DDA4210/AIR6002 - 2024 Spring **Assignment Guru: Fangchen Yu**

Assignment 2 Solution

Note: The **blue** words represent the scoring points

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

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

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: [4 points]

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

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

Solution.

The inequality is true if each term satisfies the inequality. So, we show

$$\exp\left(-y_i f\left(x_i\right)\right) \geq \mathbb{1}\left(H\left(x_i\right) \neq y_i\right).$$

First, consider any points correctly classified. This means $H\left(x_i\right) = y_i \implies \mathbb{1}\left(H\left(x_i\right) \neq y_i\right) = 0$. Then, since $e^x \geq 0 \forall x$, we have

$$\exp\left(-y_i f\left(x_i\right)\right) \geq 0 = \mathbb{1}\left(H\left(x_i\right) \neq y_i\right).$$

Next, consider any points incorrectly classified. This means $H\left(x_i\right) \neq y_i \implies \mathbb{1}\left(H\left(x_i\right) \neq y_i\right) = 1$. Then, note for incorrectly classified points, $\text{sign}\left(f\left(x_i\right)\right) \neq y_i \implies -y_i f\left(x_i\right) \geq 0$ Then, since $e^x \geq 1 \forall x \geq 0$, we have

$$\exp\left(-y_i f\left(x_i\right)\right) \geq 1 = \mathbb{1}\left(H\left(x_i\right) \neq y_i\right).$$

Thus, every term in the sequence corresponding to a correctly or incorrectly classified point is bounded above by the exponential loss function E. Therefore,

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

(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}: [4 points]

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

Solution.
We note

$$D_1(i) = \frac{1}{N} \tag{1}$$

and

$$D_{t+1}(i) = \frac{D_t(i) \exp\left(-\alpha_t y_i h_t\left(x_i\right)\right)}{Z_t}, \tag{2}$$

where Z_t is a normalization factor chosen so that D_{t+1} will be a distribution. That is,

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

Using (1) and (2), we find

$$D_{T+1}(i) = \frac{1}{N} \prod_{t=1}^{T} \frac{e^{-\alpha_t y_i h_t(x_i)}}{Z_t}.$$

(c) Show that $E = \sum_{i=1}^{N} \frac{1}{N} e^{\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i)}$. [2 points]

Solution.
Recall that

$$E = \frac{1}{N} \sum_{i=1}^{N} e^{-y_i f(x_i)}.$$

Note that

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

So

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

(d) Show that [2 points]

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

Solution.
We found above that

$$D_{T+1}(i) = \frac{1}{N} \prod_{t=1}^{T} \frac{e^{-\alpha_t y_i h_t(x_i)}}{Z_t}.$$

This implies

$$D_{T+1}(i) \cdot \prod_{t=1}^{T} Z_t = \frac{1}{N} \prod_{t=1}^{T} e^{-\alpha_t y_i h_t(x_i)}$$

$$\implies D_{T+1}(i) \cdot \prod_{t=1}^{T} Z_t = \frac{1}{N} e^{\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i)}$$

$$\implies \sum_{i=1}^{N} D_{T+1}(i) \cdot \prod_{t=1}^{T} Z_t = \sum_{i=1}^{N} \frac{1}{N} e^{\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i)}$$

$$\implies \sum_{i=1}^{N} D_{T+1}(i) \prod_{t=1}^{T} Z_t = \sum_{i=1}^{N} \frac{1}{N} e^{\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i)}.$$

We showed above that the right side is equal to E. We also know that $\sum_{i=1}^{N} D_{T+1}(i) = 1$ as in the hint. (Hint: Recall that $\sum_{i=1}^{N} D_t(i) = 1$ because D is a distribution.)

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

(e) Show that the normalizer Z_t can be written as [4 points]

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

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

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

Solution.
Note,

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

Note, if h_t classifies point x_i correctly (so $h_t(x_i) * y_i = 1$), we have

$$D_t(i) \exp(-\alpha_t y_i h_t(x_i)) = D_t(i) \exp(-\alpha_t) = (1 - \mathbb{1}(h_t(x_i) \neq y_i)) D_t(i) \exp(-\alpha_t).$$

Note, if h_t classifies point x_i incorrectly (so $h_t(x_i) * y_i = -1$), we have

$$D_t(i) \exp(-\alpha_t y_i h_t(x_i)) = D_t(i) \exp(\alpha_t) = \mathbb{1}(h_t(x_i) \neq y_i) D_t(i) \exp(\alpha_t).$$

So

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

because $\sum_{i=1}^{N} D_t(i) = 1$.

(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: [4 points]

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

Solution.

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

We note Z_t is convex, so to minimize Z_t, we find α_t such that

$$\frac{dZ_t}{d\alpha_t} = -\left(1 - \epsilon_t\right) \exp\left(-\alpha_t\right) + \epsilon_t \exp\left(\alpha_t\right) = 0$$

Multiplying by $\exp\left(\alpha_t\right)$,

$$
\begin{aligned}
&\implies -\left(1 - \epsilon_t\right) + \epsilon_t \exp\left(2\alpha_t\right) = 0. \\
&\implies \epsilon_t \exp\left(2\alpha_t\right) = 1 - \epsilon_t \\
&\implies \exp\left(2\alpha_t\right) = \frac{1 - \epsilon_t}{\epsilon_t} \\
&\implies \alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right).
\end{aligned}
$$

(g) [**AIR6002**] Implement the `GradientBoosting.fit()` and `AdaBoost.fit()` methods in the notebook "Q1_AdaBoost.ipynb" provided for you. [15 points]

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. [10 points]

Solution.

i) Training loss in gradient boosting decreases in a smooth fashion towards 0, but test loss stops decreasing quickly and then flattens with a slight increase.

ii) Training and test losses in AdaBoost both steadily approach relatively-small values without increasing much, and are not smooth (lots of spikes). This is attributed to the fact that AdaBoost uses classifiers, not regressors.

(i) [**AIR6002**] Compare the final loss values of the two models. Which performed better on the classification dataset? [3 points]

Solution.

Gradient boosting performed better on training but AdaBoost performed better on test. That is, AdaBoost performed better overall as it is naturally more inclined to a classification dataset (it uses classifiers as opposed to regressors).

(j) [**AIR6002**] For AdaBoost, where are the dataset weights the largest, and where are they the smallest? [2 points]

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

Solution.

The weights are the largest where there is the most ambiguity in classes. In the case of the given dataset, they are largest at the the edges of the spirals and smallest away from these edges.

2. Recommendation Systems

(1) What are the differences between collaborative filtering and content-based methods? [4 points]

Solution.

- *Data used for recommendation:* Collaborative filtering relies on user-item interaction data to make recommendations. Content-based methods, on the other hand, use item features or attributes to make recommendations.
- *Type of recommendation:* Collaborative filtering is good at recommending items that are popular among similar users. Content-based methods, on the other hand, are good at recommending niche items that are similar to the items a user has interacted with in the past.
- *Cold start problem:* Collaborative filtering suffers from the cold start problem, where it is difficult to make recommendations for new users or items that have no interaction data. Content-based methods, on the other hand, do not suffer from this problem as they can make recommendations based on the item features.

(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. [6 points]

Solution.

- Notations:
  ℓ is a loss function
  $f : \mathbb{R}^{d_u} \to \mathbb{R}^{n_I}$
  d_U (d_I) is number of user (item) features
  n_U (n_I) is the number of users (items)
  f_i is the i-th output of f
  z_u is the feature vector of u-th user.
- Recommendation for users:
$$\min_f \sum_{(u,i)\in\Omega} \ell(r_{ui}, f_i(z_u))$$
  For a neural network with two hidden layers,
  $f(z_u) = \sigma_3(W_3 h_2 + b_3)$,
  $h_2 = \sigma_2(W_2 h_1 + b_2)$,
  $h_1 = \sigma_1(W_1 z_u + b_1)$,
  where σ_i is the activation function, W_i is the weight matrices of layers, b_i is the corresponding bias vector.
- To make recommendations for a new user, we can use the same neural network model to predict the ratings for all items based on the feature vector of the new user. Then, we can recommend the top-k items with the highest predicted ratings to the new user.

3. Spectral Clustering

Prove $\frac{\mathbf{u}^\top \mathbf{L}\mathbf{u}}{\mathbf{u}^\top \mathbf{D}\mathbf{u}} = \mathrm{Ncut}(A, B)$. See the definitions on page 23 of Lecture 05-I. [10 points]

Solution. Let $\mathbf{u} = [u_1, \ldots, u_n]^\top$, where $u_i = \begin{cases} \frac{1}{\mathrm{Vol}(A)}, & \text{if } i \in A \\ -\frac{1}{\mathrm{Vol}(B)}, & \text{if } i \in B \end{cases}$, $\mathbf{L} = \mathbf{D} - \mathbf{W}$, and $\mathbf{D} = \mathrm{diag}\,(d_1, \ldots, d_n)$. Then we have

$$
\begin{aligned}
\mathbf{u}^\top \mathbf{L}\mathbf{u} &= \mathbf{u}^\top (\mathbf{D} - \mathbf{W})\mathbf{u} \\
&= \mathbf{u}^\top \mathbf{D}\mathbf{u} - \mathbf{u}^\top \mathbf{W}\mathbf{u} \\
&= \sum_{i=1}^{n} d_i u_i^2 - \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} u_i u_j \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} u_i^2 - \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} u_i u_j \\
&= \frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} u_i^2 + \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} u_j^2 - 2\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} u_i u_j \right) \\
&= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} \left(u_i - u_j\right)^2 \\
&= \sum_{i\in A}\sum_{j\in B} w_{ij} \left(u_i - u_j\right)^2 \\
&= \sum_{i\in A}\sum_{j\in B} w_{ij} \left(\frac{1}{\mathrm{Vol}(A)} + \frac{1}{\mathrm{Vol}(B)} \right)^2
\end{aligned}
$$

Similarly,

$$
\begin{aligned}
\mathbf{u}^\top \mathbf{D}\mathbf{u} &= \sum_{i=1}^{n} d_i u_i^2 \\
&= \sum_{i\in A} d_i u_i^2 + \sum_{j\in B} d_j u_j^2 \\
&= \frac{\sum_{i\in A} d_i}{\mathrm{Vol}(A)^2} + \frac{\sum_{j\in B} d_j}{\mathrm{Vol}(B)^2} \\
&= \frac{\mathrm{Vol}(A)}{\mathrm{Vol}(A)^2} + \frac{\mathrm{Vol}(B)}{\mathrm{Vol}(B)^2} \\
&= \frac{1}{\mathrm{Vol}(A)} + \frac{1}{\mathrm{Vol}(B)}
\end{aligned}
$$

Thus, we obtain:

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

4. Semi-supervised Learning

(1) 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. [10 points]

Solution.
Given the training data consisting of labeled and unlabeled samples, we can use the unlabeled samples as a regularizer. That is, if data point i and j are similar (large A_{ij}), then the predicted labels f_i and f_j are also similar. Let L be the set of indices for the labeled samples and U be the set of indices for the unlabeled samples, $|L \cup U| = n$. Then, the objective function can be written as:

$$\begin{aligned}J(W) &= \sum_{i\in L} \|\mathrm{y}_i - \mathrm{f}_i\|_2^2 + \lambda \sum_{i,j\in L,U} A_{ij}\,\|\mathrm{f}_i - \mathrm{f}_j\|_2^2 \\ &= \sum_{i\in L} \|\mathrm{y}_i - \mathrm{x}_i\mathbf{w}\|_2^2 + \lambda \sum_{i,j\in L,U} A_{ij}\,\|\mathrm{x}_i\mathbf{w} - \mathrm{x}_j\mathbf{w}\|_2^2 \\ &= \mathrm{Tr}((Y - X_L W)^\top (Y - X_L W)) + 2\lambda \mathrm{Tr}((XW)^\top L(XW))\end{aligned}$$

where $L = D - A, d_{ii} = \sum_{j=1}^n A_{ij}, X_L = [x_1, \ldots, x_l] \in \mathbb{R}^{l\times d}, Y \in \mathbb{R}^{l\times k}, W \in \mathbb{R}^{d\times k}$. We can compute the gradient of the objective function:

$$\frac{\partial J(W)}{\partial W} = -2X_L^\top Y + 2X_L^\top X_L W + 4\lambda X^\top L X W$$

We can obtain the closed-form solution by setting the gradient to zero and solving for W:

$$\frac{\partial J(W)}{\partial W} = 0 \Rightarrow W = (X_L^\top X_L + 2\lambda X^\top L X)^{-1} X_L^\top Y$$

(2) 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 $\mathbf{Y}^{(0)}$? [5 points]

Solution.

- The label propagation algorithm uses the matrix $D^{-1}W$ instead of W to ensure that the similarity matrix is normalized and less sensitive to the scale of the data. Normalizing the rows of W ensures that the algorithm is not biased towards data points with many strong connections and reduces the impact of noisy or irrelevant data points.
- We set $\hat{\mathbf{Y}}_l^{(t+1)} \leftarrow \mathbf{Y}_l$ to carry forward the information from the original data. In this way, we avoid losing information from the labeled nodes.
- Assume the label set be $\{1, 2, \ldots, C\}$, we can initialize $\mathbf{Y}^{(0)}$ using a one-hot encoding. Specifically, if the label is 1, then $y_i = (1, 0, \ldots, 0)^\top \in \mathbb{R}^C$.

5. Graph Neural Networks

(1) 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? [5 points]

Solution.

Since $\hat{A}$ is a symmetric matrix, we perform the eigendecomposition: $\hat{A} = V\Lambda V^{-1}, \Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$. Then $\hat{A}^c = (V\Lambda V^{-1})^c = V\Lambda^c V^{-1} = V\text{diag}(\lambda_1^c, \lambda_2^c, \ldots, \lambda_n^c)V^{-1}$. Generally, we can perform the singular value decomposition $\hat{A} = U\Lambda V^\top$ and obtain $\hat{A}^c = (U\Lambda V^\top)^c = U\Lambda^c V^\top$.

(2) Show the loss functions of node classification, graph classification, and link prediction. Explain your notations. [5 points]

Solution.

- **Node classification:** is to predict the label of each node in a graph. The loss function is defined as
$$\mathcal{L} = -\sum_{i \in \mathcal{Y}_L} \sum_{k=1}^{K} \mathbf{Y}_{ik} \ln(\mathbf{Z}_{ik}),$$
where $\mathcal{Y}_L$ is the set of labeled node indices, K is the number of classes, $\mathbf{Y}_{ik}$ is the indicator of whether the labeled node i is of class k, $\mathbf{Z}_{ik}$ is the GCN output of the probability of node i in class k. Typically, we use the common structure of GCN:
$$\mathbf{Z} = f(\mathbf{X}, \mathbf{A}) = \text{softmax}(\hat{\mathbf{A}}\text{ReLu}(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(0)})\mathbf{W}^{(1)}),$$
where $\mathbf{X}$ is the feature matrix, $\hat{\mathbf{A}}$ is the preprocessed adjacency matrix, $\mathbf{W}$ is the parameter matrix of hidden layer 0 or 1.
- **Graph classification:** is to predict a label for an entire graph. The loss function is defined as
$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{k=1}^{K} \mathbf{Y}_{ik} \ln(\mathbf{Z}_{ik}),$$
where N is the number of graphs.
- **Link prediction:** is to predict the presence or absence of edges between pairwise nodes in a graph. The loss function is defined as
$$\mathcal{L} = -\sum_{(i,j) \in \Omega} \mathbf{A}_{ij} \ln \sigma(\mathbf{z}_i^\top \mathbf{z}_j),$$
where Ω is the set of all observed edges in the graph and $\mathbf{A}$ is the adjacency matrix.

(3) Provide two examples of algorithms for each learning types or methods: [5 points]

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

Solution.

- **Parametric method**: linear regression, logistic regression; Graph Convolutional Network (GCN), Graph Attention Network (GAT), GraphSAGE
- **Non-parametric method**: k-nearest neighbors, decision tree; Graph Matching Network (GMN), Spectral Graph Convolutional Network (SGCN)
- **Inductive learning**: support vector machine, gradient boosting; Graph Attention Network (GAT), Graph Isomorphism Network (GIN)
- **Transductive learning**: label propagation, spectral clustering; Graph Convolutional Network (GCN), Graph Attention Network (GAT)
- **Semi-supervised learning**: label propagation, label spreading; Graph-based Semi-supervised Learning (GSSL) with GCN

Note: Some of these GNNs can fit into multiple categories, and there may be some overlap between the categories.

6. Nonlinear Dimensionality Reduction

(1) Show the algorithm of multidimensional scaling here. [5 points]

Solution. **Algorithm for Classical Multidimensional Scaling (CMDS):**

Input: dissimilarity matrix $D \in \mathbb{R}^{n\times n}$ (e.g., Euclidean distance), number of dimensions k.

1. Compute the geometric centering matrix $J = I - \frac{1}{n}E$, where I is an identity matrix and E is a matrix of all ones.
2. Compute the Gram matrix $G = -\frac{1}{2}JD^2J$, where D^2 denotes the squared Euclidean distance matrix.
3. Perform eigen-decomposition $G = U\Lambda U^\top$, where $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ and $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$.
4. Compute the k-dimensional representation of the data by $Y = [\text{diag}(\sqrt{\lambda_1}, \cdots, \sqrt{\lambda_k}), \mathbf{0}_{k\times(n-k)}]U^\top \in \mathbb{R}^{k\times n}$. Or equivalently, $Y = \Lambda_k^{\frac{1}{2}}U_k^\top$, where $\Lambda_k^{\frac{1}{2}}$ is the diagonal matrix of the square roots of the k largest eigenvalues and U_k is their corresponding eigenvectors.

Output: feature matrix $Y \in \mathbb{R}^{k\times n}$

(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. [5 points]

Solution.

Input: $X_{\text{train}} \in \mathbb{R}^{d \times n}$ with n samples, $X_{\text{new}} \in \mathbb{R}^{d \times m}$ with m samples.

1. Run t-SNE on the high-dimensional training data X_{train} to obtain the low-dimensional embedding Z_{train}.
2. Compute the pairwise similarity matrix between new samples and training samples in the original data space:

$$P_{i,j} = \frac{\exp(-\|x^i_{\text{new}} - x^j_{\text{train}}\|^2 / 2\sigma_i^2)}{\sum_{k=1}^{n} \exp(-\|x^i_{\text{new}} - x^k_{\text{train}}\|^2 / 2\sigma_i^2)} \Rightarrow P \in \mathbb{R}^{m \times n}$$

3. Compute the pairwise similarity matrix between new samples and training samples in the lower-dimensional space:

$$Q_{i,j} = \frac{(1 + \|z^i_{\text{new}} - z^j_{\text{train}}\|^2)^{-1}}{\sum_{k=1}^{m} \sum_{l=1}^{n} (1 + \|z^k_{\text{new}} - z^l_{\text{train}}\|^2)^{-1}} \Rightarrow Q \in \mathbb{R}^{m \times n}$$

4. Optimize the embedding of new samples z_{new} by minimizing the KL divergence between similarity matrices P and Q:

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

(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 bassed on the methods learned in Lecture 07. [5 points]

Solution.

Nonlinear dimensionality reduction (NLDR) methods can be used to detect outliers in high-dimensional data by projecting the data into a lower-dimensional space where outliers can be easier to identify. Here is a basic outline of a method or strategy to detect outliers based on NLDR:

Input:

- X: a matrix of size $d \times n$ containing the high-dimensional data features
- method: the NLDR method to be used (e.g., t-SNE, UMAP, etc.)
- k: the number of nearest neighbors to use for outlier detection
- threshold: the outlier detection threshold

Output:

- indices of the detected outliers in the original dataset

Steps:

1. Apply the chosen NLDR method to the high-dimensional data X to obtain a lower-dimensional representation of the data.
2. Calculate the pairwise distances between the data points in the lower-dimensional space.
3. For each data point z_i, find its k nearest neighbors in the lower-dimensional space.
4. Calculate the average distance between z_i and its k nearest neighbors.
5. Identify the outliers as those data points with an average distance greater than the threshold.

Note: This method is just one way to detect outliers based on NLDR, and there are many other methods and strategies available.

7. Generative Models

(1) Derive the evidence lower bound (ELBO) for the VAE model. [5 points]

Solution.

VAE aims to transform $\mathbf{x}$ into a prior distribution $p_{\mathbf{z}}$ using encoder $f_\phi : \mathbf{x} \to \mathbf{z}$ and then to reconstruct $\mathbf{x}$ using decoder $g_\theta : \mathbf{z} \to \mathbf{x}'$. We start with the log likelihood of the data and manipulate it to introduce the ELBO.

- Decompose the log-likelihood:

$$\begin{aligned}\log p_\theta(\mathbf{x}) &= \log \frac{p_\theta(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})}{p_\theta(\mathbf{z} \mid \mathbf{x})} = \log \frac{p_\theta(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})q_\phi(\mathbf{z} \mid \mathbf{x})}{p_\theta(\mathbf{z} \mid \mathbf{x})q_\phi(\mathbf{z} \mid \mathbf{x})} \\ &= \log p_\theta(\mathbf{x} \mid \mathbf{z}) - \log \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{z})} + \log \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p_\theta(\mathbf{z} \mid \mathbf{x})}\end{aligned}$$

- Take expectation:

$$\begin{aligned}\log p_\theta(\mathbf{x}) = \; & \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x})\right] = \int q_\phi(\mathbf{z} \mid \mathbf{x}) \log p_\theta(\mathbf{x}) d\mathbf{z} \\ = \; & \int q_\phi(\mathbf{z} \mid \mathbf{x}) \log p_\theta(\mathbf{x} \mid \mathbf{z}) d\mathbf{z} - \int q_\phi(\mathbf{z} \mid \mathbf{x}) \log \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{z})} d\mathbf{z} \\ & + \int q_\phi(\mathbf{z} \mid \mathbf{x}) \log \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p_\theta(\mathbf{z} \mid \mathbf{x})} d\mathbf{z} \\ = \; & \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - D_{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z})\right) \\ & + D_{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z} \mid \mathbf{x})\right)\end{aligned}$$

Then, we obtain

$$\log p_\theta(\mathbf{x}) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - D_{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z})\right) + D_{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z} \mid \mathbf{x})\right)$$

Because KL-divergence is always non-negative, we obtain

$$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - D_{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z})\right) \triangleq \mathcal{L}_{\phi,\theta}(\mathbf{x}),$$

where $\mathcal{L}_{\phi,\theta}(\mathbf{x})$ is the ELBO for $\log p_\theta(\mathbf{x})$.

(2) Derive the objective functions for the generator and discriminator in a GAN, respectively. [5 points]

Solution.

- Generator: The generator is trained to increase the chances of D producing a high probability for generated samples, thus

$$\text{minimize}_G \, \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

- Discriminator:
  - Ensure the discriminator $D's$ decisions over real data are accurate by

$$\text{maximize}_D \, \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}[\log D(\mathbf{x})]$$

  - Given a fake sample $G(\mathbf{z}), \mathbf{z} \sim p_z(\mathbf{z})$, the discriminator is expected to output a probability, $D(G(\mathbf{z}))$, close to zero by

$$\text{maximize}_D \, \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

  Thus, the objective function for the discriminator is

$$\text{maximize}_D \, \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

(3) Briefly explain the relationship between diffusion models and Langevin dynamics. [5 points]

Solution.

Diffusion models are a class of generative models that simulate the process of adding noise to data samples and then learning to reverse this process. Langevin dynamics is a physical process described by stochastic differential equations, used to model the behavior of particles in a potential field with the presence of thermal fluctuations.

(i) Diffusion models can be thought of as leveraging the principles underlying Langevin dynamics to perform generative tasks in machine learning. By simulating the process of adding and removing noise, diffusion models effectively "navigate" through the data distribution in a manner analogous to how Langevin dynamics navigates the energy landscape of a physical system.

(ii) Langevin dynamics can be viewed as a continuous counterpart to the discrete process modeled by diffusion models in machine learning.

8. Graph Convolutional Network

(1) [**AIR6002**] Implement the `GCNLayer.forward()` in the notebook "Q8_GCN.ipynb" provided for you. [10 points]

Solution.

```python
class GCNLayer(nn.Module):

    def __init__(self, c_in, c_out):
        super().__init__()
        self.projection = nn.Linear(c_in, c_out)

    def forward(self, node_feats, adj_matrix):
        # Calculate the number of neighbors for each node
        num_neighbours = adj_matrix.sum(dim=-1, keepdims=True)

        # Apply a linear projection to the node features
        node_feats = self.projection(node_feats)

        # Aggregate neighbor features for each node
        node_feats = torch.bmm(adj_matrix, node_feats)

        # Normalize aggregated features by the number of neighbors
        node_feats = node_feats / num_neighbours

        return node_feats
```

(2) [**AIR6002**] Define the adjacency matrix of the graph in the notebook with self-connections. [5 points]

Solution.

```python
adjacency_matrix = torch.Tensor([[[1, 1, 0, 0],
                                  [1, 1, 1, 1],
                                  [0, 1, 1, 1],
                                  [0, 1, 1, 1]]])
```

(3) [**AIR6002**] Apply the GCN layer defined in (1) on the graph in (2), and write down the output features for all nodes. [5 points]

Solution.

```python
Input_features = torch.Tensor([[[0., 1.],
                                [2., 3.],
                                [4., 5.],
                                [6., 7.]]])

Output_features = torch.Tensor([[[1., 2.],
                                 [3., 4.],
                                 [4., 5.],
                                 [4., 5.]]])
```