Decision Tree Classifier

110505259 陳柏燊

Function implementation

- _feature_split

```python
def _feature_split(self, X, y,n_classes):
        mx = len(np.unique(y))
        if mx <= 1:
            return None, None

        best_criterion = self._entropy(y,n_classes)
        best_idx, best_thr = None, None

        for i in range(X.shape[1]):
            splitnum = np.unique(X[:,i])
            for t in splitnum:
                left_y = y[X[:,i] <= t]
                right_y = y[X[:,i] > t]
                left_criterion = self._entropy(left_y,n_classes)
                right_criterion = self._entropy(right_y,n_classes)

                criterion = left_criterion*(left_y.size/y.size) + right_criterion*(right_y.size/y.size)
                if criterion < best_criterion:
                    best_criterion = criterion
                    best_idx = i
                    best_thr = t

        return best_idx, best_thr
```

If np.unique(y) == 1 means that's all instance in the node have the same target, so we don't need to split.
O.W. we go through all the feature and try all the treadhold to find which split method will get the least entropy and return the index of the split feature and the threadhold.

- _build_tree

```python
def _build_tree(self, X, y, depth=0):
    num_samples_per_class = [np.sum(y == i) for i in range(self.n_classes_)]
    predicted_class = np.argmax(num_samples_per_class)
    correct_label_num = num_samples_per_class[predicted_class]
    num_errors = y.size - correct_label_num
    node = Node(
        entropy = self._entropy(y,self.n_classes_),
        num_samples=y.size,
        num_samples_per_class=num_samples_per_class,
        predicted_class=predicted_class,
        num_errors=num_errors
    )

    if depth < self.max_depth:
        idx, thr = self._feature_split(X, y,self.n_classes_)

        if idx is not None:
            node.left = self._build_tree(X[X[:,idx] <= thr], y[X[:,idx] <= thr], depth + 1)
            node.right = self._build_tree(X[X[:,idx] > thr], y[X[:,idx] > thr], depth + 1)
            node.feature_index = idx
            node.threshold = thr

            pass
    return node
```

If the depth didn't exceed the limitation, we call _feature_split function to find how to split the tree. IF the return value aren't NULL we split the data and call _build_tree function to find the left and right children nodes of this node and link to it.

- _find_min_alpha

```
def _find_min_alpha(self, root):
    MinAlpha = float("inf")
    ret = [None, float("inf")]
    if root.left == None and root.right == None:
        return [root, float("inf")]

    rootAlpha = self._compute_alpha(root)
    if ret[1] > rootAlpha:
        ret[0] = root
        ret[1] = rootAlpha
    if root.left != None:
        tmp = self._find_min_alpha(root.left)
        if ret[1] > tmp[1]:
            ret[0] = tmp[0]
            ret[1] = tmp[1]
    if root.right != None:
        tmp = self._find_min_alpha(root.right)
        if ret[1] > tmp[1]:
            ret[0] = tmp[0]
            ret[1] = tmp[1]
    return ret
```

By the definition of alpha, it is numerical error to define an alpha of a leave node since it well cause to divide something with "Zero". So we return "inf" with the leave node.

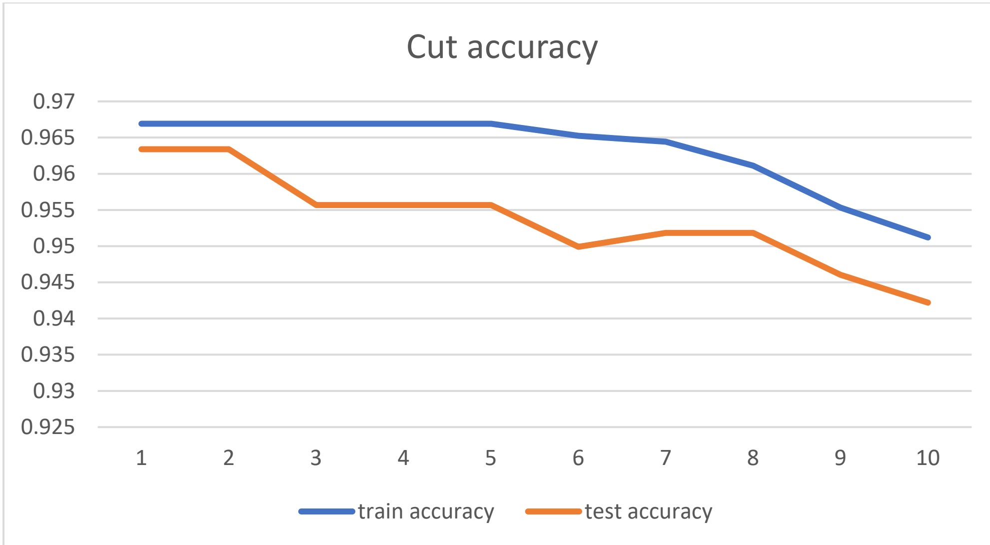
Otherwise, we computer the alpha of now node and compare with the minimum alpha of the left sub tree and right sub tree which is calculate by calling _find_min_alpha.

Decision tree before post-pruning accuracy (depth = 8, random_state = 1)

```
tree train accuracy: 0.966915
tree test accuracy: 0.963391
```

Decision tree after post-pruning accuracy (dep = 8, random_state = 1)

Cut	1	2	3	4	5	6	7	8	9	10
trai nac	0.966 915	0.966 915	0.966 915	0.966 915	0.966 915	0.965 261	0.964 433	0.961 125	0.955 335	0.951 199
test ac	0.963 391	0.963 391	0.955 684	0.955 684	0.955 684	0.949 904	0.951 83	0.951 83	0.946 05	0.942 197



The effect of different parameters

- Depth

  Depth = 8 to 15 (random_state = 1)

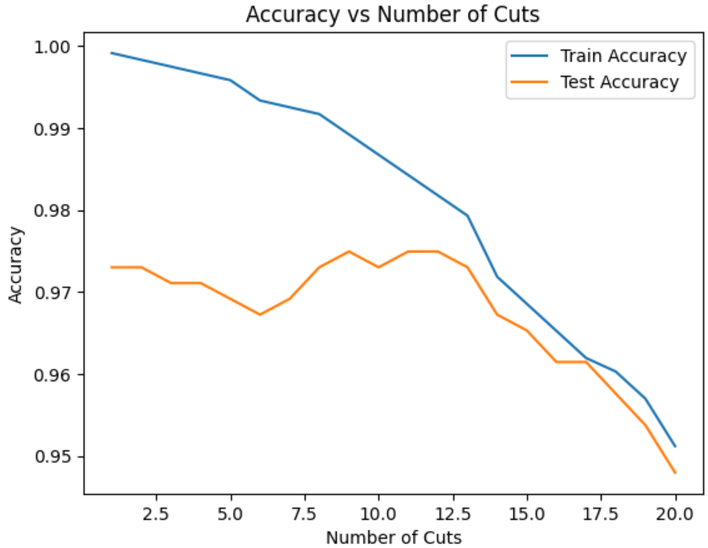
```
deep: 8
tree train accuracy: 0.966915
tree test accuracy: 0.944123
deep: 9
tree train accuracy: 0.984285
tree test accuracy: 0.976879
deep: 10
tree train accuracy: 0.995037
tree test accuracy: 0.986513
deep: 11
tree train accuracy: 0.997519
tree test accuracy: 0.988439
deep: 12
tree train accuracy: 1.000000
tree test accuracy: 0.988439
deep: 13
tree train accuracy: 1.000000
tree test accuracy: 0.988439
deep: 14
tree train accuracy: 1.000000
tree test accuracy: 0.988439
deep: 15
tree train accuracy: 1.000000
tree test accuracy: 0.988439
```

  We can observe that if we make tree go more deeper, we will finally get pefect predition on training data.

  Testing data accuracy will also increase but will have a upper boundary.

- Prune_times

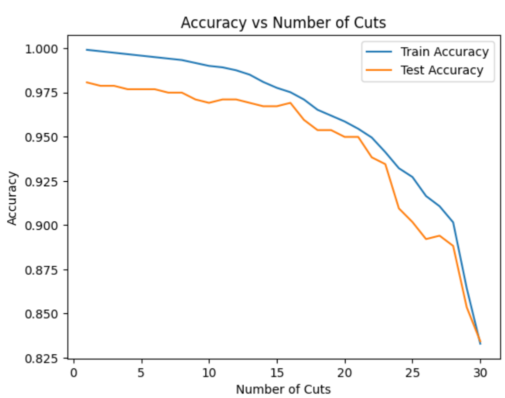
  Depth = 15, random_state = 1355373270



  We can see training accuracy keep decrease with more cut. When test accuracy increase a little and decrease. We consider the test accuracy with 5 cut lower with 10 cut cause by overfitting.

Conclusion

It hard to find the random_state to present overfitting. I have try a lot of random_state that most the of the different split with testing data and training data lead to testing accuracy and testing accuracy both decrease (like the picture below).



(random_state = 1903942106)

In my opinion, it seems to be cause by the data size. Maybe some bigger data set will lead to present overfitting easier.