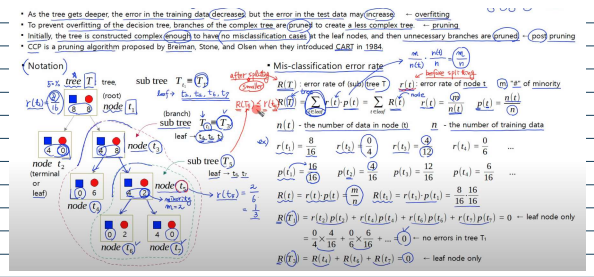
Cost Complexity Pruning [weakest link pruning]

Friday 3 May 2024 11:13 PM

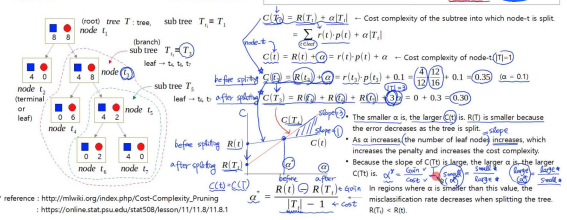


aims to find a subtree. Find a tree as good as the fully grown tree while having less # of leaves.

■ Cost Complexity Pruning (CCP) : post-pruning – definition of Cost Complexity

- Since trying to lower the misclassification rate R(T) results in a deeper tree, we define a new measure, C(T) and build a tree that lowers C(T).
- C(T) is called cost complexity. Objective: (Min C(T)) instead of Min R(T)
- C(T) is R(T) plus penalty. The deeper the tree, the greater the penalty.

• Cost Complexity: $C(T) = R(T) + \alpha|\tilde{T}|$ (misclassification rate of a tree; regularization constant; the number of leaf node in the tree; penalty term)



$C(t_4) = R(t_4) + \alpha = 0.25 + \alpha$ ← Cost complexity of the subtree into which node-t is split.
$= \sum_{i \in leaf} r(t) \cdot p(t) + \alpha|T_t|$
$C(t) = R(t) + \alpha = r(t) \cdot p(t) + \alpha$ ← Cost complexity of node-t (|T|=1)
before splitting: $C(t_2) = R(t_2) + \alpha = r(t_2) \cdot p(t_2) + 0.1 = \frac{4}{12} \times \frac{12}{16} + 0.1 = 0.35$ ($\alpha = 0.1$)
after splitting: $C(T_t) = R(t_4) + R(t_5) + 2\alpha = 0 + 0.3 = 0.30$

- The smaller α is, the larger C(t) is. R(T) is smaller because the error decreases as the tree is split.
- As α increases, the number of leaf nodes decreases, which increases the penalty and increases the cost complexity.
- Because the slope of C(Tt) is large, the larger α is, the larger C(Tt) is.

In regions where α is smaller than this value, the misclassification rate decreases when splitting the tree. R(Tt) < R(t).

* reference : http://mlwiki.org/index.php/Cost-Complexity_Pruning
: https://online.stat.psu.edu/stat508/lesson/11/11.8/11.8.1

Minimal cost-complexity pruning is an algorithm used to prune a tree to avoid over-fitting, described in Chapter 3 of [BRE]. This algorithm is parameterized by $\alpha \geq 0$ known as the complexity parameter. The complexity parameter is used to define the cost-complexity measure, $R_\alpha(T)$ of a given tree T:

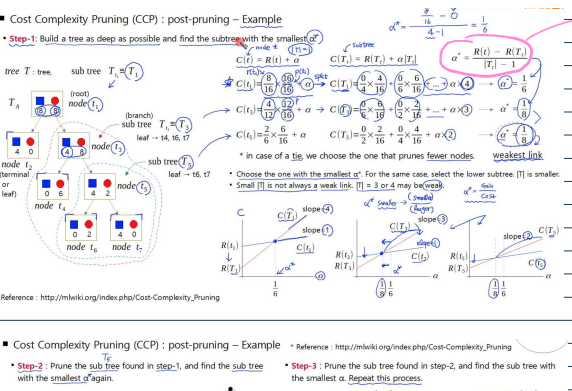
$$R_\alpha(T) = R(T) + \alpha|\tilde{T}|$$

where $|\tilde{T}|$ is the number of terminal nodes in T and $R(T)$ is traditionally defined as the total misclassification rate of the terminal nodes. Alternatively, scikit-learn uses the total sample weighted impurity of the terminal nodes for $R(T)$. As shown above, the impurity of a node depends on the criterion. Minimal cost-complexity pruning finds the subtree of T that minimizes $R_\alpha(T)$.

The cost complexity measure of a single node is $R_\alpha(t) = R(t) + \alpha$. The branch, T_t, is defined to be a tree where node t is its root. In general, the impurity of a node is greater than the sum of impurities of its terminal nodes, $R(T_t) < R(t)$. However, the cost complexity measure of a node, t, and its branch, T_t, can be equal depending on α. We define the effective α of a node to be the value where they are equal, $R_\alpha(T_t) = R_\alpha(t)$ or $\alpha_{eff}(t) = \frac{R(t) - R(T_t)}{|\tilde{T}_t| - 1}$. A non-terminal node with the smallest value of α_{eff} is the weakest link and will be pruned. This process stops when the pruned tree's minimal α_{eff} is greater than the `ccp_alpha` parameter.

■ Cost Complexity Pruning (CCP) : post-pruning – Example

- Step-1: Build a tree as deep as possible and find the subtree with the smallest α^*

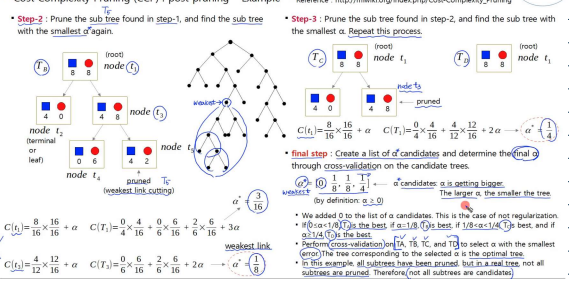


Reference : http://mlwiki.org/index.php/Cost-Complexity_Pruning

$\alpha^* = \frac{R(t) - R(T_t)}{|T_t| - 1}$ = Decrease in Impurity caused by split / # of leafs added to the subtree = gain / cost

■ Cost Complexity Pruning (CCP) : post-pruning – Example

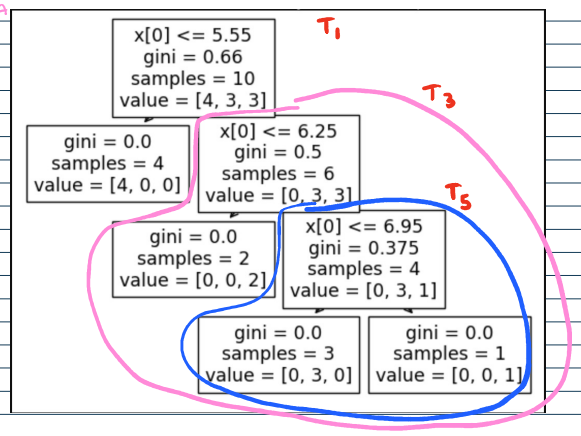
- Step-2 : Prune the sub tree found in step-1, and find the sub tree with the smallest α again.
- Step-3 : Prune the sub tree found in step-2, and find the sub tree with the smallest α. Repeat this process.



• final step : Create a list of α candidates and determine the final α through cross-validation on the candidate trees.
(by definition: α ≥ 0) ← α candidates: α is getting bigger. The larger α, the smaller the tree.

- We added 0 to the list of α candidates. This is the case of not regularization.
- If 0≤α<1/8, T_A is the best. if α=1/8, T_B is best. if 1/8<α<1/4, T_C is best, and if α≥1/4, T_D is the best.
- Perform cross-validation on TA, TB, TC, and TD to select α with the smallest error. The tree corresponding to the selected α is the optimal tree.
- In this example, all subtrees have been pruned, but in a real tree, not all subtrees are pruned. Therefore, (not all subtrees are candidates)

T_A



$R(t_1) = \frac{10}{10} \times 0.66 + \alpha$ $R(T_1) = 0 + 4\alpha$ $\alpha^* = \frac{R(t_1) - R(T_1)}{4 - 1} = 0.22$

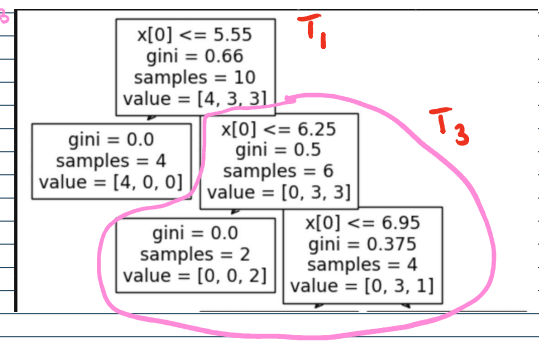
$R(t_3) = \frac{6}{10} \times 0.5 + \alpha$ $R(T_3) = 0 + 3\alpha$ $\alpha^* = \frac{R(t_3) - R(T_3)}{3 - 1} = 0.15$

$R(t_5) = \frac{4}{10} \times 0.375 + \alpha$ $R(T_5) = 0 + 2\alpha$ $\alpha^* = \frac{R(t_5) - R(T_5)}{2 - 1} = 0.15$

```
N_t / N * (impurity - N_t_R / N_t * right_impurity
                    - N_t_L / N_t * left_impurity)
```

↳ How much decrease in impurity this particular splitting has, compared to the whole tree.

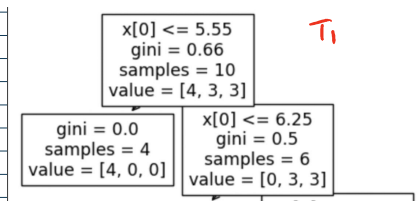
T_B



$R(t_2) = \frac{10}{10} \times 0.66 + \alpha$ $R(T_2) = \frac{4}{10} \times 0.375 + 3\alpha$ $\alpha^* = \frac{R(t_2) - R(T_2)}{3 - 1} = 0.255$

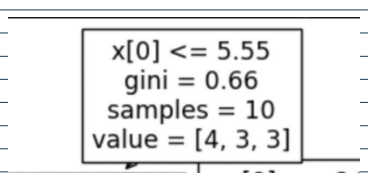
$R(t_3) = \frac{6}{10} \times 0.5 + \alpha$ $R(T_3) = \frac{4}{10} \times 0.375 + 2\alpha$ $\alpha^* = \frac{R(t_3) - R(T_3)}{2 - 1} = 0.15$

T_C



$R(t_1) = \frac{10}{10} \times 0.66 + \alpha$ $R(T_1) = \frac{6}{10} \times 0.5 + 2\alpha$ $\alpha^* = \frac{R(t_1) - R(T_1)}{2 - 1} = 0.36$

T_D



$\alpha^* = [0, 0.15, 0.15, 0.36]$ (T_A, T_B, T_C, T_D)

in sklearn → $\alpha^* = [0, 0.15, 0.36]$

T_A T_C T_D

```
path['ccp_alphas']
```

```
array([0.  , 0.15, 0.36])
```

```
path['impurities']
```

```
array([0.  , 0.3 , 0.66])
```

G.I (T) = weighted sum of G.I leaf nodes

$T(ccp\_alpha = X)$

↓

① start from fully grown tree

② prune the tree w.r.t minimum α

③ Stop when pruned tree has minimum $\alpha > X$