THE CHINESE UNIVERSITY OF HONGKONG, SHENZHEN
COMPUTER AND INFORMATION ENGINEERING

Homework #2

Student name: *Fanyi Meng (223015127)*

Course: *Advanced Machine Learning (AIR 6002)* – Professor: *Prof Tongxin Li*
Due date: *March 31st, 2024*

Question 1: Adaboost

(*a*) Let $h_t : \mathbb{R}^m \to \{-1, 1\}$ be the weak classifier obtained at step t, and let α_t be its weight. Recall that the final classifier is

$$H(x) = \text{sign}(f(x)) = \text{sign}\left(\sum_{i=1}^{T} \alpha_t h_t(x)\right).$$

Suppose $\{(x_1, y_1), \ldots, (x_N, y_N)\} \subset \mathbb{R}^m \times \{-1, 1\}$ is our training dataset. Show that the training set error of the final classifier can be bounded from above if an exponential loss function is used:

$$E = \frac{1}{N}\sum_{i=1}^{N} \exp\left(-y_i f\left(x_i\right)\right) \geq \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left(H\left(x_i\right) \neq y_i\right)$$

where $\mathbb{1}$ is the indicator function.

(*b*) Find $D_{T+1}(i)$ in terms of Z_t, α_t, x_i, y_i, and the classifier h_t, where T is the last timestep and $t \in \{1, \ldots, T\}$. Recall that Z_t is the normalization factor for distribution D_{t+1}:

$$Z_t = \sum_{i=1}^{N} D_t(i) \exp\left(-\alpha_t y_i h_t\left(x_i\right)\right).$$

(*c*) Show that $E = \sum_{i=1}^{N} \frac{1}{N} e^{\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i)}$.

(*d*) Show that

$$E = \prod_{t=1}^{T} Z_t.$$

(*e*) Show that the normalizer Z_t can be written as

$$Z_t = (1 - \epsilon_t) \exp\left(-\alpha_t\right) + \epsilon_t \exp\left(\alpha_t\right)$$

where ϵ_t is the training set error of weak classifier h_t for the weighted dataset:

$$\epsilon_t = \sum_{i=1}^{N} D_t(i) \mathbb{1}\left(h_t\left(x_i\right) \neq y_i\right).$$

(*f*) We derived all of this because it is hard to directly minimize the training set error, but we can greedily minimize the upper bound E on this error. Show that choosing α_t greedily to minimize Z_t at each iteration leads to the choices in AdaBoost:

$$\alpha_t^* = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right).$$

(*g*) **AIR6002** Implement the `GradientBoosting.fit()` and `AdaBoost.fit()` methods in the notebook "Q1_AdaBoost.ipynb" provided for you.

Some important notes and guidelines:

- For both methods, make sure to work with the class attributes provided to you. Namely, after `GradientBoosting.fit()` is called, `self.clfs` should be appropriately filled with the `self.n_clfs` trained weak hypotheses. Similarly, after `AdaBoost.fit()` is called, `self.clfs` and `self.coefs` should be appropriately filled with the `self.n_clfs` trained weak hypotheses and their coefficients, respectively.
- `AdaBoost.fit()` should additionally return an (N, T) shaped numpy array D such that `D[:,t]` contains D_{t+1} for each $t \in \{0, \ldots,$ `self.n_clfs`$\}$.
- For the `AdaBoost.fit()` method, use the $0/1$ loss instead of the exponential loss.

(*h*) **AIR6002** Plot the loss curves for gradient boosting and for AdaBoost. Describe and explain the behaviour of the two loss curves you plot. You should consider two kinds of behaviours: the smoothness of the curves and the final values that the curves approach.

(*i*) **AIR6002** Compare the final loss values of the two models. Which performed better on the classification dataset?

(*j*) **AIR6002**For AdaBoost, where are the dataset weights the largest, and where are they the smallest?

Hint: *Watch how the dataset weights change across time in the animation.*

Answer [1]

(*a*) Since $f(x_i) = \text{sign}(f(x_i)) \, |f(x_i)| = H(x_i) \, |f(x_i)|$

And:

- If $H(x_i) \neq y_i$ then $\mathbb{1}(H(x_i) \neq y_i) = 1 \leq e^{|f(x_i)|}$.
- If $H(x_i) = y_i$ then $\mathbb{1}(H(x_i) \neq y_i) = 0 \leq e^{-|f(x_i)|}$.

So, the inequality holds for each term

$$\mathbb{1}(H(x_i) \neq y_i) \leq \exp(-y_i f(x_i))$$

[1] Some of the Answer 1 were referenced from `https://cse.buffalo.edu/~jcorso/t/CSE555/files/lecture_boosting.pdf`

and hence, the inequality is true.

(*b*)

$$\begin{aligned} D_{T+1}(i) &= \frac{1}{Z_T} D_T(i) \exp(-\alpha_T y_i h_T(x_i)) \\ &= \frac{1}{Z_T Z_{T-1}} D_{T-1}(i) \exp\left[-y_i(\alpha_t h_T(x_i) + \alpha_{t-1} h_{T-1}(x_i))\right] \\ &= \cdots \\ &= \frac{1}{Z_1 \cdots Z_T} D_1(i) \exp\left[-y_i(\alpha_T h_T(x_i) + \cdots + \alpha_1 h_1(x_i))\right] \\ &= \prod_1^T \frac{1}{Z_t} \exp\left(-y_i \sum_1^T \alpha_t h_t\left(x_i\right)\right). \end{aligned}$$

(*c*) $E = \frac{1}{N}\sum_{i=1}^N \exp\left(-y_i f\left(x_i\right)\right)$, so $E = \sum_{i=1}^N \frac{1}{N} e^{\sum_{t=1}^T -\alpha_t y_i h_t(x_i)}$.

(*d*) Following (b), we also have $\sum_i^N D_{T+1}(i) = 1$, so

$$\sum_{i=1}^n D_t(x_i) = \prod_1^T \frac{1}{Z_t} \sum_{i=1}^N \frac{1}{N} \exp[-y_i f(x_i)] = 1$$

implies:

$$\prod_{t=1}^T Z_t = \frac{1}{N} \sum_{i=1}^N \exp\left(-y_i f\left(x_i\right)\right) = E.$$

(*e*) The equation can be represented as :

$$Z_t = \sum_i^N D_t(x_i) \exp[-y_i \alpha_t h_t(x_i)](\mathbb{1}\left(h_t\left(x_i\right) = y_i\right) + \mathbb{1}\left(h_t\left(x_i\right) \neq y_i\right))$$

As $y_i h_t(x_i) = 1$ when $h_t\left(x_i\right) = y_i$ and $y_i h_t(x_i) = -1$ otherwise, so:

$$\begin{aligned} Z_t &= \sum_i^N D_t(i) \exp[-\alpha_t] \mathbb{1}\left(h_t\left(x_i\right) = y_i\right) + \sum_i^N D_t(i) \exp[\alpha_t] \mathbb{1}\left(h_t\left(x_i\right) \neq y_i\right) \\ &= (1 - \epsilon_t) \exp\left(-\alpha_t\right) + \epsilon_t \exp\left(\alpha_t\right) \end{aligned}$$

(*f*) By calculating the gradient of (e):, we get:

$$(\epsilon_t - 1) \exp\left(-\alpha_t\right) + \epsilon_t \exp\left(\alpha_t\right) = 0$$

which implies:

$$\exp\left(\alpha_t\right) = \sqrt{\frac{1 - \epsilon_t}{\epsilon_t}}$$

so

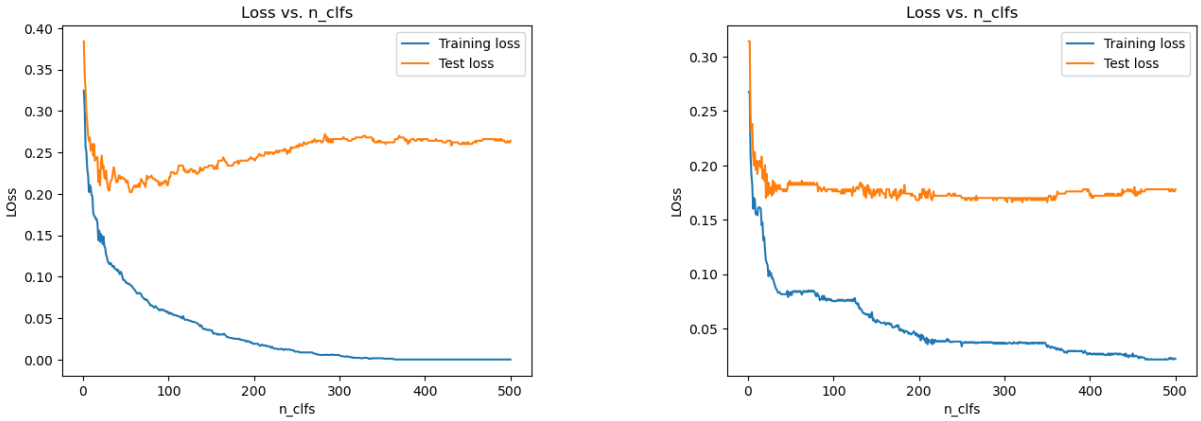
$$\alpha_t^* = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right).$$

(*g*) The code can be viewed in the code part.

(*h*) Gradient Boosting seems to be more smooth and has a good performance on training dataset, but something like overfitting probably occurs since the test loss grows as the n clfs grows. Adaboost is not smooth but has a better performance on test dataset.

(*i*) Adaboost.

(*j*) For the data which is predicted wrong in the origin model, the weights are larger. And for the data which has been predicted to be true, the weights are smaller.



(a) Gradient boosting Loss (b) Adaboost Loss

Figure 1: Loss curves for gradient boosting and for AdaBoost

Question 2: Recommendation Systems

(*a*) What are the differences between collaborative filtering and content-based methods?

(*b*) Suppose we have m items and n users. Let Ω be the set of indices of observed ratings given by the users on the items. Provide the objective function of content-based recommendation for users, where the model class is a neural network with two hidden layers. You need to define the notations clearly. In addition, explain how to make recommendations for a new user using your model.

Answer

(*a*) Collaborative filtering relies on the user-itsm interaction. It makes recommendations based on patterns of users' behavior with items, without requiring knowledge about the items themselves. Content-Based Methods focus on the properties of the items. Recommendations are made by analyzing the content of the items and the profile of the user's preferences. So Collaborative Filtering can provide more personalized recommendations and Content-Based Methods recommend items that are similar to what the user has liked before.

(*b*) Then the objective function can be represented as:

Notations	Description
$W^{(1)}$	Weights of the first hidden layer
$b^{(1)}$	Biases of the first hidden layer
$W^{(2)}$	Weights of the second hidden layer
$b^{(2)}$	Biases of the second hidden layer
$W^{(3)}$	Weights of the output layer
$b^{(3)}$	Bias of the output layer
r_{ui}	The element of the rating matrix
f_i	The activation function of corresponding layer(e.g. ReLu)

Table 1: notations

$$J(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, W^{(3)}, b^{(3)}) = \frac{1}{|\Omega|} \sum_{(u,i)\in\Omega} (r_{ui} - \hat{r}_{ui})^2 \tag{1}$$

where the predicted rating $\hat{r}_{ui}$ is computed as:

$$\hat{r}_{ui} = W^{(3)} f_2 \left(W^{(2)} f_1 \left(W^{(1)} Z_i + b^{(1)} \right) + b^{(2)} \right) + b^{(3)} \tag{2}$$

To make recommendations for a new user, we need to collect some user's preferences and use the model to get $r_u i$. By sorting the score, we will recommend the top items.

Question 3:Spectral Clustering

Prove $\frac{\mathbf{u}^\top \mathbf{L}\mathbf{u}}{\mathbf{u}^\top \mathbf{D}\mathbf{u}} = \text{Ncut}(A, B)$. See the definitions on page 23 of Lecture 05-I.

Answer

Since

$$L = D - W$$

and

$$D = \text{diag}(d_1, \ldots, d_N)$$

And

$$u_i = \begin{cases} \frac{1}{\sqrt{\text{vol}(A)}} & \text{if } i \in A \\ -\frac{1}{\sqrt{\text{vol}(B)}} & \text{if } i \in B \end{cases}$$

Then, we can express $\mathbf{u}^T L \mathbf{u}$ and $\mathbf{u}^T D \mathbf{u}$ as follows:

$$\mathbf{u}^T L \mathbf{u} = \frac{1}{2} \sum_{i,j} w_{ij} (u_i - u_j)^2 = \sum_{i\in A, j\in B} w_{ij} \left(\frac{1}{\sqrt{\text{vol}(A)}} + \frac{1}{\sqrt{\text{vol}(B)}} \right)^2$$

$$\mathbf{u}^T D \mathbf{u} = \sum_i d_i u_i^2 = \sum_{i\in A} \frac{d_i}{\text{vol}(A)^2} + \sum_{j\in B} \frac{d_j}{\text{vol}(B)^2} = \frac{1}{\text{vol}(A)} + \frac{1}{\text{vol}(B)}$$

Because the sums of the degrees for vertices in each set A and B equal the volume of those sets, respectively.

Therefore, the normalized cut (Ncut) for the partition of the graph into sets A and B is given by:

$$\text{Ncut}(A, B) = \frac{\mathbf{u}^T L \mathbf{u}}{\mathbf{u}^T D \mathbf{u}} = \sum_{i \in A, j \in B} w_{ij} \left(\frac{1}{\text{vol}(A)} + \frac{1}{\text{vol}(B)} \right)$$

This concludes the proof that the expression $\frac{\mathbf{u}^T L \mathbf{u}}{\mathbf{u}^T D \mathbf{u}}$ is equal to the normalized cut value.

Question 4:Semi-supervised Learning

(*a*) Suppose $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{y} \in \mathbb{R}^k$ and the training data are $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^l \bigcup \{\mathbf{x}_i\}_{i=l+1}^n$. Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be an affinity matrix constructed from the training data.

i) Show the objective function of linear regression with square loss and graph regularization. For simplicity, do not consider the bias term.

ii) Compute the gradient of the objective function with respect to the parameters.

iii) Is there a closed-form solution? If yes, find it.

(*b*) In the label propagation algorithm, why do we need to use $\mathbf{D}^{-1}\mathbf{W}$ instead of $\mathbf{W}$? Why do we set $\hat{\mathbf{Y}}_l^{(t+1)} \leftarrow \mathbf{Y}_l$? If there are more than 2 classes, how do we set $\mathbf{Y}^{(0)}$?

Answer

(*a*) i)

$$J(w) = \frac{1}{2} \sum_{i=1}^{l} (y_i - w^T x_i)^2 + \frac{\lambda}{2} w^T (D - A) w$$

(D is the degree matrix of the graph)

ii) The gradient is:

$$\nabla_w J(w) = \sum_{i=1}^{l} (w^T x_i - y_i) x_i + \lambda (D - A) w$$

iii) The closed form is:

$$w = (X^T X + \lambda (D - A))^{-1} X^T Y$$

(*b*) i) Since the degree matrix D contains the information of the degree, so using $\mathbf{D}^{-1}\mathbf{W}$ will make each data sample have the same impact on its neighbor.

ii) Since we want to keep the same label for the labeled data.

iii) One-hot Encoding is one possible method.

Question 5: Graph Neural Networks

(*a*) In GNN, given $\hat{\mathbf{A}}$, how to compute $\hat{\mathbf{A}}^c := \underbrace{\hat{\mathbf{A}}\hat{\mathbf{A}}\cdots\hat{\mathbf{A}}}_{c}$ efficiently when c is large?

(*b*) Show the loss functions of node classification, graph classification, and link prediction. Explain your notations.

(*c*) Provide two examples of algorithms for each learning types or methods:

- parametric method
- nonparametric method
- inductive learning
- transductive learning
- semi-supervised learning

Answer

(*a*) Since $\hat{\mathbf{A}}$ is always invertible, we can calculate the EVD of the matrix as $\hat{\mathbf{A}} = \mathbf{U}\Lambda\mathbf{U}^{-1}$, so $\hat{\mathbf{A}}^c = \mathbf{U}\Lambda^C\mathbf{U}^{-1}$

(*b*) Here is the notation.[2]

Notation	Description
V_{train}	Set of nodes used for training.
z_u	Embedding of node u.
y_u	Class label of node u .
T	Set of graphs used for training.
z_{G_i}	Graph-level embedding of graph G_i.
y_{G_i}	Target value for training graph G_i
E	Set of edges, representing all possible node pairs.
y_{uv}	Actual label indicating whether a link exists between node pair (u, v).
$\hat{y}_{uv}$	Predicted probability that a link exists between node pair (u, v).

Loss function for node classification:

$$L = -\sum_{u \in V_{\text{train}}} \log(\text{softmax}(z_u)[y_u])$$

Loss function for graph classification:

$$L = -\sum_{G_i \in T} \log(\text{softmax}(z_{G_i})[y_{G_i}])$$

Loss function for link prediction:

$$L = -\sum_{(u,v) \in E} y_{uv} \log(\hat{y}_{uv}) + (1 - y_{uv}) \log(1 - \hat{y}_{uv})$$

[2]Some of this answer were referenced fromhttps://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book-Chapter_6-GNNs_in_Practice.pdf

(*c*) **Parametric method**:Linear Regression, Logistic Regression

Nonparametric method :K-Nearest Neighbors,Decision Trees

Inductive Learning:Support Vector Machines,Random Forest

Transductive Learning:Label Propagation,Label Spreading

Semi-supervised Learning:Autoencoders,Semi-Supervised SVM

Question 6: Nonlinear Dimensionality Reduction

(*a*) Show the algorithm of multidimensional scaling here.

(*b*) Present a method for out-of-sample extension of t-SNE. In other words, reduce the dimension of new samples using the training result of t-SNE.

(*c*) Given a dataset, suppose most of the samples are normal or good data, and there are a few outliers. Design a method or strategy to detect these outliers bassed on the methods learned in Lecture 07.

Answer

(*a*)
- Calculate the distance between the datas.(e.g.Euclidean distance)
- Build Gram matrix B , where

$$b_{ij} = -\frac{1}{2}(d_{ij}^2 - \bar{d}_{i\cdot}^2 - \bar{d}_{\cdot j}^2 + \bar{d}_{\cdot\cdot}^2)$$

- Calculate EVD of B,so

$$B = V\Lambda V^T$$

- Choose the eigen vector for the top k eigenvectors, then get the low dimension

$$X = V_k\sqrt{\Lambda_k}$$

(*b*) Firstly, we can use t-SNE to map it into low dimension. And then we find its neighbors and calculate the center of there neighbors as the new sample.

(*c*) We can use Non-Linear Dimensionality Reduction to map it into low dimension and find whetehr the data has enough neighbors. Outliers may has no neighbor.

Question 7: Generative Models

(*a*) Derive the evidence lower bound (ELBO) for the VAE model.

(*b*) Derive the objective functions for the generator and discriminator in a GAN, respectively.

(*c*) Briefly explain the relationship between diffusion models and Langevin dynamics.

Answer

(*a*)

$$\text{ELBO} = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z};\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$$

Where:

- $p\theta(x|z)$ is the likelihood of observing data x given latent variables z.

- $q\phi(z|x)$ is the variational distribution (encoder) that approximates the true posterior distribution of latent variables given data.

- $p(z)$ is the prior distribution of latent variables.

- $\text{KL}(q\phi(z|x)||p\theta(z))$ is the Kullback-Leibler (KL) divergence between the variational distribution and the prior.

(*b*) Generator's objective:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))]$$

Discriminator's objective:

$$\min_D \max_G V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))]$$

Where:

- $p_{\text{data}}(x)$ is the true data distribution.

- $p(z)$ is the prior distribution of noise variables z.

- $G(z)$ is the generated data sample.

- $D(x)$ represents the discriminator's output indicating the probability that x is a real data sample.

(*c*) - **Diffusion models** describe the evolution of a stochastic process over time, where the process starts from an initial distribution and evolves according to a diffusion process.

- **Langevin dynamics**, is a type of stochastic differential equation that describes the motion of particles subjected to random forces. In the context of probabilistic modeling, Langevin dynamics can be used for sampling from probability distributions by simulating the motion of particles according to the gradients of the log-probability density.

Question 8: Graph Convolutional Network

(*a*) **AIR6002** Implement the `GCNLayer.forward()` in the notebook "Q8_GCN.ipynb" provided for you.

(*b*) **AIR6002** Define the adjacency matrix of the graph in the notebook with self-connections.

(*c*) **AIR6002** Apply the GCN layer defined in (1) on the graph in (2), and write down the output features for all nodes.

Answer

All these parts are shown in code part.