CMSC5724 Project Report-Project I

1155164822 FU Junchen
1155166098 ZHANG Weidong

Abstract

In this report, we implement Hunt's algorithm for decision tree classification from the lecture with two types of "best split" metrics from scratch in Python, namely GINI Index and Entropy. To improve the training performance, we adopt the optimization procedure from exercise list 1 problem 3. After this optimization, the calculation time complexity of ordered features is reduced to O(nlogn). The whole development procedure contains three phases. In the first phase, we develop a naive edition of a decision tree with Numpy. The total training latency on my laptop is around 700 seconds. After that, the second edition is to apply the optimization algorithm for the ordered attributes. The training time drops sharply to only around 5 seconds with the same parameters. In the final procedure, we implement the Numpy functions by ourselves. The code of our project is achieved with only the standard library from Python, namely Counter, random, math, and time.

1. The decision tree built from the Adult training set

1.1 Hunt's Algorithm

We adopt Hunt's algorithm from the first lecture to construct our classification model based on the training set. The algorithm description for our experiment is,

Algorithm Hunt(S)
/* $\boldsymbol{S}$ is the training set; the function returns the root of a decision tree */
1. if all the objects in $\boldsymbol{S}$ belong to the same class
2. return a leaf node with the value of this class
3. if (all the objects in $\boldsymbol{S}$ have the same attribute values) or ($|\boldsymbol{S}|$is too small) or (depth of tree is too large)
4. return a leaf node whose class value is the majority one in S
5. find the "best" split attribute $\boldsymbol{A}^*$and predicate $\boldsymbol{P}^*$
6. $\boldsymbol{S_1}$ = the set of objects in $\boldsymbol{R}$ satisfying $\boldsymbol{P}^*$; $\boldsymbol{S_2} = \boldsymbol{S} \backslash \boldsymbol{S_1}$
7. $\boldsymbol{u_1} = \boldsymbol{Hunt}(\boldsymbol{R_1})$; $\boldsymbol{u_2} = \boldsymbol{Hunt}(\boldsymbol{R_2})$
8. create a root $\boldsymbol{u}$ with left child $\boldsymbol{u_1}$ and right child $\boldsymbol{u_2}$
9. set $\boldsymbol{A_u} = \boldsymbol{A}^*$ and $\boldsymbol{P_u} = \boldsymbol{P}^*$
10. return $\boldsymbol{u}$

Compared to the algorithm from the lecture note, we add the condition of the depth of the decision tree for preventing overfitting and reducing training time.

1.2 "Best" Split Metrics and Optimization

In our experiment, we implement and optimize two metrics to identify the "best" split, namely GINI index and Entropy.

The GINI index of the two splits is defined as,

$$n = |S|$$
$$n_y = number\ of\ objects\ in\ S\ with\ label\ yes$$
$$p_y = \frac{n_y}{n}$$
$$p_n = 1 - p_y$$
$$GINI(S) = 1 - (p_y^2 + p_n^2)$$
$$GINI_{split} = \frac{|S_1|}{|S|} GINI(S_1) + \frac{|S_2|}{|S|} GINI(S_2)$$

The smaller $GINI_{split}$ is, the better the split quality. The Entropy of the splits is similar to the GINI index, which is defined as,

$$Entropy(S) = -(p_y log(p_y) + p_n log(p_n))$$
$$Entropy_{split} = \frac{|S_1|}{|S|} Entropy(S_1) + \frac{|S_2|}{|S|} Entropy(S_2)$$

In the training procedure, we normally calculate the difference between the entropy of the parent dataset and the entropy after the split. From that, which is called the information gain. The bigger the information gain is, the better the split quality.

The naïve way to implement our metrics on ordered features for the "best" split calculation is to compute every unique value's GINI and information gain, which is time-consuming. The two metrics share the similarity of calculating the p_y, p_n, $|S|$, $|S_1|$ and $|S_2|$. Therefore, we only need to apply an optimized method to calculate these values, the training latency can be improved.

We adopt the solution method from exercise list 1 to make the time complexity O(nlogn), which is shown below,

Answer. Let S be the set of records in the table. Each record r is in the form (r_A, r_B), representing its values on A and B, respectively. For simplicity, we will assume that all records have distinct values on A. Removing this assumption requires only minor modification of our algorithm, which is left for you to figure out.

Recall that a candidate split is at the value $a = r_A$ of some record $r \in S$. The split at a divides S into: (i) S_1 which includes all the records $r' \in S$ satisfying $r'_A \leq a$, and (ii) S_2 with records $r' \in S$ satisfying $r'_A > a$. To calculate the Gini value of the split, we need to obtain 4 counts:

- The number of yes-records (i.e., yes on B) in S_1—denote this as $c_y^1(a)$;
- The number of no-records in S_1;
- The number of yes-records in S_2;
- The number of no-records in S_2.

Overall, we need to obtain $4(n-1)$ counts (there are $n-1$ candidate splits on A). We will show that all these counts can be obtained in $O(n \log n)$ total time, after which one can easily compute the Gini value of each split in $O(n)$ time.

Due to symmetry, it suffices to explain how to obtain $c_y^1(a)$ for all possible a. Sort S in ascending order by A. Set a count $c = 0$. Scan the records of S in ascending order of A. For each record r, (i) increase c by 1 if $r.B$ is yes, or otherwise, do nothing to c, and then (ii) set $c_y^1(a)$ to c where $a = r.A$.

Because the values are duplicated, we should extend this algorithm. Instead of adding by 1, we should record the numbers of each value with and label, and add these two values respectively.

The function I achieved is the "*_optimized_gini_table_continuous*" and "*_optimized_entropy_table_continuous*" functions from the "*DT_from_scratch.py*" file. Since the code is too long, please check the code for detailed implementation.

1.3 Node structure

The definition of a node in decision tree in our algorithm is Node(self, feature=None, threshold=None, left=None, right=None,*,value=None). The meaning of parameters is shown as follows:

feature: the feature index used to classify
threshold: the threshold for the feature to split the dataset. "==" for the discrete feature and "<=" for the continuous feature.
X: the training data for the current node
y: the label for the training data for the current node
left: the node of the left child
right: the node of the right child
value: if the node is a leaf node, this is the value for the leaf node.
is_leaf: to identify if the node is a leaf node or not.

```
class Node:
    # need to store the best split feature, and threshold. also the left and right child tree,
    # at leaf node we need to store the most common labels

    def __init__(self, feature=None, threshold=None,left=None,right=None,*,value=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value

    def is_leaf(self):
        return self.value is not None
```

Fig.1: The structure of tree node

1.3 Growing Decision Tree

Figure 2 and Figure 3 show the decision tree class and function to grow the decision tree for the training set.

```
# n_feats -> subset of number of features
class DecisionTree:
    def __init__(self,min_samples_split=2,max_depth=10,n_feats=None,criterion="gini"):
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth
        self.criterion = criterion
        self.n_feats = n_feats
        # later need to know the root
        self.root = None
```

Fig. 2: Definition of decision tree class

The parameters in the decision tree mainly include the stopping criteria, such as the number of samples in the split and depth. The "n_feats" is the number of features we use in our training set. The growing procedure is mainly based on Hunt's algorithm as described in Section 1.1. All functions are implemented in the class "*DecisionTree*".

```
def _grow_tree(self,X,y,depth = 0):
    n_samples, n_features = len(X),len(X[0])
    n_labels = len(set(y))

    # stopping criteria
    if (depth >= self.max_depth
        or n_labels == 1
        or n_samples < self.min_samples_split
        or getLenOfUniqueRows(X) <= 1):
        leaf_value = self._most_common_label(y)
        return Node(value=leaf_value)

    # select random features, the returned value should be a list
    feat_idxs = choice(n_features)
    # greedy search to find the best split
    best_feat, best_thresh = self._best_criteria(X,y,feat_idxs)

    left_idxs, right_idxs = self._split(best_feat, getOneColumn(X,best_feat), best_thresh)
    left = self._grow_tree(getRows(X,left_idxs), getRows(y,left_idxs), depth+1)
    right = self._grow_tree(getRows(X,right_idxs), getRows(y,right_idxs), depth+1)

    return Node(best_feat, best_thresh, left, right)
```

Fig. 3: The method for growing a decision tree

2. Experimental Result

We run our experiments on Windows 10 with Intel Core i7-7700HQ CPU and 16GB memory. In this experiment, we mainly change 2 parameters to modify the decision tree model to get better test accuracy. These parameters are the minimal number of samples for a leaf node and the max depth of the decision tree respectively. For every set of parameters, the source code will be run to generate a decision tree according to the GINI index or Information Entropy. Table 1 and 2 reflect the results using entropy while Table 3 and 4 reflect the results using GINI. Specified details are shown in the following tables. The unit for the time is a "second".

Table 1 Results of decision tree with maximum depth 10 (Entropy)

#Depth	**10**	**10**	**10**	**10**	**10**	**10**	**10**
#Min_num	5	10	15	20	25	30	35
Train time	11.46	9.80	9.59	10.05	9.68	9.87	9.90
Test time	0.08	0.11	0.08	0.09	0.10	0.08	0.08
Accuracy	85.60%	85.62%	85.61%	85.58%	85.64%	85.58%	85.58%

Table 2 Results of decision tree with minimum number of split 10 (Entropy)

#Depth	5	10	15	20	25	30	35
#Min_num	**10**	**10**	**10**	**10**	**10**	**10**	**10**
Train time	6.04	10.23	12.34	14.99	14.33	20.99	19.98
Test time	0.05	0.08	0.15	0.12	0.16	0.16	0.18
Accuracy	84.08%	85.61%	84.02%	82.94%	81.90%	81.90%	81.62%

Table 3 Results of decision tree with maximum depth 10 (Gini)

#Depth	**10**	**10**	**10**	**10**	**10**	**10**	**10**
#Min_num	5	10	15	20	25	30	35
Train time	8.74	8.89	10.42	9.67	9.26	9.29	9.34
Test time	0.08	0.09	0.10	0.10	0.08	0.09	0.09
Accuracy	85.50%	85.53%	85.57%	85.58%	85.61%	85.54%	85.52%

Table 4 Results of decision tree with minimum number of split 10 (Gini)

#Depth	5	10	15	20	25	30	35
#Min_num	**10**	**10**	**10**	**10**	**10**	**10**	**10**
Train time	5.78	9.17	13.45	14.37	13.70	15.37	15.46
Test time	0.05	0.08	0.13	0.19	0.43	0.16	0.22
Accuracy	84.06%	85.54%	84.08%	82.46%	81.67%	81.48%	81.18%

The upper tables show trends of how accuracy changes following the ascending number of maximum tree depth or the minimal number of samples of the leaf node. “#Depth” and “#Min_num” represent the number of max depths of the decision tree and the minimal number of samples when spilled into a single leaf node respectively. All results are reserved to two decimal places, following the rounding principle.

From these four tables, we can get some conclusions. There is only a slight difference of accuracy between using Entropy or GINI as a split method when each of the two independent variables is equal. From Table 1 and 3, the accuracy is almost unchanged with the increase of the minimal number of samples and fixed depth. Because the amount of training samples is large enough, which is more than thirty thousand, and the depth of the tree is fixed to 10, the minimal number of samples at the leaf node is not the main impact factor of accuracy. From

Table 2 and 4, the accuracy significantly decreases with the depth of decision rise. The main reason for this phenomenon may be the model is trained to be over-fitting.

As a result, we pick 10 as the depth of the decision, 10 as the minimal number of samples at a leaf node, and Entropy as its split method when training. A screenshot of the training and test results is shown in Figure 4 as followed. The accuracy of this decision tree is 85.64%, which is almost the best result we can get.

```
PS E:\OneDrive - The Chinese University of Hong Kong\Desktop\Data Min Project\CMSC 5724> & E:/python/python.exe "e:/OneDrive - The Chinese Unive
rsity of Hong Kong/Desktop/Data Min Project/CMSC 5724/Modeling_scratch.py"
training latency:9.677152395248413
test latency:0.09775185585021973
[INFO] fail to predict num:  0
[INFO] accuracy:  0.8563745019920319
```

Fig. 4 Decision tree training and test result

We also make this decision tree visualized, printing out decision tree from left to right. The following Figure 5 shows the general structure

Fig 5 Decision tree general structure visualization

Figure 6 shows the details for each node, "->" means this is a node. The following two strings show the splitting feature and threshold of this node. The last string "level" represents the current node's layer, which also represents the depth of it. Level 0 means this node is the root node.

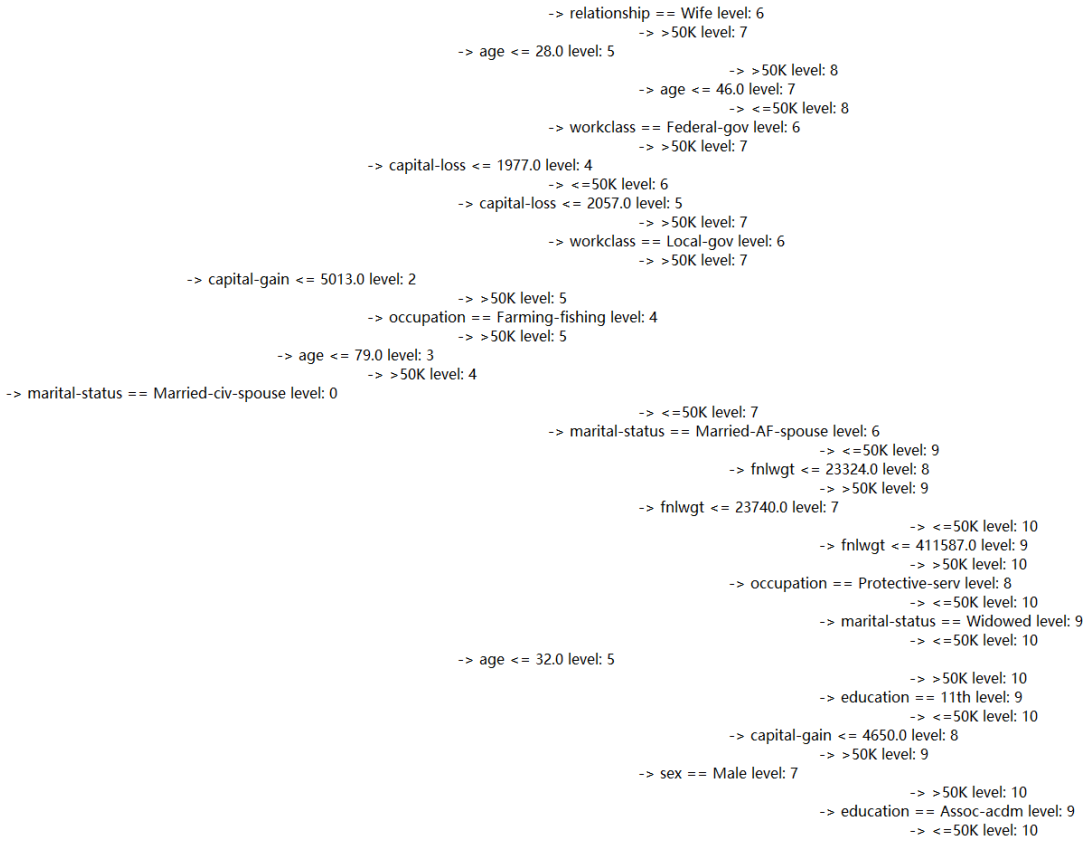

Fig. 6 Decision tree details visualization

3. Appendix

The list of records is too long and it is not elegant to directly put them in the report. Therefore, the detail can be checked in the "*log.txt*", each line here represents one sample from the evaluation dataset, which includes the index of the sample, features, prediction result, actual result, and if the prediction is true or false. The sample is shown below,

```
================================================================================================================
==========================================THE RESULT FOR EACH SAMPLE============================================
================================================================================================================

The 1-th sample [25.0, 'Private', 226802.0, '11th', 7.0, 'Never-married', 'Machine-op-inspct', 'Own-child', 'Black', 'Male', 0.0, 0.0, 40.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
The 2-th sample [38.0, 'Private', 89814.0, 'HS-grad', 9.0, 'Married-civ-spouse', 'Farming-fishing', 'Husband', 'White', 'Male', 0.0, 0.0, 50.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
The 3-th sample [28.0, 'Local-gov', 336951.0, 'Assoc-acdm', 12.0, 'Married-civ-spouse', 'Protective-serv', 'Husband', 'White', 'Male', 0.0, 0.0, 40.0], the predicted result is <=50K, the actual result is >50K, the prediction is False
The 4-th sample [44.0, 'Private', 160323.0, 'Some-college', 10.0, 'Married-civ-spouse', 'Machine-op-inspct', 'Husband', 'Black', 'Male', 7688.0, 0.0, 40.0], the predicted result is >50K, the actual result is >50K, the prediction is True
The 5-th sample [34.0, 'Private', 198693.0, '10th', 6.0, 'Never-married', 'Other-service', 'Not-in-family', 'White', 'Male', 0.0, 0.0, 30.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
The 6-th sample [63.0, 'Self-emp-not-inc', 104626.0, 'Prof-school', 15.0, 'Married-civ-spouse', 'Prof-specialty', 'Husband', 'White', 'Male', 3103.0, 0.0, 32.0], the predicted result is >50K, the actual result is >50K, the prediction is True
The 7-th sample [24.0, 'Private', 369667.0, 'Some-college', 10.0, 'Never-married', 'Other-service', 'Unmarried', 'White', 'Female', 0.0, 0.0, 40.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
The 8-th sample [55.0, 'Private', 104996.0, '7th-8th', 4.0, 'Married-civ-spouse', 'Craft-repair', 'Husband', 'White', 'Male', 0.0, 0.0, 10.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
The 9-th sample [65.0, 'Private', 184454.0, 'HS-grad', 9.0, 'Married-civ-spouse', 'Machine-op-inspct', 'Husband', 'White', 'Male', 6418.0, 0.0, 40.0], the predicted result is >50K, the actual result is >50K, the prediction is True
The 10-th sample [36.0, 'Federal-gov', 212465.0, 'Bachelors', 13.0, 'Married-civ-spouse', 'Adm-clerical', 'Husband', 'White', 'Male', 0.0, 0.0, 40.0], the predicted result is >50K, the actual result is <=50K, the prediction is False
The 11-th sample [26.0, 'Private', 82091.0, 'HS-grad', 9.0, 'Never-married', 'Adm-clerical', 'Not-in-family', 'White', 'Female', 0.0, 0.0, 39.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
The 12-th sample [48.0, 'Private', 279724.0, 'HS-grad', 9.0, 'Married-civ-spouse', 'Machine-op-inspct', 'Husband', 'White', 'Male', 3103.0, 0.0, 48.0], the predicted result is <=50K, the actual result is >50K, the prediction is False
The 13-th sample [43.0, 'Private', 346189.0, 'Masters', 14.0, 'Married-civ-spouse', 'Exec-managerial', 'Husband', 'White', 'Male', 0.0, 0.0, 50.0], the predicted result is >50K, the actual result is >50K, the prediction is True
The 14-th sample [20.0, 'State-gov', 444554.0, 'Some-college', 10.0, 'Never-married', 'Other-service', 'Own-child', 'White', 'Male', 0.0, 0.0, 25.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
The 15-th sample [43.0, 'Private', 128354.0, 'HS-grad', 9.0, 'Married-civ-spouse', 'Adm-clerical', 'Wife', 'White', 'Female', 0.0, 0.0, 30.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
The 16-th sample [37.0, 'Private', 60548.0, 'HS-grad', 9.0, 'Widowed', 'Machine-op-inspct', 'Unmarried', 'White', 'Female', 0.0, 0.0, 20.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
The 17-th sample [34.0, 'Private', 107914.0, 'Bachelors', 13.0, 'Married-civ-spouse', 'Tech-support', 'Husband', 'White', 'Male', 0.0, 0.0, 47.0], the predicted result is >50K, the actual result is >50K, the prediction is True
The 18-th sample [34.0, 'Private', 238588.0, 'Some-college', 10.0, 'Never-married', 'Other-service', 'Own-child', 'Black', 'Female', 0.0, 0.0, 35.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
The 19-th sample [25.0, 'Private', 220931.0, 'Bachelors', 13.0, 'Never-married', 'Prof-specialty', 'Not-in-family', 'White', 'Male', 0.0, 0.0, 43.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
The 20-th sample [25.0, 'Private', 205947.0, 'Bachelors', 13.0, 'Married-civ-spouse', 'Prof-specialty', 'Husband', 'White', 'Male', 0.0, 0.0, 40.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
The 21-th sample [45.0, 'Self-emp-not-inc', 432824.0, 'HS-grad', 9.0, 'Married-civ-spouse', 'Craft-repair', 'Husband', 'White', 'Male', 7298.0, 0.0, 90.0], the predicted result is >50K, the actual result is >50K, the prediction is True
The 22-th sample [22.0, 'Private', 236427.0, 'HS-grad', 9.0, 'Never-married', 'Adm-clerical', 'Own-child', 'White', 'Male', 0.0, 0.0, 20.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
The 23-th sample [23.0, 'Private', 134446.0, 'HS-grad', 9.0, 'Separated', 'Machine-op-inspct', 'Unmarried', 'Black', 'Male', 0.0, 0.0, 54.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
The 24-th sample [54.0, 'Private', 99516.0, 'HS-grad', 9.0, 'Married-civ-spouse', 'Craft-repair', 'Husband', 'White', 'Male', 0.0, 0.0, 35.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
The 25-th sample [32.0, 'Self-emp-not-inc', 109282.0, 'Some-college', 10.0, 'Never-married', 'Prof-specialty', 'Not-in-family', 'White', 'Male', 0.0, 0.0, 60.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
The 26-th sample [46.0, 'State-gov', 106444.0, 'Some-college', 10.0, 'Married-civ-spouse', 'Exec-managerial', 'Husband', 'Black', 'Male', 7688.0, 0.0, 38.0], the predicted result is >50K, the actual result is >50K, the prediction is True
The 27-th sample [56.0, 'Self-emp-not-inc', 186651.0, '11th', 7.0, 'Widowed', 'Other-service', 'Unmarried', 'White', 'Female', 0.0, 0.0, 50.0], the predicted result is <=50K, the actual result is <=50K, the prediction is True
```