CmpE 462 : Machine Learning

Assignment 3

Emre Girgin - 2016400099

Part 1

In this part of the assignment, we are asked to implement a decision tree that classifies the Iris dataset. We only used the “Iris-Setosa” and “Iris-versicolor” classes.

Decision tree is a classification algorithm that divides the set of samples into subgroups recursively. At each node, for every feature it tries to find a decision boundary that maximizes **the information gain**. Then divides the sample set based on that decision boundary into two parts. If the data is separable based on a feature in case there are more than two classes, the algorithm is still applicable since we re-use the features used before. **Figure1** shows an example.

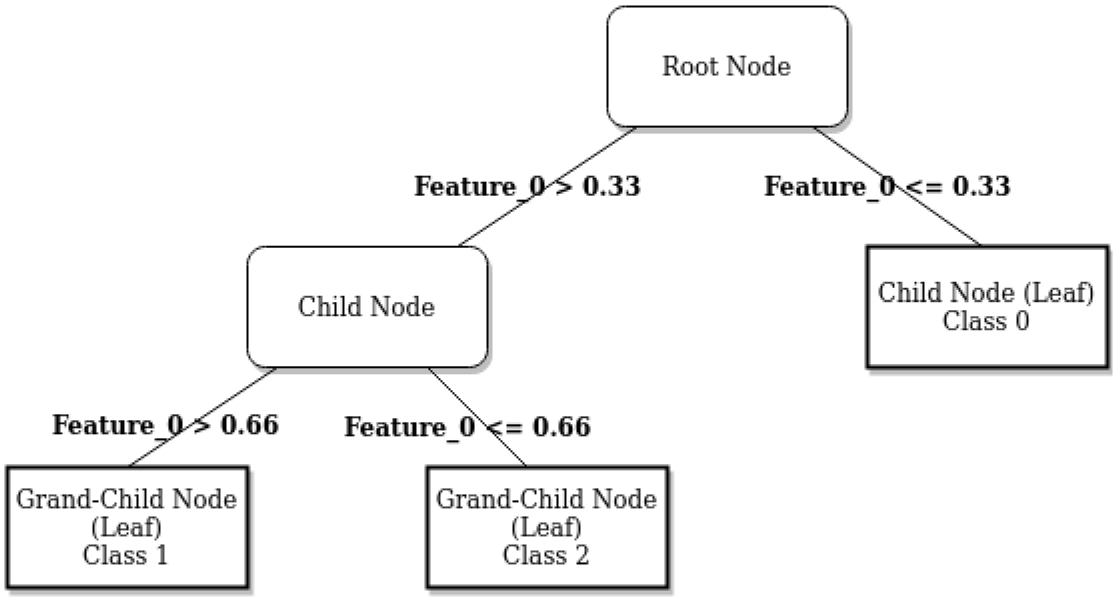

Root Node
Feature_0 > 0.33
Feature_0 <= 0.33
Child Node
Child Node (Leaf) Class 0
Feature_0 > 0.66
Feature_0 <= 0.66
Grand-Child Node (Leaf) Class 1
Grand-Child Node (Leaf) Class 2

Figure 1 When there are three classes, the data is still separable based on a feature only dividing into two subgroups. In the figure, the data is divided into three classes based on the 0th feature.

When the features are not categorical (which is the case in Iris Dataset) a Decision Tree follows the recursive algorithm below:

```
function divide_node(subset):
    1. Take a subset of samples from the parent node. (Root node takes all the trainset)
    2. prev_entropy = calculate_entropy(subset)
    3. current_info_gain = 0
    4. for each feature in feature_list:
        a. separator = find_seperator(feature, subset)
        b. right_subset, left_subset = divide_subgroup(separator, feature, subset)
        c. right_entropy = calculate_entropy(right_subset)
        d. left_entropy = calculate_entropy(left_subset)
        e. weighted_entropy = calculate_weighted(right_entropy, left_entropy)
        f. if prev_entropy - weighted_entropy > current_info_gain:
            i. current_info_gain = prev_entropy - weighted_entropy
            ii. last_left = left_subset
            iii. last_right = right_subset
    5. divide_node(last_left)
    6. divide_node(last_right)
```

The algorithm above consists of several sub-procedures. I'll go over them one by one

Entropy

Entropy is a measure of impureness. In our context, a node is pure when its samples belong to the same class, where the entropy is zero. If not, we calculate the entropy using the following formula:

$$entropy(node) \approx -\frac{p}{p+n}log_2(\frac{p}{p+n}) - \frac{n}{p+n}log_2(\frac{n}{p+n})$$

where $node$ is the subset and p is the number of samples belonging to class 0 and n is the number of samples belonging to class 1 and $n + p = size(subset)$.

Weighted Entropy

Weighted entropy is the weighted sum of entropies of the child nodes.

$$weighted(nodes) = \sum_{i=1}^{nodes} \frac{size(node_i)}{size(nodes)} \times entropy(node_i)$$

Where nodes is the list of children of a node.

Finding Decision Boundary (Separator)

We need to find a decision boundary for a given feature column of a dataset. The subsets of the child nodes are decided based on this threshold. There are various ways to find this point. For example a classifier can be trained such as logistic regression or SVM. However, in our case, since the Iris dataset is a very simple dataset, I directly used the mean of the feature column. Besides the mean is a powerful natural unbiased estimator when the model weights are all one. $\mathbf{1}X = avg(X)$, where $\mathbf{1}$ is a vector of ones.

Information Gain and Gain Ratio

Information gain is defined as $Info - Gain = ParentEntropy - WeightedEntropy$. At each step, the decision tree algorithm tries to find the best feature and decision boundary that maximizes information gain by default, which is the same as minimizing weighted entropy.

On the other hand, this technique exploits unique identifiers and hurts generalization. Thus another technique called gain ratio can also be used. Maximizing gain ratio also forms a more robust classifier.

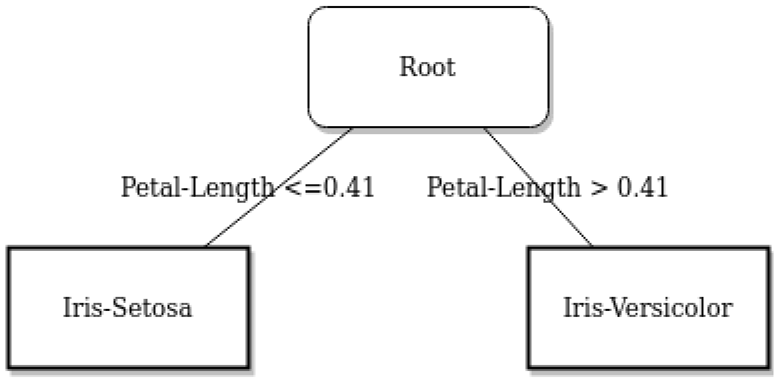
$$Gain - Ratio = \frac{Info - Gain}{-\sum_{i=1}^{nodes} \frac{size(node_i)}{size(nodes)} log_2(\frac{size(node_i)}{size(nodes)})}$$

In step 2 of part 1, gain-ratio is used instead of information gain.

Note that the dataset is **normalized** before the train-test split.

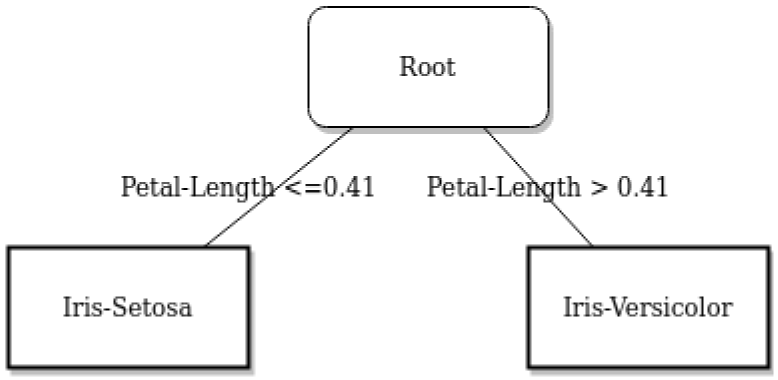
Step 1

The decision tree that was found using information gain.



Step 2

The decision tree that was found using the gain ratio.



Since Iris-Dataset is a relatively simple dataset and does not contain unique identifiers, the resulting decision trees of the both steps are the same. Also not that the decision boundary is for the **normalized** features (MinMax Normalization).

Part 2

SVM is a classifier that is used in nonlinear datasets frequently. It defines a safety region called **margin** around the classifying hyper-plane. The closest data points to the hyper-plane called **support vectors**. If any of the data points are allowed inside the margin, then it's called **hard-margin SVM**, otherwise, **soft-margin SVM**. We can control the severity of the margin rule by a variable called **C**. As the C gets bigger the margin becomes more strict and closer to the hard-margin SVM. Soft-margin SVM is obtained in the dual formulation of it. In dual formulation, we can also transform the data points into another space (called z) from the input space to make them linearly separable. This is called **kernel trick** and the transformation function is the **kernel**.

$$\max_{\alpha \in \mathbb{R}^N} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m=1}^{N} \sum_{n=1}^{N} \alpha_n \alpha_m y_n y_m \mathbf{\Phi}(\mathbf{x_n})^{\mathbf{T}} \mathbf{\Phi}(\mathbf{x_m})$$

$$subjectto \sum_{n=1}^{N} y_n \alpha_n = 0, 0 \leq \alpha_n \leq C, \forall n$$

Dual soft-margin SVM optimization, where Φ is the kernel function and C is the margin parameter.

Note that the dataset is **normalized** before the train-test split. If we remove the normalization, the accuracy and number of support vectors change, and get worse.

Step 1

SVM kernel=1 C=1 acc=0.78 n=346

SVM kernel=1 C=5 acc=0.89 n=310

SVM kernel=1 C=10 acc=0.96 n=266

SVM kernel=1 C=50 acc=0.98 n=172

SVM kernel=1 C=100 acc=0.98 n=139

When a Polynomial kernel is used (kernel=1) the effect of **C** is more visible. Since larger C implies more strict classification, it contains less misclassified samples in it. On the other hand, a larger margin (smaller C) touches more data points so those hyperplanes have more support vectors.

Step 2

```
SVM kernel=0 C=1 acc=0.97 n=73
SVM kernel=1 C=1 acc=0.78 n=346
SVM kernel=2 C=1 acc=0.98 n=174
SVM kernel=3 C=1 acc=0.98 n=214
```

The transformed data points in the Z space creates different patterns based on the kernels (linear, polynomial, RBF, Sigmoid) used. For example, in the experiment, when linear, RBF, or Sigmoid transform is applied, the data in the Z-space becomes separable easily. However, the Polynomial Kernel can not classify them correctly when the C is one. Also, kernel 0, 2 and 3 classify with less number of support vectors compared to kernel 1. The reason behind this is the polynomial kernel needs a larger C value to correctly classify most of the data points. (See part2 step1) There are a lot of misclassified data points inside the margin since it is relatively larger when C is one.