Implement Decision Tree with Python

Yueqi Li
yueqi.li@uky.edu

February 19, 2022

I. Introduction

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. It is a hierarchical model for supervised learning whereby the local region is identified in a sequence of recursive split in a smaller number of steps. A decision tree is composed of internal decision nodes and terminal leaves.

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.

II. Methods

In this project, we have four synthetics data, each of data set have two feature and binary label. We also have one Pokemon's Legendary test file, which contain 45 feature, it also has binary label. We want to implement tree to classify the data.

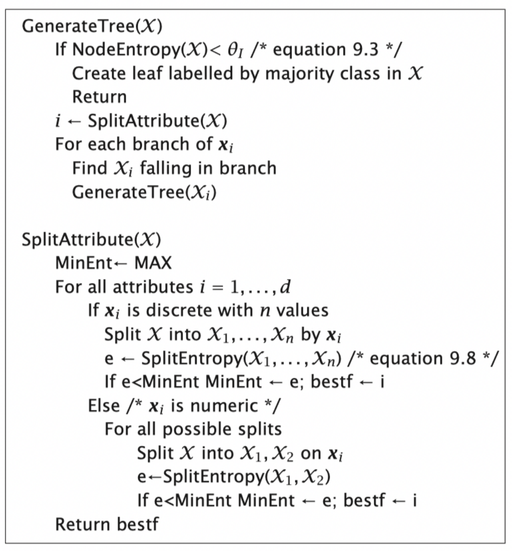
Before Implement Tree, the first thing we do is discretize data, in other words, we split data to different bins, we use equal distance for the splitting.Then we implement decision tree for classification, namely, a *classification tree.*

In the classification tree, we use impurity measure to qualify the split. We choose to use *entropy*(Quinlan 1986) function to measure the impurity and decide how to split. Let us say at node m, N_{mj} of N_m tale brach j; these are x^t for which the test $f_m(x^t)$ returns outcome j . For a discrete attribute with n values, there are n outcomes, and for a numeric attribute, there are two outcomes ($n = 2$), in either case satisfying $\sum_{j=1}^{n} N_{mj} = N_{mj}$. N_{mj}^i of N_{mj} belong to class C_i : $\sum_{j=1}^{k} N_{mj}^i = N_{mj}$. Similarly,$\sum_{j=1}^{n} N_{mj}^i = N_m^i$.

Then given that at node m, the test returns outcome j , the estimate for the probability of class C_i is $\hat{p}(c_i|x, m, j) \equiv p_{mj}^i = \frac{N_{mj}^i}{N_{mj}}$ and total impurity after the split is given as:

$$\mathcal{I}' = -\sum_{j=1}^{n} \frac{N_{mj}}{N_{mj}} \sum_{j=1}^{K} p_{mj}^i \log_2 p_{mj}^i \tag{1}$$

So for discreet attributes we calculate the impurity and choose the one that has minimum entropy. We build the tree based on the algorithm provide by the textbook, Since we have limited at most two layer, so we skip the first step: calculate the Node Entropy. Also, for our project, we prepossessed data to make it discrete, therefore, we also skip the last part when we implement. We use (1) to calculate the split entropy.



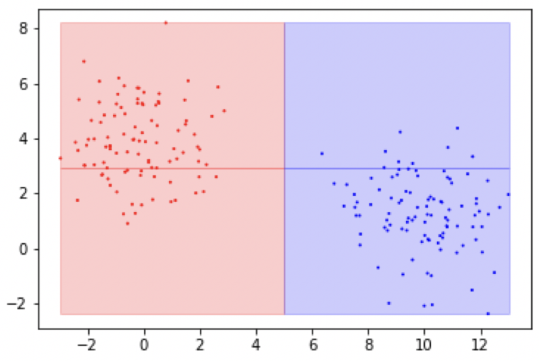
```
GenerateTree($\mathcal{X}$)
    If NodeEntropy($\mathcal{X}$)< $\theta_I$ /* equation 9.3 */
        Create leaf labelled by majority class in $\mathcal{X}$
        Return
    $i$ ← SplitAttribute($\mathcal{X}$)
    For each branch of $\boldsymbol{x}_i$
        Find $\mathcal{X}_i$ falling in branch
        GenerateTree($\mathcal{X}_i$)

SplitAttribute($\mathcal{X}$)
    MinEnt← MAX
    For all attributes $i = 1, \dots, d$
        If $\boldsymbol{x}_i$ is discrete with $n$ values
            Split $\mathcal{X}$ into $\mathcal{X}_1, \dots, \mathcal{X}_n$ by $\boldsymbol{x}_i$
            e ← SplitEntropy($\mathcal{X}_1, \dots, \mathcal{X}_n$) /* equation 9.8 */
            If e<MinEnt MinEnt ← e; bestf ← i
        Else /* $\boldsymbol{x}_i$ is numeric */
            For all possible splits
                Split $\mathcal{X}$ into $\mathcal{X}_1, \mathcal{X}_2$ on $\boldsymbol{x}_i$
                e←SplitEntropy($\mathcal{X}_1, \mathcal{X}_2$)
                If e<MinEnt MinEnt ← e; bestf ← i
    Return bestf
```

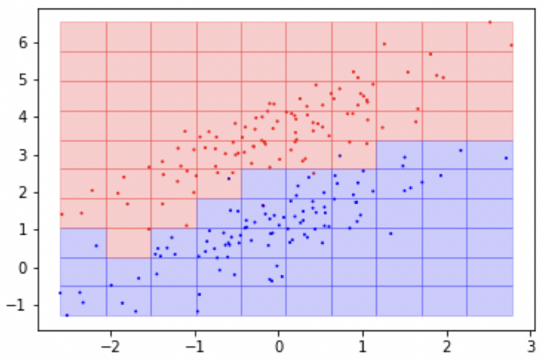
Figure 9.3 Classification tree construction.

III. Result

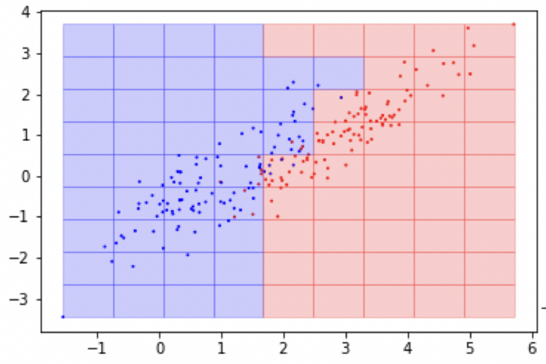
In our experience, we split data from 2 to 10 bins and test the accuracy rate based on confusion matrix,and we choose the best one from those 8 split based on accuracy rate. Here is result: For synthetic-1, when we split it to 2 bin,we already reach 100% accuracy; For synthetic-2, when we split it to 10 bin, we reach 98% accuracy.For synthetic-3, when we split it to 9 bin, we reach 91.5% accuracy; For synthetic-4, when we split it to 10 bin, we reach 97% accuracy



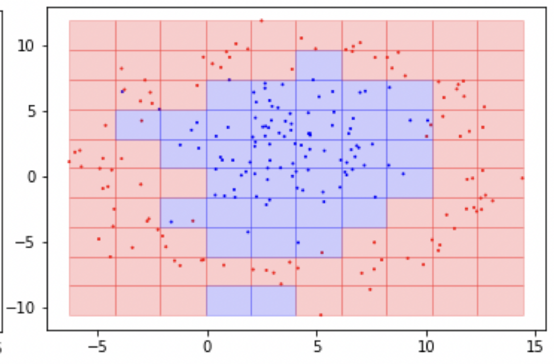
synthetic-1: 100% accuracy



Synthetic-2: 98% accuracy



Synthetic-3: 91.5% accuracy



Synthetic-4: 97% accuracy

For the pokemon legendery classification dataset, if we split it to 10 bins, we reach 97.01% accuracy.