CS 240 : Lab 6
Decision Tree Classifiers

TAs: Rounak Dalmia, Vidit Goel

Instructions

- This lab will be **graded**. The weightage of each question is provided in this PDF.
- Please read the problem statement and the submission guidelines carefully.
- All code fragments need to be written within the `TODO` blocks in the given Python files. Do not change any other part of the code.
- **Do not** add any **additional** *print* statements to the final submission since the submission will be evaluated automatically.
- For any doubts or questions, please contact either the TA assigned to your lab group or one of the 2 TAs involved in making the lab.
- The deadline for this lab is **Monday, 19 February, 5 PM**.
- The submissions will be checked for plagiarism, and any form of cheating will be appropriately penalized.

The submissions will be on Gradescope. You need to upload the following Python files: `q1.py` and `q2.py`. In Gradescope, you can directly submit these Python files using the upload option (you can either drag and drop or upload using browse). No need to create a tar or zip file.

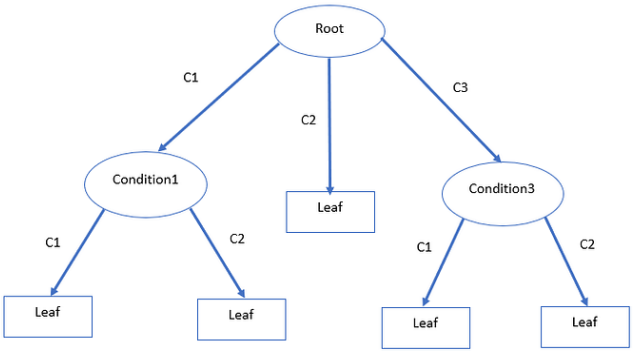
1 Decision Tree Classifier [70 marks]

Decision Tree is a supervised learning technique that can be used for classification and regression problems, but it is mostly preferred for solving classification problems. It is a tree-structured classifier where each internal node represents a feature of a dataset, each branch represents a partial decision on its class, and each leaf node represents the final decision on its class. In the last lab, we talked about perceptron classifiers. In this lab, we will implement a decision tree classifier from scratch and will also have a look on how we can use `sklearn` library to construct a decision tree classifier.

1.1 What are decision tree classifiers and how are they constructed?

Just skim through this section quickly. Make sure you are familiar with the concepts of entropy, information gain and gini index

Decision Trees are a popular and surprisingly effective technique, particularly for **classification** problems. But, the seemingly intuitive interface hides complexities. The criterion for selecting variables and hierarchy can be tricky, and there are a few criteria used in practice, e.g., **Gini index**, **Entropy** (wait, isn't that physics?) and **information gain** (isn't that information theory?).



A decision tree begins with the target variable. This is usually called the **parent node**. The Decision Tree then makes a sequence of splits based in hierarchical order of impact on this target variable. From the analysis perspective the first node is the **root node**, which is the first variable that splits the target variable. To identify the root node we would evaluate the impact of all the variables that we have currently on the target variable to identify the variable that splits the target classes into the most homogenous groups.

Decision trees follow a **top-down**, greedy approach that is known as **recursive binary splitting**. The recursive binary splitting approach is top-down because it begins at the top of the tree and then successfully splits the predictor space. Here is where the true complexity and sophistication of decision lies. **Variables** are selected on a complex statistical criterion which is applied at each decision node. Now, variable selection criterion in Decision Trees can be done via two approaches:

1. Entropy and Information Gain
2. Gini Index

1.1.1 Entropy and Information Gain

In the context of Decision Trees, **entropy** is a measure of disorder or impurity in a node. Thus, a node with more variable composition, such as 2 Pass and 2 Fail would be considered to have higher Entropy than a node which has only pass or only fail. The maximum level of entropy or disorder is given by 1 and minimum entropy is given by a value 0. Leaf nodes which have all instances belonging to a single class would have an entropy of 0. Whereas, the entropy for a node where the classes are divided equally would be 1. Entropy is measured by the formula:

$$\text{Entropy} = -\sum_{z \in \mathcal{Z}} P(Z = z) \log_2 P(Z = z).$$

A Decision Tree determines the root node by calculating the conditional entropy for each feature and its potential splits. To calculate the conditional entropy, we use the formula

$$H(Y|X_i) = \sum_{x_i} P(X_i = x_i) H(Y|X_i = x_i).$$

Hence we consider a potential split based on each feature (denoting $X_i = x_i$), calculate the conditional entropy $H(Y|X_i = x_i)$ that we denote as $\text{Entropy}_{\text{child}}$ and take the weighted sum where weights are $P(X_i = x_i) = \frac{N_{\text{child}}}{N_{\text{parent}}}$, where N_{child} is the number of instances in the child node, and N_{parent} is the number of instances in the parent node. Then the **Information Gain** can be written as follows.

$$\text{Information Gain} = \text{Entropy}_{\text{parent}} - \sum_{\text{children}} \frac{N_{\text{child}}}{N_{\text{parent}}} \times \text{Entropy}_{\text{child}}.$$

1.1.2 Gini Index and Gini gain

Now, let's delve into another criterion used in Decision Trees: **Gini index**. Gini index measures the degree or probability of a particular variable being wrongly classified when it is randomly chosen. It is calculated by summing the probability of each item being chosen times the probability of a mistake in categorizing that item. A node with a Gini index of 0 is pure, meaning all its elements belong to a single class. Gini index is calculated by the formula:

$$\text{Gini index} = 1 - \sum_{z \in Z} (P(Z = z))^2.$$

The **Gini gain**, on the other hand, is a metric to quantify the purity of the split in a dataset. It is calculated by subtracting parent's gini index with that of all the children. The node split that results in the highest Gini gain is selected as the optimal split. The Gini gain for a split is calculated by:

$$\text{Gini gain} = \text{Gini index}_{\text{parent}} - \sum_{\text{children}} \frac{N_{\text{child}}}{N_{\text{parent}}} \times \text{Gini index}_{\text{child}}$$

where N_{child} is the number of instances in the child node, and N_{parent} is the number of instances in the parent node.

1.2 Problem Statement and Dataset

In this problem, you will implement a decision tree classifier from scratch without using any external libraries, except the one already imported in the given file. The problem here is a binary classification problem based on breast cancer data where a particular data instance has to be classified as a cancer-recurrence-event or no-cancer-recurrence-event. You have been provided with 4 datasets, `X_train`, `Y_train`, `X_test`, and `Y_test`.

The decision tree algorithm is as follows:

Algorithm 1 DT Algorithm

function ID3(*Examples*, *Classes*, *Attr_list*)
    Input: *Examples* are training examples.
    Input: *Classes* is the class to be predicted by the tree.
    Input: *Attr_list* is a list of attributes.
    Output: Returns a trained decision tree.
    Create a Root node for the tree
    if all examples are + **then**
        return a single node Root with label +
    end if
    if all examples are − **then**
        return a single node Root with label −
    end if
    if *Attr_list* is empty **then**
        return a single node Root with label = most common value of *Classes* in *Examples*
    end if

    begin
    $A \leftarrow$ the attribute from $Attr_list$ with highest information gain
    for each possible value v of A **do**
        Add a new tree branch below Root, corresponding to $A = v$
        Let *Examples_v* be the subset of *Examples* with value v for A
        if *Examples_v* is empty **then**
            Below this new branch, add a leaf node with label = most common value of *Classes* in *Examples*
        else
            Below this new branch, add the following subtree:
            ID3(*Examples_v*, *Classes*, *Attr_list* $\setminus \{A\}$)
        end if
    end for
    end
    return Root
end function

1.3 Tasks to be completed

You have been provided with four datasets, `X_train`, `Y_train`, `X_test`, and `Y_test`. The dataset is categorical, so you will be splitting for each unique value. In the file `q1.py`, you are expected to:

- Task 1: Complete the `entropy` function using the formula discussed in class or as mentioned in the theory section above. Do not use for loops. [10 marks]
- Task 2: Complete the `gini_index` function using the formula described in the theory section above. Do not use for loops. [10 marks]
- Task 3: Complete the `information_gain` function. You can refer to the class notes or the theory section above. You will only get marks for this part if your `entropy` function is correct, which is obvious. [10 marks]
- Task 4: Complete the `gini_gain` function. You can refer to the theory section above. This function is very similar to `information_gain`. You will only get marks for this part if your `gini_index` function is correct, which is obvious. [10 marks]‘

- Task 5: Complete the `__build_tree__` function in the `DecisionTree` class. Base cases are already implemented. You have to complete the recursive part. [30 marks]
- Comments have been provided at each step to guide you further.

To run the file, simply use the command line argument: `python3 -W ignore q1.py`
Make sure that the variables returned by these functions (and others in general) follow the prescribed format.

2 Optimal decision tree using `sklearn` library [30 marks]

In our initial problem, we constructed a decision tree from scratch and applied it to solve our classification task. One of the objectives in this question is to utilize the `sklearn DecisionTreeClassifier` to address our classification requirements. Additionally, we aim to tackle the challenge of overfitting commonly encountered in decision tree classifiers.

Overfitting poses a significant concern with Decision Trees. The algorithm recursively splits the dataset until it reaches leaf nodes, potentially resulting in leaf nodes containing very few instances. Consequently, this could lead to the formation of a deep tree structure that may not generalize well to unseen test data. Each leaf node represents a specific combination of attributes observed in the training dataset, limiting the tree's ability to classify attribute combinations absent in the training data. To mitigate this, various techniques can be employed, such as controlling the maximum depth, minimum sample size of leaf nodes, among others. The `sklearn` library offers comprehensive control over different hyperparameters, allowing for the regularization of learned decision trees.

We encourage you to explore the documentation of `sklearn.tree.DecisionTreeClassifier`, accessible via the following hyperlink: sklearn.tree.DecisionTreeClassifier. Familiarize yourself with the utilization of various hyperparameters and how they contribute to the regularization of decision trees. In this lab, we will primarily focus on three key hyperparameters: `max_depth`, `min_samples_split`, and `min_samples_leaf`.

2.1 Code Tasks and Functions

In the file `q2.py`, you are expected to:

- Task 1: Complete the function `train_model` inside the `SKLearnDecisionTree` class. The function should return the train and test accuracies as a dictionary. Also, the data to be passed to the function for model training should be the data frame already available in `q2.py`; do not pass data in a numpy array or use any other format. [10 marks]
- Task 2: Complete the `train_model_regularised` function in the `SkLearnDecisionTree` class. The aim here is to identify the optimal decision tree model after regularization. You are permitted to adjust three hyperparameters: `max_depth`, `min_samples_split`, and `min_samples_leaf`.

  You will vary the `max_depth` parameter from 1 to 10, `min_samples_split` from 3 to 30 (with steps of 3, i.e., 3, 6, 9, ..., 30), and `min_samples_leaf` from 1 to 10 (with steps of 1, i.e., 1, 2, 3, ..., 10). Combining these three parameters will result in a distinct decision tree model. Consequently, there will be a total of 1000 combinations, yielding 1000 different decision tree models.

  Your task is to generate and return a list containing the training accuracies and test accuracies for all 1000 models. You'll start by varying the `min_samples_leaf`, followed by `min_samples_split`, and finally `max_depth`. This means that for indices 0-9, you will have `max_depth`=1, `min_samples_split`=3, and `min_samples_leaf`=1,2,3,...,10. Similarly, for indices 10-19, you will have `max_depth`=1, `min_samples_split`=6, and `min_samples_leaf`=1,2,3,...,10. This pattern continues until index 999, resulting in 1000 models. Your function should return a dictionary of the form as described in the code file. [20 marks]
- Task 3 (**UNGRADED**): Once you have the above functions implemented, you can have a look at the plot `analysis.png`. Also, in the main function, we have found the best decision tree using the highest test accuracies. Observe how you improved the test accuracy, by making a few changes to your decision tree. We have also provided you with a `plot_tree` function using which you can construct the `self.tree` and save it as a plot.
- Comments have been provided at each step to guide you further.

To run the file, simply use the command line argument: `python3 -W ignore q2.py`