DDA4210 Spring 2024 - Assignment 2

Yohandi

March 31, 2024

1. AdaBoost

In this problem, you will show that the choice of the α_t parameter in the AdaBoost algorithm corresponds to greedily minimizing an exponential upper bound on the loss term at each iteration.

(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

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

Suppose $\{(x_1, y_1), \ldots, (x_N, y_N)\} \subseteq \mathbb{R}^m \times \{-1, 1\}$ 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:

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

where $\mathbb{I}$ 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}:

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

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

(d) Show that $E = \prod_{t=1}^{T} Z_t$.

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

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

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

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

(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:

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

(a) We will show by case analysis:

- If $H(x_i) = y_i$, then $y_i f(x_i) > 0$, implying $0 \leq \exp(-y_i f(x_i)) < 1$. In this case $\mathbb{I}(H(x_i) \neq y_i)$ returns 0, and we observe that $\exp(-y_i f(x_i)) \geq \mathbb{I}(H(x_i) \neq y_i)$.
- If $H(x_i) \neq y_i$, then $y_i f(x_i) < 0$, implying $\exp(-y_i f(x_i)) \geq 1$. In this case $\mathbb{I}(H(x_i) \neq y_i)$ returns 1, and we still observe that $\exp(-y_i f(x_i)) \geq \mathbb{I}(H(x_i) \neq y_i)$.

By the above case analysis, $\exp(-y_i f(x_i)) \geq \mathbb{I}(H(x_i) \neq y_i), \forall i \in \{1, \ldots, N\}$, further implying that:

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

(b) At a timestep $T+1$, the update rule of AdaBoost for the distribution $D_{T+1}(i)$ is given by:

$$\begin{aligned}
D_{T+1}(i) &= D_T(i)\frac{\exp(-a_T y_i h_T(x_i))}{Z_T}, \forall i \in \{1,\dots,N\} \\
&= D_{T-1}(i)\frac{\exp(-a_{T-1} y_i h_{T-1}(x_i))}{Z_{T-1}}\frac{\exp(-a_T y_i h_T(x_i))}{Z_T}, \forall i \in \{1,\dots,N\} \\
&\vdots \\
&= D_0(i)\left(\prod_{t=1}^{T}\frac{1}{Z_t}\right)\left(\prod_{t=1}^{T}\exp(-a_t y_i h_t(x_i))\right) \\
&= \underbrace{\frac{1}{N}}_{\text{uniform}}\left(\prod_{t=1}^{T}\frac{1}{Z_t}\right)\exp(-\sum_{t=1}^{T}a_t y_i h_t(x_i)) \text{ (as } D_0 \text{ represents the initial distribution)}
\end{aligned}$$

(c) The composite hypothesis at iteration T for a data point x_i is given by:

$$f(x_i) = \sum_{t=1}^{T}\alpha_t h_t(x_i)$$

Substitute $f(x_i)$ to E, we obtain:

$$\begin{aligned}
E &= \frac{1}{N}\sum_{i=1}^{N}\exp(-y_i f(x_i)) \\
&= \frac{1}{N}\sum_{i=1}^{N}\exp(-y_i\sum_{t=1}^{T}\alpha_t h_t(x_i)) \\
&= \sum_{i=1}^{N}\frac{1}{N}\exp(-\sum_{t=1}^{T}\alpha_t y_i h_t(x_i))
\end{aligned}$$

(d) We will utilize the results in part (b) and (c).

$$\begin{aligned}
E &= \sum_{i=1}^{N}\frac{1}{N}\exp(-\sum_{t=1}^{T}\alpha_t y_i h_t(x_i)) \\
&= \sum_{i=1}^{N}D_{T+1}(i)\underbrace{\left(\prod_{t=1}^{T}Z_t\right)}_{\text{constant}} \\
&= \left(\prod_{t=1}^{T}Z_t\right)\underbrace{\sum_{i=1}^{N}D_{T+1}(i)}_{\text{sum to 1}} \\
&= \prod_{t=1}^{T}Z_t
\end{aligned}$$

(e) Z_t can be written as:

$$\begin{aligned}
Z_t &= \sum_{i=1}^{N} D_t(i)\exp(-\alpha_t y_i h_t(x_i)) \\
&= \sum_{i:h_t(x_i)=y_i} D_t(i)\exp(-\alpha_t \underbrace{y_i h_t(x_i)}_{\text{equals to 1}}) + \sum_{i:h_t(x_i)\neq y_i} D_t(i)\exp(-\alpha_t \underbrace{y_i h_t(x_i)}_{\text{equals to -1}}) \\
&= \exp(-\alpha_t)\left(\sum_{i:h_t(x_i)=y_i} D_t(i)\right) + \exp(\alpha_t)\left(\sum_{i:h_t(x_i)\neq y_i} D_t(i)\right) \\
&= \exp(-\alpha_t)\left(\underbrace{\sum_{i=1}^{N} D_t(i)}_{\text{sum to 1}} - \underbrace{\sum_{i:h_t(x_i)\neq y_i} D_t(i)}_{\epsilon_t\text{ definition}}\right) + \exp(\alpha_t)\left(\underbrace{\sum_{i:h_t(x_i)\neq y_i} D_t(i)}_{\epsilon_t\text{ definition}}\right) \\
&= (1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t\exp(\alpha_t)
\end{aligned}$$

(f) Derivative of Z_t towards α_t is given by:

$$\begin{aligned}
\frac{\partial Z_t}{\partial \alpha_t} &= (1-\epsilon_t)\frac{\partial \exp(-\alpha_t)}{\partial \alpha_t} + \epsilon_t\frac{\partial \exp(\alpha_t)}{\partial \alpha_t} \\
&= -(1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t\exp(\alpha_t)
\end{aligned}$$

For an optimal α_t^*, $\frac{\partial Z_t}{\partial \alpha_t}(\alpha_t^*) = 0$. Then, α_t^* can be obtained as follows:

$$\begin{aligned}
(1-\epsilon_t)\exp(-\alpha_t^*) &= \epsilon_t\exp(\alpha_t^*) \\
\Rightarrow \exp(2\alpha_t^*) &= \frac{1-\epsilon_t}{\epsilon_t} \\
\Rightarrow 2\alpha_t^* &= \ln(\frac{1-\epsilon_t}{\epsilon_t}) \\
\Rightarrow \alpha_t^* &= \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})
\end{aligned}$$

Moreover, as

$$\begin{aligned}
\frac{\partial^2 Z_t}{(\partial \alpha_t)^2} &= \underbrace{(1-\epsilon_t)}_{\in[0,1]}\underbrace{\exp(-\alpha_t)}_{>0} + \underbrace{\epsilon_t}_{\in[0,1]}\underbrace{\exp(\alpha_t)}_{>0} \\
&> 0
\end{aligned}$$

, Z_t is strictly convex, further confirming that $\alpha_t^* = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$ is optimal.

2. Recommendation Systems

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

(2) 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.

(1) Differences between collaborative filtering method and content-based method:

- Collaborative filtering methods recommend items by identifying patterns of user interactions. In contrast, content-based methods recommend items by analyzing the features of the items themselves.
- The underlying assumption in collaborative filtering is that users with similar past preferences will likely have similar future preferences. Content-based methods assume that a user will continue to be interested in items with features similar to those they have liked before.
- Collaborative filtering excels in delivering a broad spectrum of recommendations from variations of user preferences. Content-based methods, however, have a tendency to suggest items that are closely aligned with a user's past preferences, which may limit diversity.
- The collaborative filtering method requires more data on handling new items and users, which is a more difficult task compared to the content-based method that works without using the rating data.

(2) Below are the notations that will be used to describe the objective function:

- m as the number of items.
- d_u as the number of user u's features.
- $z_u \in \mathbb{R}^{d_u}$ as the feature vector of user u.
- r_{ui} as the ground truth rate of item i given by user u.
- $f : \mathbb{R}^{d_u} \to \mathbb{R}^m$ as neural network with two hidden layers.
- ℓ as the loss function.

Then, recommendation for users is:

$$f^* = \arg\min_{f \in \mathcal{F}} \mathcal{L}(f)$$

, where $\mathcal{F}$ is the model class of neural networks with two hidden layers, and

$$\mathcal{L}(f) = \sum_{(u,i)\in\Omega} \ell(r_{ui}, f_i(z_u))$$

is the objective function.
With the assumption that all the layers are fully connected, and the following suppositions:

- ϕ_1, ϕ_2, g are activation functions for those layers.
- $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3$ are weight matrices and $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$ are bias vectors.

$$\begin{tikzcd} \boxed{z_u} \arrow[r, "\phi_1(\cdot)"] & \boxed{h_1} \arrow[r, "\phi_2(\cdot)"] & \boxed{h_2} \arrow[r, "g(\cdot)"] & \boxed{f} \arrow[r, "\ell(v,\cdot)"] & \mathcal{L} \end{tikzcd}$$

Then, the computational graph is as follows:

$$f(z_u) = g(\mathbf{W}_3\phi_2(\mathbf{W}_2\phi_1(\mathbf{W}_1 z_u + \mathbf{b}_1) + \mathbf{b}_2) + \mathbf{b}_3)$$

The same model can be applied to recommend new users by selecting the top items with the highest predicted ratings.

3. Spectral Clustering

Prove that $\frac{\mathbf{u}^T\mathbf{L}\mathbf{u}}{\mathbf{u}^T\mathbf{D}\mathbf{u}} = \text{Ncut}(A, B)$.

Based on the given definitions:

- $\mathbf{u} = [u_1, \dots, u_n]^T$ is a vector, where $u_i = \begin{cases} \frac{1}{\text{Vol}(A)} & \text{, if } i \in A \\ -\frac{1}{\text{Vol}(B)} & \text{, if } i \in B \end{cases}$
- $\mathbf{D} = \texttt{diag}(d_1, \dots, d_n)$ is a diagonal matrix, where $d_i = \sum_{j=1}^n w_{ij}$ and $\mathbf{W}$ is the similarity matrix of the given graph.
- $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is a Laplacian matrix.

$\mathbf{u}^T\mathbf{L}\mathbf{u}$ can be derived as follows:

$$\begin{aligned}
\mathbf{u}^T\mathbf{L}\mathbf{u} &= \mathbf{u}^T(\mathbf{D} - \mathbf{W})\mathbf{u} \\
&= \mathbf{u}^T\mathbf{D}\mathbf{u} - \mathbf{u}^T\mathbf{W}\mathbf{u} \\
&= \sum_{i=1}^n d_i u_i^2 - \sum_{i=1}^n \sum_{j=1}^n w_{ij} u_i u_j \\
&= \sum_{i=1}^n \sum_{j=1}^n w_{ij} u_i^2 - \sum_{i=1}^n \sum_{j=1}^n w_{ij} u_i u_j \\
&= \sum_{i=1}^n \sum_{j=1}^n w_{ij}(u_i^2 - u_i u_j) \\
&= \frac{1}{2} \cdot 2 \cdot \sum_{i=1}^n \sum_{j=1}^n w_{ij}(u_i^2 - u_i u_j) \\
&= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij}(u_i^2 - u_i u_j + u_j^2 - u_i u_j) \\
&= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij}(u_i - u_j)^2 \\
&= \sum_{i \in A} \sum_{j \in B} w_{ij}(u_i - u_j)^2 \text{ (due to double counting and } \mathbf{W} \text{ being symmetric)} \\
&= \sum_{i \in A} \sum_{j \in B} w_{ij} \left(\frac{1}{\text{Vol}(A)} + \frac{1}{\text{Vol}(B)} \right)^2
\end{aligned}$$

$\mathbf{u}^T\mathbf{D}\mathbf{u}$ can be derived as follows:

$$\begin{aligned}\mathbf{u}^T\mathbf{D}\mathbf{u} &= \sum_{i=1}^{n} d_i u_i^2 \\ &= \sum_{i\in A}^{n} \frac{d_i}{\text{Vol}(A)^2} + \sum_{j\in B}^{n} \frac{d_j}{\text{Vol}(B)^2} \\ &= \frac{1}{\text{Vol}(A)} + \frac{1}{\text{Vol}(B)} \text{ (by definition of Vol)}\end{aligned}$$

By the above derivations, it can be concluded that:

$$\begin{aligned}\frac{\mathbf{u}^T\mathbf{L}\mathbf{u}}{\mathbf{u}^T\mathbf{D}\mathbf{u}} &= \frac{\sum_{i\in A}\sum_{j\in B} w_{ij}\left(\frac{1}{\text{Vol}(A)} + \frac{1}{\text{Vol}(B)}\right)^2}{\frac{1}{\text{Vol}(A)} + \frac{1}{\text{Vol}(B)}} \\ &= \sum_{i\in A}\sum_{j\in B} w_{ij}\left(\frac{1}{\text{Vol}(A)} + \frac{1}{\text{Vol}(B)}\right) \\ &= \text{Ncut}(A, B)\end{aligned}$$

4. Semi-supervised Learning

(1) Suppose $x \in \mathbb{R}^d$ and $y \in \mathbb{R}^k$ and the training data are $\{(x_i, y_i)\}_{i=1}^{l} \cup \{x_j\}_{j=l+1}^{n}$. Let $A \in \mathbb{R}^{n\times n}$ 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.

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

i) Below are the suppositions that will be used:

- $\mathbf{X} = [\mathbf{X}_l, \mathbf{X}_u] = [\underbrace{x_1, \ldots x_l}_{\mathbf{X}_l}, \underbrace{x_{l+1}, \ldots x_n}_{\mathbf{X}_u}]$
- $\mathbf{Y} = [y_1, \ldots, y_l]$
- $\mathbf{W} \in \mathbb{R}^{d\times k}$ as the weight matrix that represents the linear regression model.
- Prediction vectors $\mathbf{f}_i = x_i\mathbf{W} \in \mathbb{R}^k, \forall i$.
- $L = D - A$ as the Laplacian matrix, where $D = (\sum_{i=1}^{n} A_{i1}, \ldots, \sum_{i=1}^{n} A_{in})$.

The objective function for linear regression with square loss and graph regularization is:

$$\begin{aligned}
\mathcal{L}(\mathbf{W}) &= \sum_{i=1}^{l} \|y_i - \mathbf{f}_i\|_2^2 + \lambda \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}\|\mathbf{f}_i - \mathbf{f}_j\|_2^2 \\
&= \sum_{i=1}^{l} \|y_i - x_i\mathbf{W}\|_2^2 + \lambda \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}\left(\sum_{t=1}^{k} \mathbf{f}_i^{(t)2} - 2\mathbf{f}_i^{(t)}\mathbf{f}_j^{(t)} + \mathbf{f}_j^{(t)2}\right) \\
&= \sum_{i=1}^{l} \|y_i - x_i\mathbf{W}\|_2^2 + 2\lambda \sum_{t=1}^{k}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}\mathbf{f}_i^{(t)2} - 2\lambda \sum_{t=1}^{k}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}\mathbf{f}_i^{(t)}\mathbf{f}_j^{(t)} \\
&= \sum_{i=1}^{l} \|y_i - x_i\mathbf{W}\|_2^2 + 2\lambda \sum_{t=1}^{k}\sum_{i=1}^{n} \mathbf{f}_i^{(t)T} D\mathbf{f}_i^{(t)} - 2\lambda \sum_{t=1}^{k} \mathbf{f}^{(t)T} A\mathbf{f}_j^{(t)} \\
&= \text{Trace}((\mathbf{Y} - \mathbf{X}_l\mathbf{W})^T(\mathbf{Y} - \mathbf{X}_l\mathbf{W})) + \underbrace{2\lambda}_{\lambda'} \text{Trace}((\mathbf{XW})^T L(\mathbf{XW}))
\end{aligned}$$

ii) The gradient of the objective function can then be derived as:

$$\begin{aligned}
\nabla\mathcal{L}(\mathbf{W}) &= \nabla\left(\text{Trace}((\mathbf{Y} - \mathbf{X}_l\mathbf{W})^T(\mathbf{Y} - \mathbf{X}_l\mathbf{W})) + \lambda'\text{Trace}(\mathbf{W}^T(\mathbf{X}^T L\mathbf{X})\mathbf{W})\right) \\
&= \nabla\left(\text{Trace}(\mathbf{Y}^T\mathbf{Y} - \mathbf{Y}^T\mathbf{X}_l\mathbf{W} - \mathbf{W}^T\mathbf{X}_l^T\mathbf{Y} + \mathbf{W}^T\mathbf{X}_l^T\mathbf{X}_l\mathbf{W}) + \lambda'\text{Trace}(\mathbf{W}^T(\mathbf{X}^T L\mathbf{X})\mathbf{W})\right) \\
&= \nabla\left(-2\text{Trace}(\mathbf{Y}^T\mathbf{X}_l\mathbf{W}) + \text{Trace}(\mathbf{W}^T(\mathbf{X}_l^T\mathbf{X}_l)\mathbf{W}) + \lambda'\text{Trace}(\mathbf{W}^T(\mathbf{X}^T L\mathbf{X})\mathbf{W})\right) \\
&= -2\mathbf{X}_l^T\mathbf{Y} + 2\mathbf{X}_l^T\mathbf{X}_l\mathbf{W} + 2\lambda'\mathbf{X}^T L\mathbf{X}\mathbf{W}
\end{aligned}$$

iii) Yes. The closed form solution $\mathbf{W}^*$ is obtainable by solving $\nabla\mathcal{L}(\mathbf{W}^*) = 0$. $\mathbf{W}^*$ is obtained to be:

$$\mathbf{W}^* = (\mathbf{X}_l^T\mathbf{X}_l + \lambda'\mathbf{X}^T L\mathbf{X})^{-1}\mathbf{X}_l^T\mathbf{Y}$$

. Moreover, as

$$\nabla^2\mathcal{L}(\mathbf{W}) = \underbrace{2\mathbf{X}_l^T\mathbf{X}_l}_{\text{positive semi-definite}} + \underbrace{2\lambda'(\mathbf{X}^T L\mathbf{X})}_{\text{positive semi-definite}} \geq 0$$

, $\mathcal{L}(\mathbf{W})$ is convex, further convincing the previous closed-form solution.

(2) Label propagation algorithm

- The use of $\mathbf{D}^{-1}\mathbf{W}$ normalizes the propagation process so it is not biased by high degrees nodes, i.e., points with strong connections. Moreover, outliers have less effect on the result.
- The reason for assigning $\hat{\mathbf{Y}}^{(t+1)}$ with $\mathbf{Y}_l$ is simply to keep the labels for labeled data points. This is important as the labeled data points should not change during the propagation process.
- Assuming that each row of $\mathbf{Y}^{(0)}$ corresponds to a data point, and each column corresponds to a class, then for a labeled data point, the column corresponding to its class is set to 1, and all other columns are set to 0, indicating a one-hot encoding of class membership. For unlabeled data points, all columns are initially set to 0.

5. Graph Neural Networks

(1) In GNN, given $\hat{A}$, how to compute $\hat{A}^c = \underbrace{\hat{A}\dots\hat{A}}_{c}$ efficiently when c is large?

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

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

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

(1) $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$, where $\tilde{A} = A + I$, and $\tilde{D} = \text{diag}\left(\sum_{i=1}^{n}\tilde{A}_{i1}, \dots, \sum_{i=1}^{n}\tilde{A}_{in}\right)$ are symmetric matrices; hence, $\hat{A}$ is also symmetric. Performing eigendecomposition on $\hat{A}$ results in $V\Lambda V^{-1}$. $\hat{A}^c$ can then be computed as follows:

$$\begin{aligned}
\hat{A}^c &= \underbrace{\hat{A}\dots\hat{A}}_{c} \\
&= \underbrace{(V\Lambda V^{-1})\dots(V\Lambda V^{-1})}_{c} \\
&= V\underbrace{\Lambda\dots\Lambda}_{c}V^{-1} \\
&= V\Lambda^c V^{-1} \\
&= V\text{diag}(\lambda_1^c, \dots, \lambda_n^c)V^{-1} \text{ (due to } \Lambda \text{ being a diagonal matrix)}
\end{aligned}$$

(2) Loss functions of node classification, graph classification, and link prediction.

- Node Classification

  Below are the notations that will be used to describe the process:

  - K as the number of classes.
  - $\mathcal{Y}_L$ as set of labeled node indices.
  - $\mathbf{Y} \in \{0,1\}^{L\times K}$ as label matrix, where $\mathbf{Y}_{ik} = 1$ implies that node i is a member of class k and $\mathbf{Y}_{ik} = 0$ implies non-membership of the node.
  - $\hat{\mathbf{A}}$ as preprocessed adjacency matrix.

  The loss function is then defined as:

$$\mathcal{L} = -\sum_{i\in\mathcal{Y}_L}\sum_{k=1}^{K}\mathbf{Y}_{ik}\ln(\mathbf{Z}_{ik})$$

, where $\mathbf{Z}_{ik}$ is the predicted probability that node i belongs to class k.

To compute the predicted probability $\mathbf{Z}_{ik}$, we first obtain an embedding vector $\mathbf{z}_i$ for each node by employing a node embedding model based on Graph Convolutional Networks. Following this, we update the embedding vector $\mathbf{z}_i$ using the softmax function: $\mathbf{z}_i \leftarrow \text{softmax}(\mathbf{z}_i)$, normalizing the vector to represent the probability distribution across the classes for node i. The output of GCN is also represented by:

$$\mathbf{Z} = f(\mathbf{X}, \mathbf{A}) = \text{softmax}(\hat{\mathbf{A}}\text{ReLu}(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(0)})\mathbf{W}^{(1)})$$

- Graph Classification

  Below are the notations that will be used to describe the process:

  - K as the number of classes.
  - $\mathcal{G} = \{G_1, G_2, \ldots, G_N\}$ as set of graphs, where graph G_j has n nodes, feature map $\mathbf{X}_j \in \mathbb{R}^{n_j \times d}$, and affinity matrix $\hat{A}_j \in \mathbb{R}^{n_j \times n_j}$.
  - $\mathbf{Y} \in \{0,1\}^{L \times K}$ as label matrix, where $\mathbf{Y}_{jk} = 1$ implies that graph G_j is a member of class k and $\mathbf{Y}_{jk} = 0$ implies non-membership of the graph.
  - $\mathbf{z}_j = \sum_{m=1}^{n_j} z_{jm}$ as the feature of the graph G_j.

  The loss function is then defined as:

$$\mathcal{L} = -\sum_{j=1}^{N}\sum_{k=1}^{K} \mathbf{Y}_{jk} \ln(\mathbf{Z}_{jk})$$

  , where $\mathbf{Z}_{jk}$ is the predicted probability that graph G_j belongs to class k.

  To compute the predicted probability $\mathbf{Z}_{jk}$, we first obtain an embedding vector $\mathbf{z}_j$ for the graph G_j, each node i in G_j is first represented by an embedding vector $\mathbf{z}_{ji}$ using a node embedding model based on Graph Convolutional Networks. The embedding vector $\mathbf{z}_j$ for the graph is then updated by applying the softmax function across the sum of the node embedding vectors: $\mathbf{z}_j \leftarrow \text{softmax}\left(\sum_{i=1}^{n_j} \mathbf{z}_{ji}\right)$, normalizing the vector to represent the probability distribution across the classes for graph G_j. The output of GCN is also represented by:

$$\mathbf{Z} = f(\mathbf{X}, \mathbf{A}) = \text{softmax}(\hat{\mathbf{A}}\text{ReLu}(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(0)})\mathbf{W}^{(1)})$$

- Link Prediction

  Below are the notations that will be used to describe the process:

  - n as the number of nodes in the graph.
  - d as the number of dimensions of the graph's feature map.
  - $\mathbf{X}_j \in \mathbb{R}^{n \times d}$ as the graph's feature map.
  - $\hat{\mathbf{A}} \in \mathbb{R}^{n \times n}$ as the graph's affinity matrix, where $\hat{\mathbf{A}}_{ij} = 1$ implies that link from node i to j present and $\hat{\mathbf{A}}_{ij} = 0$ implies absence of the node.
  - Ω as the set of known links in the original graph.

The loss function is then defined as:

$$\mathcal{L} = -\sum_{(i,j)\in\Omega} \hat{\mathbf{A}}_{ij} \ln(\sigma(\mathbf{z}_i^T \mathbf{z}_j))$$

, where σ represents the logistic sigmoid function, converting the dot product of the embedding vectors into a probability.

To predict the presence of a link between two nodes, we utilize the embedding vectors derived from a Graph Convolutional Network. The probability that a link exists between nodes i and j is modeled using the sigmoid function applied to the dot product of their embedding vectors:

$$\Pr(\mathbf{A}_{ij} = 1 | \mathbf{z}_i, \mathbf{z}_j) = \sigma(\mathbf{z}_i^T \mathbf{z}_j)$$

, where $\mathbf{z}_i$ is the stochastic latent variables summarized in $\mathbf{Z}$, and $\mathbf{Z}$ can be computed as:

$$\mathbf{Z} = f(\mathbf{X}, \mathbf{A}) = \hat{\mathbf{A}}\text{ReLu}(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(0)})\mathbf{W}^{(1)}$$

(3) Algorithm examples for the following methods:

- parametric method: Linear Regression, GraphSAGE
- nonparametric method: k-Nearest Neighbors, Decision Tree
- inductive learning: Gradient Boosting, Support Vector Machine
- transductive learning: Graph Convolutional Network, Spectral Clustering
- semi-supervised learning: Label Propogation, Graph Convolutional Network for Graph-based Semi-supervised Learning

6. Nonlinear Dimensionality Reduction

(1) Show the algorithm of multidimensional scaling here.

(2) 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.

(3) 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 based on the methods learned in Lecture 07.

(1) Multidimensional Scaling
Parameters:

- $\mathbf{D} \in \mathbb{R}^{(n\times n)}$ as the distances between points.
- k as the number of dimensions.

Expected return:

- $\mathbf{X} \in \mathbb{R}^{(k\times n)}$ as the original position of points (recovering task).

Algorithm:

1. Compute the squared dissimilarity matrix, $\hat{\mathbf{D}}$, where $\hat{\mathbf{D}}_{ij} = \mathbf{D}_{ij}^2, \forall i, j$.

2. Apply double centering to $\hat{\mathbf{D}}$:

$$\mathbf{B} = -\frac{1}{2} J \hat{\mathbf{D}} J$$

where $J = I_n - \frac{1}{n}\mathbf{1}\mathbf{1}^T$ and $\mathbf{1}$ is an n-dimensional column vector of ones. This implies that each element in $\mathbf{B}$ is:

$$\mathbf{B}_{ij} = -\frac{1}{2}\left(\hat{\mathbf{D}}_{ij} - \frac{1}{n}\sum_{k=1}^{n}\hat{\mathbf{D}}_{ik} - \frac{1}{n}\sum_{k=1}^{n}\hat{\mathbf{D}}_{kj} + \frac{1}{n^2}\sum_{k=1}^{n}\sum_{l=1}^{n}\hat{\mathbf{D}}_{kl}\right) \text{ (row and column centering)}$$

3. Perform the eigenvalue decomposition of $\mathbf{B}$:

$$\mathbf{B} = V \Lambda V^T$$

where V is the matrix of eigenvectors and Λ is the diagonal matrix of eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$.

4. Select the k largest eigenvalues and their corresponding eigenvectors. Let Λ_k be the diagonal matrix containing the k largest eigenvalues and V_k the matrix containing the corresponding eigenvectors. Then, compute and return the k-dimensional representation of the data, $\mathbf{X} = V_k \Lambda_k^{\frac{1}{2}}$, where

$$\Lambda_k^{\frac{1}{2}} = [\operatorname{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_k}), \mathbf{0}_{k\times(n-k)}]$$

(2) t-SNE out-of-sample extension

The brief idea is to use the original t-SNE mapping to find the relationships between points and then apply some form of approximation technique to map new points.

Parameters:

- $\mathbf{X}_{\text{train}} \in \mathbb{R}^{d\times n}$ as the original high-dimensional training data.
- $\mathbf{X}_{\text{new}} \in \mathbb{R}^{d\times m}$ as the new high-dimensional data points to be embedded.

Expected return:

- $\mathbf{Z}_{\text{new}} \in \mathbb{R}^{k\times m}$ as the low-dimensional embedding of the new data points.

Algorithm:

1. Obtain a low-dimensional embedding of the training data, $\mathbf{Z}_{\text{train}} \in \mathbb{R}^{k\times n}$, by computing t-SNE($\mathbf{X}_{\text{train}}$).

2. Compute the Gaussian perplexity-based pairwise affinities in the original high-dimensional space:

$$P_{j|i} = \frac{\exp(-\|x_{\text{new}}^i - x_{\text{train}}^j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|x_{\text{new}}^i - x_{\text{train}}^k\|^2/2\sigma_i^2)}, \text{ for } i \neq j$$

, where σ_i^2 is the variance of the Gaussian kernel around sample i in $\mathbf{X}_{\text{train}}$.

3. Obtain joint probabilities by symmetrizing the affinities:

$$\begin{aligned} P_{ij} &= \frac{P_{j|i} + P_{i|j}}{2n} \\ &= \frac{\exp(-\|x^i_{\text{new}} - x^j_{\text{train}}\|^2/2\sigma_i^2)}{\sum_{k=1}^{n} \exp(-\|x^i_{\text{new}} - x^k_{\text{train}}\|^2/2\sigma_i^2)}, \text{ for } i \neq j\text{; otherwise, } 0 \end{aligned}$$

4. Initialize $\mathbf{Z}_{\text{new}}$ by some method, for example, by placing new points at the center of mass of their nearest neighbors in $\mathbf{Z}_{\text{train}}$.

5. Compute the pairwise affinities in the low-dimensional space using a t-distribution:

$$Q_{ij} = \frac{(1 + \|z^i_{\text{new}} - z^j_{\text{train}}\|^2)^{-1}}{\sum_{k=1}^{m} \sum_{l=1}^{n} (1 + \|z^k_{\text{new}} - z^l_{\text{train}}\|^2)^{-1}}, \text{ for } i \neq j$$

6. Optimize $\mathbf{Z}_{\text{new}}$ by minimizing the Kullback-Leibler divergence between P and Q using gradient descent:

$$\min_{\mathbf{Z}_{\text{new}}} \mathcal{L} = \min_{\mathbf{Z}_{\text{new}}} \text{KL}(P||Q) = \min_{\mathbf{Z}_{\text{new}}} \sum_{i,j} P_{ij} \log \frac{P_{ij}}{Q_{ij}}$$

, where the gradient with respect to z^i_{new} is given by:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial z^i_{\text{new}}} &= \frac{\partial}{\partial z^i_{\text{new}}} \sum_j P_{ij} \log \frac{P_{ij}}{Q_{ij}} \\ &= -\sum_j P_{ij} \cdot \frac{\partial}{\partial z^i_{\text{new}}} \log Q_{ij} \\ &= -\sum_j (P_{ij} - Q_{ij}) \cdot \left(-\frac{(z^i_{\text{new}} - z^j_{\text{train}})}{1 + \|z^i_{\text{new}} - z^j_{\text{train}}\|^2} \right) \\ &= \sum_j (P_{ij} - Q_{ij}) \cdot (z^i_{\text{new}} - z^j_{\text{train}}) \cdot (1 + \|z^i_{\text{new}} - z^j_{\text{train}}\|^2)^{-1} \end{aligned}$$

7. Update $\mathbf{Z}_{\text{new}}$ iteratively based on the gradient until convergence.

(3) t-SNE to detect outliers
Method:

1. Apply t-SNE to the dataset $\mathbf{X}$, reducing its dimensionality to either two or three dimensions (for interpretation purposes).

2. Present the dimensionally reduced dataset $\mathbf{X}$ visually in the corresponding lower-dimensional space.

3. Examine the spatial distribution of data points in the visualization. Outliers are typically identifiable as data points that are markedly isolated from main clusters or groupings, indicating a significant deviation from the majority of data samples.

4. Points that are far away from the main groups are marked as outliers and the corresponding points in the data set are the outliers that we are looking for.

Additionally, instead of relying on "eyes" to visualize and locate the outliers manually, it is possible to apply a clustering algorithm for objectivity and scalability purposes.

By applying a clustering algorithm, such as k-means or DBSCAN, we can group similar data points together based on their features in the reduced space. Outliers will stand out as points that don't fit well into any of these groups or, in the case of DBSCAN, are labeled as "noise" due to their distance from dense clusters.

7. Generative Models

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

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

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

(1) The evidence lower bound (ELBO) for the VAE model is derived as follows:

$$
\begin{aligned}
\log p_\theta(x) &= \log p_\theta(x) \underbrace{\int q_\phi(z|x)dz}_{1} \\
&= \int \log p_\theta(x) q_\phi(z|x)dz \\
&= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x)] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{p_\theta(z|x)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right] + \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right] + \underbrace{D_{\text{KL}}\left(q_\phi(z|x) \| p_\theta(z|x)\right)}_{\geq 0} \quad \text{(by KL divergence definition)} \\
&\geq \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right] \\
&\geq \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)}\right] \\
&\geq \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(z)}{q_\phi(z|x)}\right] \\
&\geq \underbrace{\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{\text{KL}}\left(q_\phi(z|x) \| p_\theta(z)\right)}_{\mathcal{L}_{\phi,\theta}(x)}
\end{aligned}
$$

Hence, the evidence lower bound for the VAE model, $\mathcal{L}_{\phi,\theta}(x)$, is derived.

(2) We will use the binary cross-entropy function in deriving the objective function for a discriminator D and a generator G in GAN.

D is trained to improve its ability to distinguish real from generated data; therefore, we will separate it into two cases:

- If $y = 1$ (labeled as real), then $\hat{y} = D(x)$, which implies that $\mathcal{L} = \log(D(x))$.

- If $y = 0$ (labeled as fake), then $\hat{y} = D(G(z))$, which implies that $\mathcal{L} = \log(1 - D(G(z)))$.

, where $\mathcal{L} = -\sum (y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$ is the binary cross-entropy function. Adding both of the cases, we have:

$$\mathcal{L} = -\sum (\log(D(x)) + \log(1 - D(G(z))))$$
$$\Rightarrow \mathcal{L}_D = -\mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] - \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

We are interested to find:

$$\arg\min_D \mathcal{L}_D$$

; hence, minimizing the discriminator's objective function, $\mathcal{L}_D$.

By the derived $\mathcal{L}_D$, the generator minimizes the involved part of the loss, which is $-\mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$; however, minimizing this expression directly leads to diminished gradients early in training, as when $D(G(z))$ is close to 0, the gradient of $\log(1 - D(G(z)))$ with respect to G is very small; hence, slowing the learning rate for G.

Reformulation is done by maximizing $\mathbb{E}_{z \sim p_z(z)}[\log(D(G(z)))]$. Thus, the objective function for G is written as:

$$\mathcal{L}_G = -\mathbb{E}_{z \sim p_z(z)}[\log(D(G(z)))]$$

Similarly, we are interested to find:

$$\arg\min_G \mathcal{L}_G$$

; hence, minimizing the generator's objective function, $\mathcal{L}_G$.

(3) Relationship between diffusion models and Langevin dynamics

Diffusion models and Langevin dynamics use the principle of transitioning between states of order and disorder through the application of noise. Diffusion models learn to methodically convert random noise into structured samples that resemble a target dataset. This process mirrors natural diffusion, where noise is incrementally added to a system until it becomes completely random. Similarly, Langevin dynamics uses stochastic equations to conceive the data distribution, employing noise to gradually move towards more ”probable” states that align with the target data.

They use noise as a tool to explore and define the shape of data distributions. While Langevin dynamics samples from a distribution by following noisy gradients directly, diffusion models undertake a learned process of reversing the noise to reveal the structured data. In doing so, diffusion models perform operations on the concept inherent in Langevin dynamics, applying it within a framework that learns from data how to reverse the diffusion process, thereby generating new, coherent samples from noise.