**The Implementation of Decision Tree**

***Cheng Tan***

**1. The implementation of the decision tree algorithm.**

(1) Node

*# My DecisionTree*

**class** **Node**():

*"""*

*A Node in the decision tree.*

*Parameters*

*----------*

*value : double, default=None*

*The possible value.*

*best\_feat : int, default=None*

*The best feature to split in the current node.*

*best\_thr : double, default=None*

*The best threshold of the best feature.*

*leaf\_branch : Node, default=None*

*The left branch of the current node.*

*right\_branch : Node, default=None*

*The right branch of the current node.*

*"""*

**def** \_\_init\_\_(self, value, best\_feat, best\_thr, left\_branch, right\_branch):

self.value = value

self.best\_feat = best\_feat

self.best\_thr = best\_thr

self.left\_branch = left\_branch

self.right\_branch = right\_branch

(2) Decision Tree

**class** **DecisionTree**():

*"""*

*A classifier based on decision tree.*

*Parameters*

*----------*

*criterion : {'id3', 'c45'}, default='id3'*

*The criterion function to measure the quality of a split.*

*max\_depth : int, default=None*

*The maximum depth of the tree.*

*min\_sample\_leaf : int, default=None*

*The minimum numbers of samples required to be at a leaf node.*

*min\_impurity\_split : double, default=None*

*Threshold for early stopping in tree growth.*

*"""*

**def** \_\_init\_\_(self, criterion='id3', max\_depth=**None**, \

min\_sample\_leaf=4, min\_impurity\_split=1e-7):

self.criterion = criterion

self.max\_depth = (np.iinfo(np.int32).max **if** max\_depth **is** **None** **else** max\_depth)

self.min\_sample\_leaf = min\_sample\_leaf

self.min\_impurity\_split = min\_impurity\_split

self.root = **None**

**def** \_\_entropy\_\_(self, x):

*"""*

*Calculate the entropy.*

*Returns*

*-------*

*H : double*

*The entropy of x.*

*"""*

D = x.shape[0]

H = 0

**for** k **in** np.unique(x):

Ck = len(x[x == k])

H += - (Ck / D) \* np.log2(Ck / D)

**return** H

**def** \_\_gini\_\_(self, x):

*"""*

*Calculate the gini coefficient.*

*Returns*

*-------*

*G : double*

*The gini coeeficient of x.*

*"""*

D = x.shape[0]

G = 0

**for** k **in** np.unique(x):

Ck = len(x[x == k])

G += (Ck / D) \*\* 2

G = 1 - G

**return** G

**def** \_\_id3\_\_(self, y, yl, yr):

*"""*

*Information gain.*

*Returns*

*-------*

*: double*

*"""*

D, Dl, Dr = len(y), len(yl), len(yr)

H\_D = self.\_\_entropy\_\_(y)

H\_Dl = Dl / D \* self.\_\_entropy\_\_(yl)

H\_Dr = Dr / D \* self.\_\_entropy\_\_(yr)

**return** H\_D - (H\_Dl + H\_Dr)

**def** \_\_c45\_\_(self, y, yl, yr):

*"""*

*Information gain ratio.*

*Returns*

*-------*

*: double*

*"""*

D, Dl, Dr = len(y), len(yl), len(yr)

H\_D = self.\_\_entropy\_\_(y)

H\_Dl = Dl / D \* self.\_\_entropy\_\_(yl)

H\_Dr = Dr / D \* self.\_\_entropy\_\_(yr)

**return** (H\_D - (H\_Dl + H\_Dr)) / H\_D

**def** \_\_cart\_\_(self, y, yl, yr):

*"""*

*Gini coefficient.*

*Returns*

*-------*

*: double*

*"""*

D, Dl, Dr = len(y), len(yl), len(yr)

G\_Dl = Dl / D \* self.\_\_gini\_\_(yl)

G\_Dr = Dr / D \* self.\_\_gini\_\_(yr)

*# To get minimum gini, add negative symbol*

**return** - (G\_Dl + G\_Dr)

**def** \_\_criterion\_\_(self, y, yl, yr):

*"""*

*Choose one of the criterion function to measure the split.*

*Returns*

*-------*

*: double*

*"""*

**if** self.criterion == 'id3':

**return** self.\_\_id3\_\_(y, yl, yr)

**elif** self.criterion == 'c45':

**return** self.\_\_c45\_\_(y, yl, yr)

**else**:

**raise** **ValueError**('The criterion should be one of [**\'**id3**\'**, **\'**c45**\'**].')

**def** \_\_getThr\_\_(self, feature):

*"""*

*Get the initialized thresholds of a feature.*

*Returns*

*-------*

*: list*

*"""*

t = sorted(feature)

**return** [(t[i] + t[i-1]) / 2 **for** i **in** range(1, len(t))]

**def** \_\_getSplit\_\_(self, X, y, feat\_ind, thr):

*"""*

*Get the split of X and y.*

*Returns*

*-------*

*: tuple of list*

*"""*

t = X[:, feat\_ind] < thr

**return** X[t], y[t], X[~t], y[~t]

**def** \_\_build\_\_(self, X, y, depth):

*"""*

*Build the decision tree.*

*Returns*

*-------*

*node : Node*

*"""*

node = Node(**None**, **None**, **None**, **None**, **None**)

row, col = X.shape

**if** depth <= self.max\_depth **and** row > self.min\_sample\_leaf:

*# Find the best feature to split*

bfeat\_ind = **None**

bfeat\_gain = np.iinfo(np.int32).min

bfeat\_thr = **None**

**for** feat\_ind **in** range(col):

thresholds = self.\_\_getThr\_\_(X[:, feat\_ind])

**for** thr **in** thresholds:

X1, y1, X2, y2 = self.\_\_getSplit\_\_(X, y, feat\_ind, thr)

gain = self.\_\_criterion\_\_(y, y1, y2)

**if** gain >= bfeat\_gain:

bfeat\_gain = gain

bfeat\_thr = thr

bfeat\_ind = feat\_ind

**if** bfeat\_gain > self.min\_impurity\_split:

*# Continue splitting*

X1, y1, X2, y2 = self.\_\_getSplit\_\_(X, y, bfeat\_ind, bfeat\_thr)

node.best\_thr = bfeat\_thr

node.best\_feat = bfeat\_ind

node.value = **None**

**del** X

node.left\_branch = self.\_\_build\_\_(X1, y1, depth + 1)

node.right\_branch = self.\_\_build\_\_(X2, y2, depth + 1)

**else**:

*# Stop splitting*

**del** X

node.value = np.argmax(np.bincount(y.flatten()))

**return** node

**else**:

**del** X

node.value = np.argmax(np.bincount(y.flatten()))

**return** node

**def** \_\_find\_\_(self, x, node):

*"""*

*Find the potential predicted label in the decision tree.*

*Returns*

*-------*

*: int*

*"""*

**if** node.value **is** **None**:

**if** x[node.best\_feat] < node.best\_thr:

**return** self.\_\_find\_\_(x, node.left\_branch)

**else**:

**return** self.\_\_find\_\_(x, node.right\_branch)

**else**:

**return** node.value

**def** fit(self, X, y):

**assert** isinstance(X, np.ndarray) **and** isinstance(y, np.ndarray)

**if** y.ndim == 1:

y = np.reshape(y, (-1, 1))

*# Build tree*

self.root = self.\_\_build\_\_(X, y, 0)

**def** predict(self, X):

**assert** isinstance(X, np.ndarray)

y = []

**for** x **in** X:

y.append(self.\_\_find\_\_(x, self.root))

**return** np.array(y)

**def** score(self, X, y):

**assert** isinstance(X, np.ndarray) **and** isinstance(y, np.ndarray)

pred = self.predict(X)

**return** (pred == y).sum() / pred.shape[0]

**2. Comparison**

(1) Accuracy

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Accuracy (%) | Fold1 | Fold2 | Fold3 | Fold4 | Fold5 | Avg. |
| Built-in | 92.98 | 95.61 | 91.22 | 96.49 | 92.92 | 93.85 |
| Mine | 93.86 | 95.61 | 92.98 | 94.73 | 95.57 | 94.55 |

(2) Training Time

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Time (s) | Fold1 | Fold2 | Fold3 | Fold4 | Fold5 | Avg. |
| Built-in | 0.0081 | 0.0069 | 0.0064 | 0.0062 | 0.0080 | 0.0071 |
| Mine | 4.9146 | 5.5333 | 5.1658 | 5.1595 | 5.2357 | 5.2018 |

(3) Test Time

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Time (ms) | Fold1 | Fold2 | Fold3 | Fold4 | Fold5 | Avg. |
| Built-in | 0.4890 | 0.6101 | 0.3319 | 0.2789 | 0.3757 | 0.4171 |
| Mine | 0.2398 | 0.2649 | 0.2441 | 0.2868 | 0.2698 | 0.2611 |