Homework # 3

TJ Bai

November 15, 2023

1. We claim that there does exist a decision tree that can classify these N points. Intuitively, one could construct a decision tree where the boundaries approximate the linear boundary in a "jigsaw" pattern. With enough training, this "jigsaw" becomes granular enough to correctly classify all points.

   One might consider some kind of greedy algorithm to generate this decision tree. At each step, we find some axis-aligned boundary that correctly classifies at least half the points in provided region. It is intuitive that we could obtain this axis-aligned boundary, because the linearly-separable nature of the data implies that by translating some boundary along x_1 or x_2, the number of points classified increases/decreases linearly.

   With each iterate, we create 2 more half-spaces divided by a decision boundary that follows this heuristic on the parent node. This should be possible because the half-spaces of linearly separable data are still linearly separable. With enough iterations, we ultimately correctly classify all points, noting that in the base case where there are only 2 points in the region with opposite labels, they are trivially separable with a single decision boundary.

   Ultimately, we see that the depth of this tree is bounded by $\log N$, because the final decision tree is balanced and has at most N leaf nodes.

2. Even in the case when the N points are not linearly separable, we can still optimally classify them with a tree that has N leaves, with each leaf classifying each individual point with its true label.

   We consider some iterative procedure wherein we choose an axis-aligned decision boundary such that half the points exist on one side of the boundary, and the other half exist on the other side. With each iteration, we will consider the newly created half-spaces and apply the same heuristic, such that each half-space contains approximately half the points of its parent node.

   In the base case, a region contains 2 points with opposite labels. Because these 2 points must differ at at least one dimension, we can always find some decision boundary that separates them at that dimension. In the resulting regions with only one point, we classify that region with the label of the point it contains.

   At the termination of this construction, we have constructed a balanced tree with exactly N leaves, thus the depth of the tree is $\log N$. Through our iterative procedure, we have also guaranteed that each label corresponds to one leaf in the tree which classifies it with its true label.

3. We show that the training error of classifier h_{T+1} is exactly $1/2$.

   We can express the training error of h_{T+1} as $\displaystyle\sum_{y_i \neq h_{T+1}(x_i)} \frac{1}{Z} W_i^T e^{-\alpha_{T+1} y_i h_{T+1}(x_i)}$

   where W_i^T represents the error weights from the first T iterations and Z is the normalization constant.

   When $y \neq h_{T+1}(x_i)$, $y_i h_{T+1}(x_i) = -1 \implies$ the error is equal to $\displaystyle\sum_{y_i \neq h_{T+1}(x_i)} \frac{1}{Z} W_i^T e^{\alpha_{T+1}}$

This is equivalent to $\frac{e^{\alpha_{T+1}}}{Z} \sum_{y_i \neq h_{T+1}(x_i)} W_i^T = \frac{e^{\alpha_{T+1}}}{Z} \epsilon_{T+1}$, by definition of the training error.

Similarly, $\sum_{y=h_{T+1}(x_i)} W_i^T = 1 - \sum_{y \neq h_{T+1}(x_i)} W_i^T = 1 - \epsilon_{T+1} \implies \frac{e^{-\alpha_{T+1}}}{Z} \sum_{y=h_{T+1}(x_i)} W_i^T = \frac{e^{-\alpha_{T+1}}}{Z}(1 - \epsilon_{T+1})$

Now, note that $\frac{1-\epsilon_{T+1}}{\epsilon_{T+1}} = e^{\log \frac{1-\epsilon_{T+1}}{\epsilon_{T+1}}} = e^{2\alpha_{T+1}}$ where α_{T+1} is the weight of h_{T+1}

Thus, $\frac{1-\epsilon_{T+1}}{\epsilon_{T+1}} = \frac{e^{\alpha_{T+1}}}{e^{-\alpha_{T+1}}} \implies e^{\alpha_{T+1}} \epsilon_{T+1} = e^{-\alpha_{T+1}}(1 - \epsilon_{T+1}) \implies \frac{e^{\alpha_{T+1}}}{Z} \epsilon_{T+1} = \frac{e^{-\alpha_{T+1}}}{Z}(1 - \epsilon_{T+1})$

As previously shown, this is equivalent to showing the equality $\frac{e^{\alpha_{T+1}}}{Z} \sum_{y_i \neq h_{T+1}(x_i)} W_i^T = \frac{e^{-\alpha_{T+1}}}{Z} \sum_{y=h_{T+1}(x_i)} W_i^T$

We also know that $\frac{e^{\alpha_{T+1}}}{Z} \sum_{y_i \neq h_{T+1}(x_i)} W_i^T + \frac{e^{-\alpha_{T+1}}}{Z} \sum_{y=h_{T+1}(x_i)} W_i^T = \sum_{i=1}^{N} \frac{1}{Z} W_i^T e^{-\alpha_{T+1} y_i h_{T+1}(x_i)} = 1$

Therefore, $\frac{e^{\alpha_{T+1}}}{Z} \sum_{y_i \neq h_{T+1}(x_i)} W_i^T = \frac{1}{2}$ ■

Note that we can not choose $h_{T+2} = h_{T+1}$ in the following round, because AdaBoost requires that each weak classifier has training error $< \frac{1}{2}$.

4. We show that $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$ minimizes the emprical exponential loss at iteration t.

   At iteration t, the empircal exponential loss is equal to $\sum_{i=1}^{N} e^{-y_i H_{t-1}(x_i)} e^{-\alpha_t y_i h_t(x_i)}$

   This is equivalent to $e^{-\alpha_t} \sum_{y_i = h_t(x_i)} W_i^{t-1} + e^{\alpha_t} \sum_{y \neq h_t(x_i)} W_i^{t-1} = e^{-\alpha_t} \sum_{i=1}^{N} W_i^{t-1} + (e^{\alpha_t} - e^{-\alpha_t}) \sum_{y_i \neq h_t(x_i)} W_i^{t-1}$

   Because $e^{\alpha_t} - e^{-\alpha_t} > 0$, choosing the optimal h_t is equivalent to minimizing $\sum_{y \neq h_t(x_i)} W_i^{t-1}$

   Then, for a fixed h_t which minimizes the weighted error $\sum_{y_i \neq h_t(x_i)} W_i^{t-1}$, we select the optimal α_t.

   $$\frac{\partial}{\partial \alpha_t} \left(e^{-\alpha_t} \sum_{y_i = h_t(x_i)} W_i^{t-1} + e^{\alpha_t} \sum_{y \neq h_t(x_i)} W_i^{t-1} \right) = -e^{-\alpha_t} \sum_{y_i = h_t(x_i)} W_i^{t-1} + e^{\alpha_t} \sum_{y \neq h_t(x_i)} W_i^{t-1}$$

   $= -e^{-\alpha_t}(1 - \epsilon_t) + e^{\alpha_t} \epsilon_t$, and setting this to 0 we get $e^{\alpha_t} \epsilon_t = e^{-\alpha_t}(1 - \epsilon_t)$

   $\implies \frac{e^{\alpha_t}}{e^{-\alpha_t}} = \frac{1-\epsilon_t}{\epsilon_t} \implies 2\alpha_t = \log \frac{1-\epsilon_t}{\epsilon_t} \implies \alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$ ■

5. Posing this optimization in terms of the dual:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C \\ & \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned}$$

We seek H, f, A, a, B, and b to represent this problem in the canonical form of a quadratic program.

$\alpha = (\alpha_1, \ldots, \alpha_N)^\intercal \in \mathbb{R}^N$

minimize $\frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \implies H_{ij} = y_i y_j k(x_i, x_j), H \in \mathbb{R}^{N \times N}$

minimize $-\sum_{i=1}^{N} \alpha_i \implies f = (-1, \ldots, -1) \in \mathbb{R}^N$

$\sum_{i=1}^{N} \alpha_i y_i = 0 \implies B = (y_1, \ldots, y_N)^\intercal \in \mathbb{R}^N$ and $b = 0$

We can represent $0 \leq \alpha_i \leq C$ by setting 2 constraints: $-\alpha_i \leq 0$ and $\alpha_i \leq C$

Thus, $A = (I| - I)^\intercal$ where I is the identity matrix in $\mathbb{R}^{N \times N}$ and $a = (c_1, \ldots, c_N, 0_1, \ldots, 0_N)^\intercal$

6. For linear SVM, I evaluated $C \in \{0.005, 0.05, 0.5, 1, 5, 10, 50\}$.

   As expected, there was a tradeoff between under and overfitting for varying values of C. As the size of C increases, the optimization function enforces a stricter regularization, aiming to classify more training examples correctly. Thus, training accuracy increased but value accuracy decreased. However, if C was too small (such as 0.005) then we would underfit the data, resulting in low training *and* value accuracy. I found a middling value of 0.05 to be best, providing 0.824 accuracy on the Kaggle dataset.

   For the SVM with RBF kernel, the trend for C was similar.

   γ also had an analogous trend, wherein small γ values create a broader decision boundary $\implies$ less overfitting and better generalization. In contrast, large γ values can result in overfitting. Ultimately, I found $C = 0.5$ and $\gamma = 1$ to be best.