THU-70250043-0, Pattern Recognition (Spring 2021) **Homework: 11**

Decision Tree and GNN

Lecturer: Changshui Zhang zcs@mail.tsinghua.edu.cn

Hong Zhao vzhao@tsinghua.edu.cn

Student: Jingxuan Yang yangjx20@mails.tsinghua.edu.cn

Decision Tree

Consider a data set comprising 400 data points from class C_1 and 400 data points from class C_2. Suppose that a tree model A splits these data points into (300, 100) at the first leaf node and (100, 300) at the second leaf node, where (n, m) denotes that n points are assigned to C_1 and m points are assigned to C_2. Similarly, suppose that a second tree model B splits them into (200, 400) and (200, 0).

Evaluate the misclassification rates for the two trees and show that they are equal. Similarly, evaluate the cross-entropy and Gini index for the two trees and show that they are both lower for tree B than for tree A.

解: 决策树模型 A 的错分率为

$$\begin{aligned} i_A &= P_L i_A(N_L) + P_R i_A(N_R) \\ &= \frac{1}{2} \times \frac{1}{4} + \frac{1}{2} \times \frac{1}{4} \\ &= \frac{1}{4} \end{aligned} \tag{1}$$

决策树模型 B 的错分率为

$$\begin{aligned} i_B &= P_L i_A(N_L) + P_R i_A(N_R) \\ &= \frac{3}{4} \times \frac{1}{3} + \frac{1}{4} \times 0 \\ &= \frac{1}{4} \end{aligned} \tag{2}$$

所以 $i_A = i_B$, 即决策树模型 A 与 B 的错分率相等.

决策树模型 A 的交叉熵为

$$\begin{aligned} \hat{i}_A &= P_L \hat{i}_A(N_L) + P_R \hat{i}_A(N_R) \\ &= \frac{1}{2} \times \left(-\frac{3}{4} \log_2 \frac{3}{4} - \frac{1}{4} \log_2 \frac{1}{4} \right) + \frac{1}{2} \times \left(-\frac{1}{4} \log_2 \frac{1}{4} - \frac{3}{4} \log_2 \frac{3}{4} \right) \\ &= 0.8113 \end{aligned} \tag{3}$$

决策树模型 B 的交叉熵为

$$\begin{aligned}\hat{i}_B &= P_L\hat{i}_B(N_L) + P_R\hat{i}_B(N_R)\\ &= \frac{3}{4}\times\left(-\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3}\right) + \frac{1}{4}\times(-1\log_2 1)\\ &= 0.6887\end{aligned} \tag{4}$$

因此, 交叉熵 $\hat{i}_B < \hat{i}_A$.

决策树模型 A 的 Gini index 为

$$\begin{aligned}\tilde{i}_A &= P_L\tilde{i}_A(N_L) + P_R\tilde{i}_A(N_R)\\ &= \frac{1}{2}\times\left(1-\left(\frac{3}{4}\right)^2-\left(\frac{1}{4}\right)^2\right) + \frac{1}{2}\times\left(1-\left(\frac{1}{4}\right)^2-\left(\frac{3}{4}\right)^2\right)\\ &= \frac{3}{8}\end{aligned} \tag{5}$$

决策树模型 B 的 Gini index 为

$$\begin{aligned}\tilde{i}_B &= P_L\tilde{i}_B(N_L) + P_R\tilde{i}_B(N_R)\\ &= \frac{3}{4}\times\left(1-\left(\frac{1}{3}\right)^2-\left(\frac{2}{3}\right)^2\right) + \frac{1}{4}\times(1-1^2)\\ &= \frac{1}{3}\end{aligned} \tag{6}$$

因此, Gini index $\tilde{i}_B < \tilde{i}_A$.

Graph Neural Network (GNN)

One of the simplest possible propagation rule in GCN (Graph Convolutional Networks) is

$$f(H^i, A) = \sigma(AH^iW^i), \tag{7}$$

where A is a representative description of the graph structure in the form of an adjacency matrix, H^i is the feature matrix for layer i (H^i is an $N \times F^i$ matrix, where N is the number of nodes in the graph, F^i is the feature dimension of one node, and $H^0 = X$ is the input data), W^i is the weight matrix for layer i and σ is a non-linear activation function such as the ReLU function. The weight matrix has dimensions $F^i \times F^{i+1}$; in other words the size of the second dimension of the weight matrix determines the dimension of features at the next layer. If you are familiar with convolutional neural networks, this operation is similar to a filtering operation since these weights are shared across nodes in the graph.

Simplifications: Let's examine the propagation rule at its most simple level. Let

- $i = 1$, s.t. f is a function of the input feature matrix,
- σ be the identity function, and
- choose the weights s.t. $AH^0W^0 = AXW^0 = AX$.

In other words, $f(X, A) = AX$. This propagation rule is perhaps a bit too simple, but we will add in the missing parts later. As a side note, AX is now equivalent to the input layer of a multi-layer perceptron.

We'll use the following graph in Figure 1:

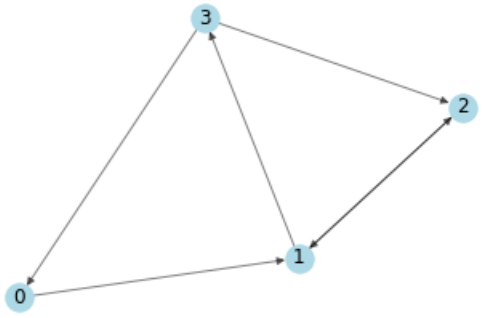

图 1: A simple directed graph

(a) Derive the adjacency matrix representation A and degree matrix D. Note that in this case a node n is a neighbor of node v if there exists an edge from v to n.

解: 邻接矩阵为

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \tag{8}$$

出度矩阵为

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \tag{9}$$

(b) We generate features for every node based on its index, i.e. $[i, -i]$ for node i. Apply the propagation rule on adjacency matrix A and input features X to derive the output matrix.

解: 由题意可知输入特征为

$$X = \begin{bmatrix} 0 & 0 \\ 1 & -1 \\ 2 & -2 \\ 3 & -3 \end{bmatrix} \tag{10}$$

所以, 输出矩阵为

$$f(X, A) = AX = \begin{bmatrix} 1 & -1 \\ 5 & -5 \\ 1 & -1 \\ 2 & -2 \end{bmatrix} \tag{11}$$

(c) We found that nodes with large degrees will have large values in their feature representation while nodes with small degrees will have small values. This can cause vanishing or exploding gradients. Therefore, the feature representations can be normalized by node degree by transforming the adjacency matrix A by multiplying it with the inverse degree matrix D. Thus our simplified propagation rule looks like this: $f(x, A) = D^{-1}AX$. We also found that the aggregated representation of a node does not include its own features. To address the problem, we add a self-loop to each node, by adding the identity matrix I to the adjacency matrix A before applying the propagation rule, that is, $\hat{A} = A + I$. $\hat{D}$ is the degree matrix of $\hat{A}$, i.e., the degree matrix of A with forced self-loops. Derive the output matrix after normalizing the feature representations and adding self-loops.

解: 加上自回路后, 邻接矩阵为

$$\hat{A} = A + I = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix} \tag{12}$$

其度矩阵为

$$\hat{D} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} \tag{13}$$

所以, 输出矩阵为

$$f(X, \hat{A}) = \hat{D}^{-1}\hat{A}X = \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} \\ 2 & -2 \\ \frac{3}{2} & -\frac{3}{2} \\ \frac{5}{3} & -\frac{5}{3} \end{bmatrix} \tag{14}$$

(d) We add back the weights matrix W as follows:

$$W = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

Derive the output matrix.

解: 输出矩阵为

$$f(X, \hat{A}) = \hat{D}^{-1}\hat{A}XW = \begin{bmatrix} 1 & -1 \\ 4 & -4 \\ 3 & -3 \\ \frac{10}{3} & -\frac{10}{3} \end{bmatrix} \tag{15}$$

(e) Use the same W and add the ReLU activation function for σ. Derive the output matrix.

解: 输出矩阵为

$$f(X, \hat{A}) = \text{ReLU}\left(\hat{D}^{-1}\hat{A}XW\right) = \begin{bmatrix} 1 & 0 \\ 4 & 0 \\ 3 & 0 \\ \frac{10}{3} & 0 \end{bmatrix} \tag{16}$$

(f) We can apply a graph convolutional network on a real graph, Zachary's karate club. Zachary's karate club is a commonly used social network where nodes represent members of a karate club and the edges their mutual relations. While Zachary was studying the karate club, a conflict arose between the administrator and the instructor which resulted in the club splitting in two. The figure below shows the graph representation of the network and nodes are labeled according to which part of the club. The administrator and instructor are marked with 'A' and 'I', respectively.

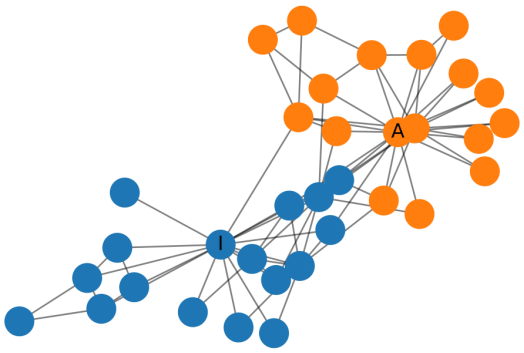

图 2: Zachary's Karate Club

You can use the following python codes to get the dataset:

```
from networkx import karate_club_graph

zkc = karate_club_graph()
```

Design a GNN to separate communities in Zachary's Karate Club. We here use just the identity matrix as input representation, that is, each node is represented as a one-hot encoded variable. Show the final output feature representations for the nodes of Figure 2.

Hint: Please try a GCN with two hidden layers just like (e), and initialize the weights randomly, then extract the feature representations and plot them. You will find even randomly initialized GCNs can separate communities in Zachary's Karate Club. Next, you can try your own GNN for better performance.

解: 首先绘制 Zachary's Karate Club 如图 3 所示, 其中标号 0 的节点对应 'I', 标号 33 的节点对应 'A'.

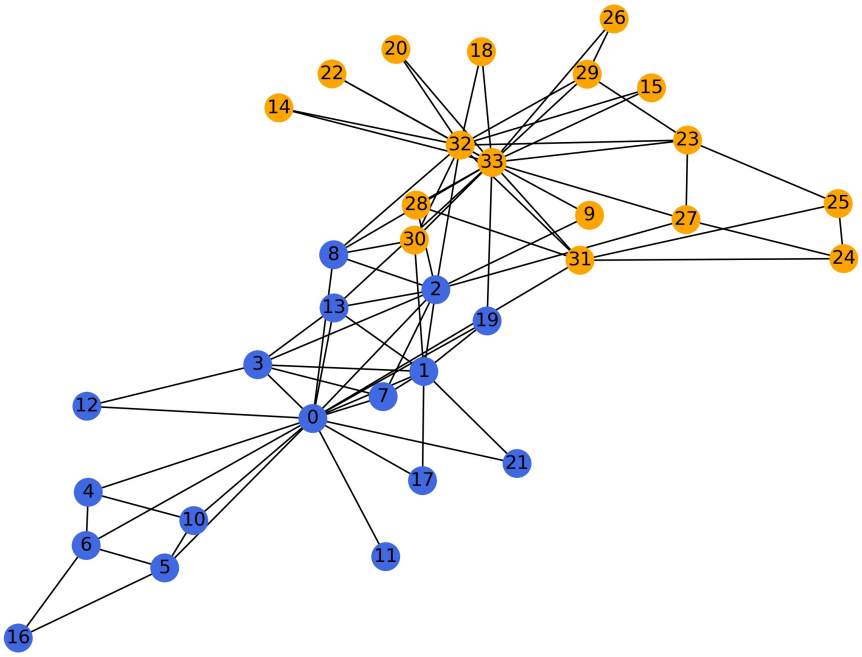

图 3: Zachary's Karate Club

与 (e) 类似, $H^0 = I$, 建立两层 GCN 网络,

$$\begin{aligned} H^1 &= \mathrm{ReLU}(\hat{D}^{-1}\hat{A}H^0W^0) \\ H^2 &= \mathrm{ReLU}(\hat{D}^{-1}\hat{A}H^1W^1) \end{aligned} \tag{17}$$

随机初始化权重矩阵 W^0, W^1 进行计算, 经过数次尝试, 得到 Zachary's Karate Club 的特征表示如图 4 所示, 由图可知即便是随机初始化的权重矩阵也可以将两个类别分开.

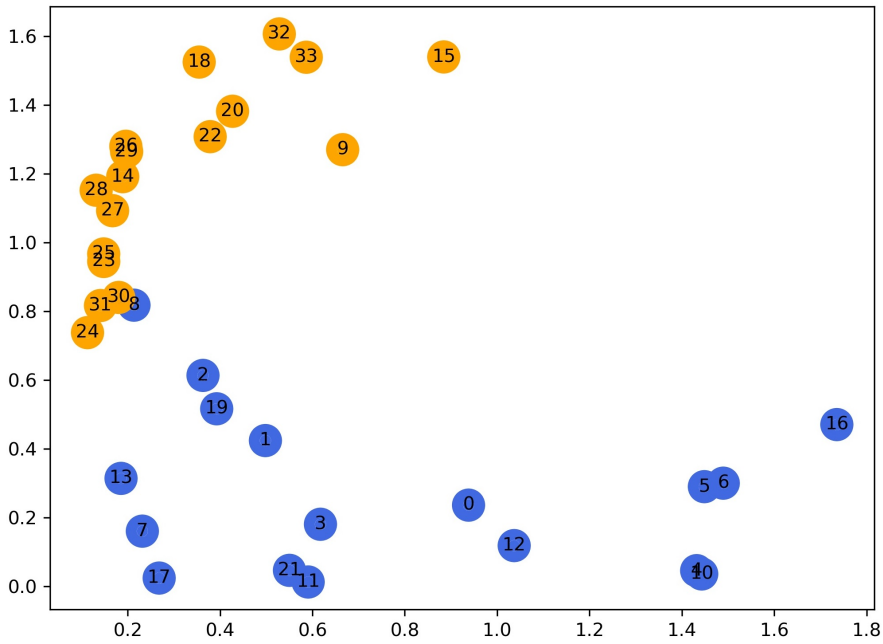

图 4: Feature Representations of Zachary's Karate Club

Programming: Decision Tree

实现决策树算法 C4.5，并且在 *Sogou Corpus* 数据集上测试它的效果（数据集详情见 readme）.

要求：

- 不能调用已有的机器学习包
- 将数据随机分为 3:1:1 的三份，分别为训练集，交叉验证集，测试集。请在训练集上训练，交叉验证集上选择超参数，用选出的最好模型在测试集上给出测试结果。因此，在报告中需要说明算法的超参数有哪些，在不同的超参数设置下训练集和交叉验证集的分类正确率，最好模型的超参数设置，以及最后的测试正确率。
- 请结构化代码，必须包含但不限于如下几个函数（请从代码中分离出来，有明确的这几个函数，函数参数可以有所变化）：
  - **main()**
    要求 main 函数在运行中，逐个测试不同的超参数，然后打印出每个超参数的设置，该设置下的训练、验证正确率（就是上面第二点中提到的要出现在报告中的结果）。
  - **GenerateTree(args)**
    生成树的总代码，*args* 为各种超参数，包括但不限于下面的 *thresh*，或者其他会影响树性能的超参数，自由发挥。

- **SplitNode(samplesUnderThisNode, thresh, ...)**

  对当前节点进行分支，*samplesUnderThisNode* 是当前节点下的样本，*thresh* 是停止分支的阈值，停止分支的条件请在实验报告中说明。

- **SelectFeature(samplesUnderThisNode, ...)**

  对当前节点下的样本，选择待分特征。

- **Impurity(samples)**

  给出样本 *samples* 的不纯度，请在实验报告中说明采用的不纯度度量。

- **Decision(GeneratedTree, XToBePredicted)**

  使用生成的树 *GenerateTree*，对样本 *XToBePredicted* 进行预测。

- **Prune(GeneratedTree, CorssValidationDataset, ...)**

  对生成好的树 *GeneratedTree*（已经过 stopped splitting) 进行剪枝：考虑所有相邻的叶子节点，如果将他们消去可以增加验证集上的正确率，则减去两叶子节点，将他们的共同祖先作为新的叶子节点。或者实现其他的剪枝方法，如有，请在实验报告中说明。

解: 数据集采用均匀分布随机分割为 3:1:1 的三份, 详见函数 `split()`, 分割结果见

`train_split.txt`, `validate_split.txt`, `test_split.txt`.

算法的超参数为分枝不纯度下降的阈值 (threshold), 设置为列表 [0.5, 0.4, 0.3, 0.2, 0.1, 0.09, 0.08, 0.07, 0.06, 0.05, 0.04, 0.03, 0.02, 0.01, 0.005, 0.003, 0.001], 停止分枝的条件为当前节点的数据纯净 (pure), 或者按任何特征进行分割都不能使得不纯度下降大于阈值, 其中不纯度采用熵不纯度 (entropy impurity) 进行度量. 算法采用的剪枝方法为 C4.5 基于规则的算法, 即把生成的决策树转化为等价的规则集合, 剪去任何一个可以使某个规则在验证集上分类正确率上升的前件 (precondition), 直到无法剪除, 最后将规则按分类正确率从高到低排序.

在不同的超参数设置下训练集, 交叉验证集和测试集的分类正确率如表 1 所示, 当阈值 threshold $\leqslant 0.01$ 时, 验证集正确率不再改变, 最佳参数 threshold $= 0.01$ 下最佳模型的测试正确率为 accuracy $= 0.77701$.

训练集正确率, 交叉验证集正确率和测试集正确率与阈值变化关系分别如图 5, 图 6 和图 7 所示, 可见使用 C4.5 的方法, 模型的分类正确率基本随着阈值的减小而逐渐升高, 最后逐渐收敛趋于不变.

表 1: 不同阈值训练集, 交叉验证集和测试集正确率

threshold	train accuracy	validate accuracy	test accuracy
0.5	0.11498	0.10131	0.10942
0.4	0.11498	0.10131	0.10942
0.3	0.11498	0.10131	0.10942
0.2	0.42276	0.41247	0.41828
0.1	0.78746	0.76844	0.73615
0.09	0.80523	0.79221	0.74965
0.08	0.80708	0.79428	0.74931
0.07	0.84251	0.83356	0.77701
0.06	0.84286	0.83425	0.77701
0.05	0.84402	0.83494	0.77909
0.04	0.84634	0.83494	0.77666
0.03	0.84599	0.8346	0.77666
0.02	0.84611	0.83494	0.77666
0.01	0.84541	0.83563	0.77701
0.005	0.84483	0.83563	0.77666
0.003	0.84483	0.83563	0.77666
0.001	0.84483	0.83563	0.77666

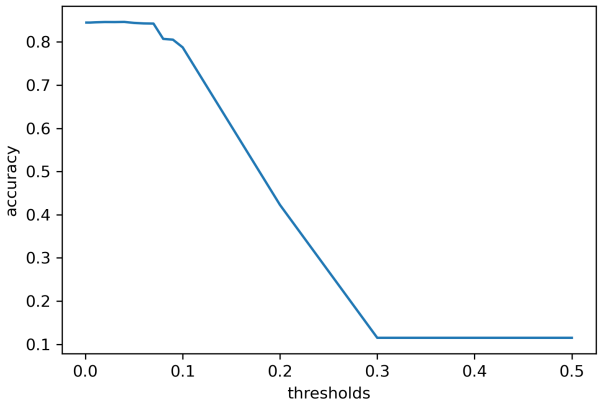

图 5: 训练集正确率与阈值变化关系

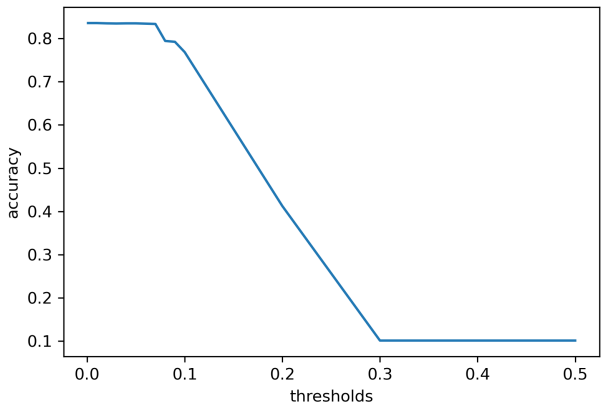

图 6: 交叉验证集正确率与阈值变化关系

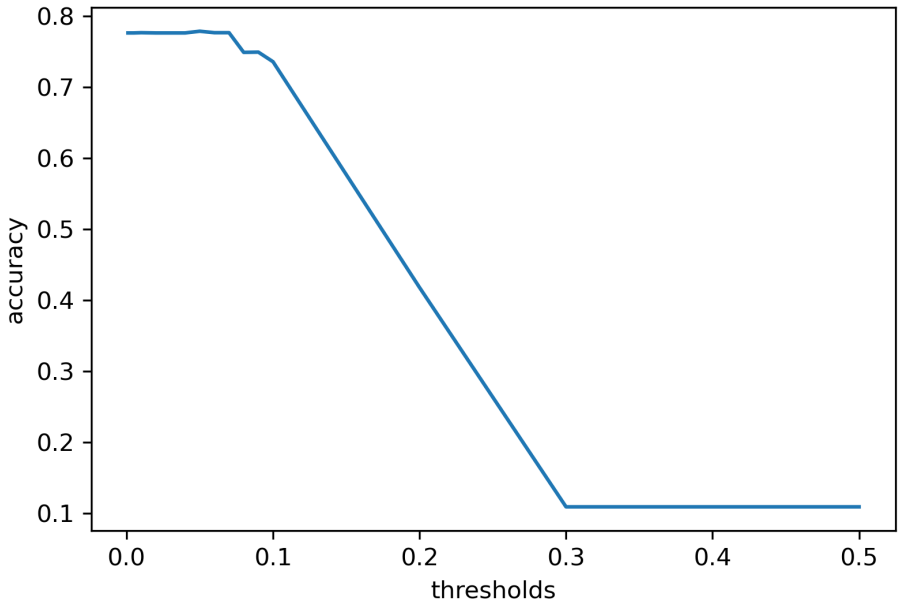

图 7: 测试集正确率与阈值变化关系