CM146, Winter 2024
Problem Set 4: Boosting, Unsupervised learning
Due March 15, 11:59pm (Math), March 17, 11:59pm (Coding)

Harris Doan

March 15, 2024

Submission instructions

- Submit your solutions electronically on the course Gradescope site as PDF files.
- If you plan to typeset your solutions, please use the LaTeX solution template. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app.
- First three math heavy questions are due Friday March 15, 11:59pm. Coding/Implementation questions will be due on Sunday March 17, 11:59pm.

1 AdaBoost [5 pts]

In the lecture on ensemble methods, we said that in iteration t, AdaBoost is picking (h_t, β_t) that minimizes the objective:

$$\begin{aligned}(h_t^*(\boldsymbol{x}), \beta_t^*) &= \underset{(h_t(\boldsymbol{x}),\beta_t)}{\arg\min} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\ &= \underset{(h_t(\boldsymbol{x}),\beta_t)}{\arg\min} \left(e^{\beta_t} - e^{-\beta_t}\right) \sum_n w_t(n) \mathbb{I}\left[y_n \neq h_t\left(\boldsymbol{x}_n\right)\right] \\ &\quad + e^{-\beta_t} \sum_n w_t(n)\end{aligned}$$

We define the weighted misclassification error at time t, ϵ_t to be $\epsilon_t = \sum_n w_t(n) \mathbb{I}\left[y_n \neq h_t\left(\boldsymbol{x}_n\right)\right]$. Also the weights are normalized so that $\sum_n w_t(n) = 1$.

(a) Take the derivative of the above objective function with respect to β_t and set it to zero to solve for β_t and obtain the update for β_t.

Solution 1a:

Given the function:

$$J(\beta) = (e^{\beta t} - e^{-\beta t})\epsilon_t + e^{-\beta t} \tag{1}$$

We want to find the value of β that minimizes this function. To do this, we take the derivative of $J(\beta)$ with respect to β and set it equal to zero.

$$\frac{\partial J(\beta)}{\partial \beta} = \frac{\partial}{\partial \beta}\left[(e^{\beta t} - e^{-\beta t})\epsilon_t + e^{-\beta t}\right] \tag{2}$$

Applying the derivative, we get:

$$\frac{\partial J(\beta_t)}{\partial \beta_t} = (e^{\beta_t} + e^{-\beta_t})t \cdot \epsilon_t - e^{-\beta_t} t \tag{3}$$

Setting the derivative equal to zero for the minimum:

$$(e^{\beta_t} + e^{-\beta_t})t \cdot \epsilon_t - e^{-\beta_t} t = 0 \tag{4}$$

After simplifying the equation, we find that:

$$e^{2\beta_t} + 1 = \frac{1}{\epsilon_t} \tag{5}$$

Which leads to:

$$e^{2\beta_t} = \frac{1}{\epsilon_t} - 1 \tag{6}$$

And solving for β_t gives us:

$$\beta_t = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right) \tag{7}$$

Here, log denotes the natural logarithm. This value of β_t is the one that minimizes the original function $J(\beta)$.

(b) Suppose the training set is linearly separable, and we use a hard-margin linear support vector machine (no slack) as a base classifier. In the first boosting iteration, what would the resulting β_1 be?

Solution 1b:

In the scenario where the training set is perfectly linearly separable, a hard-margin Support Vector Machine (SVM) would find a hyperplane that perfectly classifies all training points with the maximum margin. The AdaBoost algorithm, which adjusts the weights of misclassified instances, would assign weights based on the training errors ϵ_t.

Since the SVM classifies all points correctly, the training error ϵ_1 would be zero. In the context of boosting, the weight of the classifier β_t is calculated as follows:

$$\beta_t = \frac{1}{2} \log \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \tag{8}$$

Given that $\epsilon_1 = 0$, we have:

$$\beta_1 = \frac{1}{2} \log \left(\frac{1}{0} \right) \tag{9}$$

However, the logarithm of zero is undefined because there is no number that you can raise to the power of e to get zero. Therefore, this would imply that β_1 approaches infinity. This can be interpreted as the classifier being very confident in its predictions since the data is perfectly separable.

In practical implementations, to handle this case, ϵ_t might be set to a very small non-zero value to avoid division by zero and to provide a finite weight to the perfect classifier.

The concept of an infinite weight β_1 can be interpreted within the AdaBoost algorithm framework. Since the base classifier (the SVM) has no errors, it would be given an infinitely strong vote in the final ensemble classifier. Intuitively, if a classifier always makes the correct prediction, it should always be trusted, hence an infinite weight in a voting system. Nevertheless, in a real-world application, we would need to cap this value to prevent computational issues.

Therefore, for this problem β_1 would be ∞

2 K-means for single dimensional data [5 pts]

In this problem, we will work through K-means for a single dimensional data.

(a) Consider the case where $K = 3$ and we have 4 data points $x_1 = 1, x_2 = 2, x_3 = 5, x_4 = 7$. What is the optimal clustering for this data? What is the corresponding value of the K-means objective?

Solution 2a:

We are given that $K = 3$ and there are four data points $x_1 = 1$, $x_2 = 2$, $x_3 = 5$, $x_4 = 7$. We are asked to find the optimal clustering for this data and the corresponding value of the objective function.

To solve this, we will consider the case where each data point forms a cluster with the centroid being the data point itself.

The objective function for clustering is typically defined as the sum of the squared distances from each point to the centroid of its cluster. For optimal clustering, we aim to minimize this objective function.

In this case, we assign centroids as follows:

- $c_1 = 1.5$ which is the average of x_1 and x_2, minimizing the distance within this cluster.
- $c_2 = 5$ as x_3 is a single-point cluster.
- $c_3 = 7$ as x_4 is also a single-point cluster.

The calculation of the objective function is as follows:

$$\begin{aligned}\text{Objective} &= (x_1 - c_1)^2 + (x_2 - c_1)^2 + (x_3 - c_2)^2 + (x_4 - c_3)^2 \\ &= (1 - 1.5)^2 + (2 - 1.5)^2 + (5 - 5)^2 + (7 - 7)^2 \\ &= 0.5^2 + 0.5^2 + 0 + 0 \\ &= 0.25 + 0.25 \\ &= 0.5\end{aligned}$$

Therefore, the optimal clustering for this data is to have clusters centered at $c_1 = 1.5$, $c_2 = 5$, and $c_3 = 7$, with the corresponding value of the objective function being 0.5.

(b) One might be tempted to think that Lloyd's algorithm is guaranteed to converge to the global minimum when dimension $d = 1$. Show that there exists a suboptimal cluster assignment (i.e., initialization) for the data in the above part that Lloyd's algorithm will not be able to improve (to get full credit, you need to show the assignment, show why it is suboptimal and explain why it will not be improved).

Solution 2b:

Consider an initialization of centroids $c_1 = 1$, $c_2 = 2$, and $c_3 = 6$ for the given data points $x_1 = 1$, $x_2 = 2$, $x_3 = 5$, and $x_4 = 7$. The initial objective function, which is the sum of squared distances from each point to its nearest centroid, is calculated as:

$$\text{Objective} = (x_1-c_1)^2+(x_2-c_2)^2+(x_3-c_3)^2+(x_4-c_3)^2 = 0^2+0^2+1^2+1^2 = 2$$

Upon running Lloyd's algorithm on this configuration, the data points would be assigned to the closest centroids, resulting in the following cluster assignments:

- $z_1 = 1$ (assigned to c_1)

- $z_2 = 2$ (assigned to c_2)
- $z_3 = 3$ (assigned to c_2)
- $z_4 = 3$ (assigned to c_3)

Given that x_3 and x_4 are closer to c_3 than c_2, Lloyd's algorithm will not reassign these points to a different centroid, thus resulting in no improvement in the objective function. This assignment is suboptimal because the objective function value could be lower if x_3 and x_4 were assigned to their own cluster or a cluster with a centroid closer to them. However, due to the poor initialization and the greedy nature of Lloyd's algorithm, it converges to a local minimum and fails to find the global minimum.

The reason why this clustering is suboptimal is that the points x_3 and x_4 could be better clustered together, reducing the overall sum of squared distances. A better clustering would be $c_1' = 1.5$, $c_2' = 5$, $c_3' = 7$ with an objective of 0.5, as shown in part (a). This demonstrates that Lloyd's algorithm is sensitive to initialization and does not always converge to the global optimum in the presence of multiple local minima.

3 Gaussian Mixture Models [8 pts]

We would like to cluster data $\{x_1, \ldots, x_N\}, x_n \in \mathbb{R}^d$ using a Gaussian Mixture Model (GMM) with K mixture components. To do this, we need to estimate the parameters $\boldsymbol{\theta}$ of the GMM, i.e., we need to set the values $\boldsymbol{\theta} = \{\omega_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$ where ω_k is the mixture weight associated with mixture component k, and $\boldsymbol{\mu}_k$ and Σ_k denote the mean and the covariance matrix of the Gaussian distribution associated with mixture component k.

If we knew which cluster each sample x_n belongs to (i.e., we have the complete data), we showed in the lecture on Clustering that the log likelihood l is what we have below and we can compute the maximum likelihood estimate (MLE) of all the parameters.

$$\begin{aligned} l(\boldsymbol{\theta}) &= \sum_n \log p\left(\boldsymbol{x}_n, z_n\right) \\ &= \sum_k \sum_n \gamma_{nk} \log \omega_k + \sum_k \left\{ \sum_n \gamma_{nk} \log N\left(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right) \right\} \end{aligned} \tag{1}$$

Since we do not have complete data, we use the EM algorithm. The EM algorithm works by iterating between setting each γ_{nk} to the posterior probability $p\left(z_n = k \mid \boldsymbol{x}_n\right)$ (step 1 on slide 26 of the lecture on Clustering) and then using γ_{nk} to find the value of $\boldsymbol{\theta}$ that maximizes l (step 2 on slide 26). We will

now derive updates for one of the parameters, i.e., $\boldsymbol{\mu}_j$ (the mean parameter associated with mixture component j).

(a) To maximize l, compute $\nabla \boldsymbol{\mu}_j l(\boldsymbol{\theta})$: the gradient of $l(\boldsymbol{\theta})$ with respect to $\boldsymbol{\mu}_j$.

Solution 3a:

We are tasked with maximizing the log-likelihood function $l(\theta)$ for a Gaussian distribution. To do this, we compute the gradient with respect to the mean vector μ_j.

Given the Gaussian (or Normal) distribution:

$$N(x \mid \mu_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_k|}} \exp\left(-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)\right)$$

The log-likelihood function is defined as:

$$l(\theta) = \text{Const} + \sum_k \left\{ \sum_n \gamma_{nk} \log N(x_n \mid \mu_k, \Sigma_k) \right\}$$

This can be simplified to:

$$l(\theta) = \sum_k \left\{ \sum_n \gamma_{nk} \left[-\frac{1}{2}(x_n-\mu_k)^T \Sigma_k^{-1}(x_n-\mu_k) \right] \right\}$$

Now, to find the gradient of the log-likelihood with respect to μ_j, we differentiate:

$$\nabla_{\mu_j} l(\theta) = \sum_n \gamma_{nj} \Sigma_j^{-1}(x_n - \mu_j)$$

We use the identity $\frac{\partial}{\partial x}(x^T A x) = (A + A^T)x$, where A is a matrix, to perform this differentiation. Applying this to our log-likelihood function, we get the gradient:

$$\nabla_{\mu_j} l(\theta) = \sum_n \gamma_{nj} \Sigma_j^{-1}(x_n - \mu_j)$$

Each term $\gamma_{nj}\Sigma_j^{-1}(x_n - \mu_j)$ represents the contribution of the nth data point to the gradient, weighted by the responsibility γ_{nj} that the kth Gaussian takes for that data point.

By setting this gradient to zero, we can solve for the mean μ_j that maximizes the log-likelihood function.

(b) Set the gradient to zero and solve for $\boldsymbol{\mu}_j$ to show that $\boldsymbol{\mu}_j = \frac{1}{\sum_n \gamma_{nj}} \sum_n \gamma_{nj} \boldsymbol{x}_n$.

Solution 3b:

To find the mean vector μ_j that maximizes the log-likelihood function, we set the gradient to zero:

$$\sum_n \gamma_{nj} \Sigma_j^{-1} (x_n - \mu_j) = 0$$

This equation implies that the sum of the weighted differences between each data point x_n and the mean μ_j must be zero. Since μ_j is not sample specific, it can be taken out of the sum:

$$\sum_n \gamma_{nj} \Sigma_j^{-1} x_n = \mu_j \sum_n \gamma_{nj} \Sigma_j^{-1}$$

Multiplying both sides by Σ_j to get rid of Σ_j^{-1}, we have:

$$\Sigma_j \left(\sum_n \gamma_{nj} \Sigma_j^{-1} x_n \right) = \Sigma_j \left(\mu_j \sum_n \gamma_{nj} \Sigma_j^{-1} \right)$$

The covariance matrix Σ_j can be distributed to each term on both sides of the equation and cancels with Σ_j^{-1}:

$$\sum_n \gamma_{nj} x_n = \mu_j \sum_n \gamma_{nj}$$

Rearranging terms to solve for μ_j, we find the desired solution:

$$\mu_j = \frac{1}{\sum_n \gamma_{nj}} \sum_n \gamma_{nj} x_n$$

This result indicates that the new mean μ_j is the weighted average of all data points, with weights given by γ_{nj}, which represent the responsibility that the Gaussian component j takes for the data point x_n.

(c) Suppose that we are fitting a GMM to data using $K = 2$ components. We have $N = 5$ samples in our training data with $x_n, n \in \{1, \ldots, N\}$ equal to: $\{5, 15, 25, 30, 40\}$.

We use the EM algorithm to find the maximum likeihood estimates for the model parameters, which are the mixing weights for the two components, ω_1 and ω_2, and the means for the two components, μ_1 and μ_2. The standard deviations for the two components are fixed at 1 .

γ_1	γ_2
0.2	0.8
0.2	0.8
0.8	0.2
0.9	0.1
0.9	0.1

Table 1: Entry in row n and column k of the table corresponds to γ_{nk}

Suppose that at the end of step 1 of iteration 5 in the EM algorithm, the soft assignment γ_{nk} for the five data items are as shown in Table 1.

What are updated values for the parameters $\omega_1, \omega_2, \mu_1$, and μ_2 at the end of step 2 of the EM algorithm?

Solution 3c:

The one-dimensional GMM is defined as follows:

$$p(x) = \sum_{i=1}^{K} \omega_i N(x|\mu_i, \sigma_i) \tag{10}$$

where $N(x|\mu_i, \sigma_i)$ is the normal distribution:

$$N(x|\mu_i, \sigma_i) = \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right) \tag{11}$$

and the mixture weights ω_i satisfy:

$$\sum_{i=1}^{K} \omega_i = 1 \tag{12}$$

Now we update ω_1, ω_2 in the M-Step, The updated weights $\hat{\omega}_k$ are computed as:

$$\hat{\omega}_k = \frac{1}{N} \sum_{i=1}^{N} \gamma_{ik} \tag{13}$$

Given the responsibilities γ_{ik}, we update the weights:

$$\omega_1 = \frac{0.2 + 0.2 + 0.8 + 0.9 + 0.9}{5} = \frac{3}{5},$$
$$\omega_2 = \frac{0.8 + 0.8 + 0.2 + 0.1 + 0.1}{5} = \frac{2}{5}.$$

Next, we Update μ_1, μ_2 in the M-Step. The means μ_k are updated as:

$$\mu_k = \frac{\sum_{i=1}^{N} \gamma_{ik} x_i}{\sum_{i=1}^{N} \gamma_{ik}} \tag{14}$$

Finally, we apply this to update the means:

$$\mu_1 = \frac{0.2 \times 5 + 0.2 \times 15 + 0.8 \times 25 + 0.9 \times 30 + 0.9 \times 40}{0.2 + 0.2 + 0.8 + 0.9 + 0.9} = \frac{87}{3} = 29,$$
$$\mu_2 = \frac{0.8 \times 5 + 0.8 \times 15 + 0.2 \times 25 + 0.1 \times 30 + 0.1 \times 40}{0.8 + 0.8 + 0.2 + 0.1 + 0.1} = \frac{28}{2} = 14.$$