 11 + 8) \\ &= 8 \end{aligned}$$

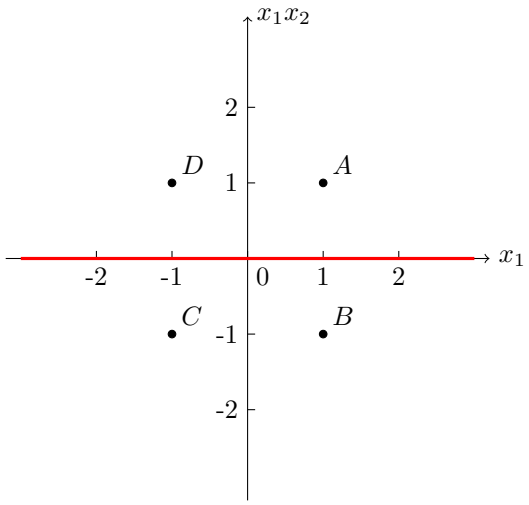
Therefore, I would predict $y = 8$ for this test example.

Problem 3.

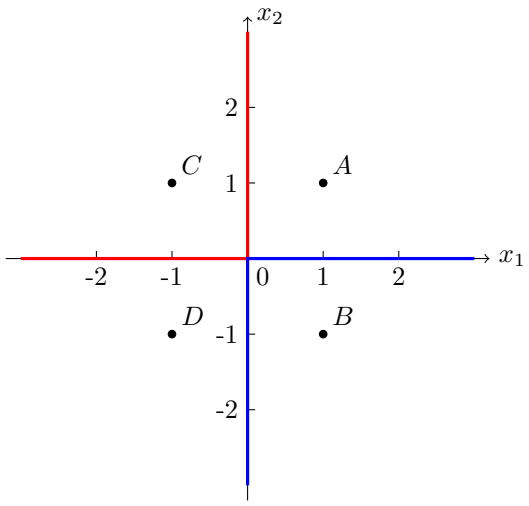
First, we map the input (x_1, x_2) into the space consisting of x_1 and x_1x_2.

	x_1	x_2	x_1	x_1x_2
A	1	1	1	1
B	1	-1	1	-1
C	-1	1	-1	-1
D	-1	-1	-1	1

We know points A and D should be labeled as -1, and points B and C should be labeled as +1.
Then, we draw the four input points in this space with the maximal margin separator. The maximal margin separator is the line $x_1x_2 = 0$ and the margin is 1.



Now, we draw the four input points back in the original Euclidean input space with the separating line.



Problem 4.

First, we denote the squared Euclidean distance of the two points $\Phi(\vec{x_i})$ and $\Phi(\vec{x_j})$ as sd_E. And we assume each $\Phi(\vec{x})$ has a dimension of n.
We can deduce that

$$\begin{aligned} sd_E &= \sum_{k=1}^{n} (\Phi(\vec{x_i})_k - \Phi(\vec{x_j})_k)^2 \\ &= \sum_{k=1}^{n} (\Phi(\vec{x_i})_k)^2 + \sum_{k=1}^{n} (\Phi(\vec{x_j})_k)^2 - \sum_{k=1}^{n} 2\Phi(\vec{x_i})_k \Phi(\vec{x_j})_k \\ &= \Phi(\vec{x_i})^T \Phi(\vec{x_i}) + \Phi(\vec{x_j})^T \Phi(\vec{x_j}) - 2\Phi(\vec{x_i})^T \Phi(\vec{x_j}) \end{aligned}$$

Since we have the kernel function

$$K(\vec{x_i}, \vec{x_j}) = \Phi(\vec{x_i})^T \Phi(\vec{x_j})$$

We can write down the squared Euclidean distance using the kernel function K as

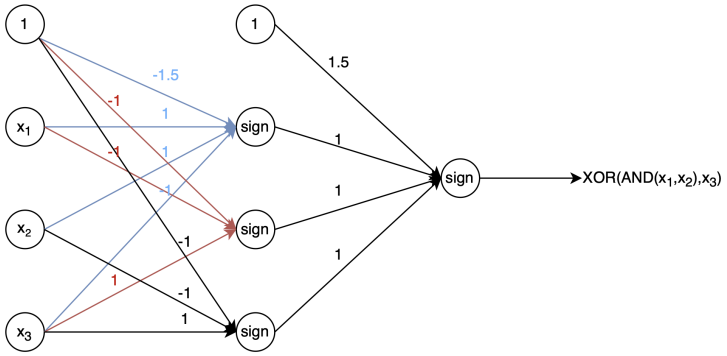
$$sd_E = K(\vec{x_i}, \vec{x_i}) + K(\vec{x_j}, \vec{x_j}) - 2K(\vec{x_i}, \vec{x_j})$$

Problem 5.

First, we simplify the Boolean formula.

$$\begin{aligned} XOR(AND(x_1, x_2), x_3) &= (x_1 \wedge x_2) \oplus x_3 \\ &= ((x_1 \wedge x_2) \wedge \neg x_3) \vee (\neg(x_1 \wedge x_2) \wedge x_3) \\ &= (x_1 \wedge x_2 \wedge \neg x_3) \vee ((\neg x_1 \vee \neg x_2) \wedge x_3) \\ &= (x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_3) \vee (\neg x_2 \wedge x_3) \end{aligned}$$

Then, design the neural network.



x

Figure 3: Neural Network for XOR(AND(x_1, x_2), x_3)

Problem 6.

a. Given that

$$\tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$
$$\tanh'(s) = 1 - \tanh^2(s)$$

We have

$$\begin{aligned}
\nabla \mathbb{E}_{in}(\vec{w}) &= \nabla \frac{1}{N} \sum_{n=1}^{N} (\tanh(\vec{w}^T \vec{x}_n) - y_n)^2 \\
&= \frac{1}{N} \sum_{n=1}^{N} 2(\tanh(\vec{w}^T \vec{x}_n) - y_n)(\frac{\mathrm{d}\tanh(\vec{w}^T \vec{x}_n)}{\mathrm{d}\vec{w}}) \\
&= \frac{1}{N} \sum_{n=1}^{N} 2(\tanh(\vec{w}^T \vec{x}_n) - y_n)(1 - \tanh^2(\vec{w}^T \vec{x}_n))\vec{x}_n \\
&= \frac{2}{N} \sum_{n=1}^{N} (\tanh(\vec{w}^T \vec{x}_n) - y_n)(1 - \tanh^2(\vec{w}^T \vec{x}_n))\vec{x}_n
\end{aligned}$$

Based on this equation, when $\vec{w}$ consists large weights, that $\vec{w} \to \infty$, $\tanh(\vec{w}) \to 1$, $\nabla \mathbb{E}_{in}(\vec{w}) \to 0$. That means with large weights, the gradient will not change, thus the hypothesis will not be improved.

Based on the conclusion above, when using gradient descent to optimize the weights of a sigmoidal perceptron, it is not a good idea to initialize the weights to be very, very large.

b. If all the weights are set to zero, then each output from the input layer will be zero. That means each $y_n = \vec{w}^T \vec{x}_n = 0$, and $\tanh(\vec{w}^T \vec{x}_n) = 0$. Based on the equation in question a, we know $\nabla \mathbb{E}_{in}(\vec{w}) = 0$. Thus, when we use gradient descent to update weights, weights will remain zero. Therefore, it is not a good idea to initialize the weights to zero.