# Decision tree

## 1. Basic theory

### 1.1 Concept

A decision tree is a nonparametric supervised learning algorithm for classification and regression tasks. It is a hierarchical tree structure consisting of root nodes, branches, internal nodes and leaf nodes. A decision tree starts at the root node, which does not have any incoming branches. The outgoing branches of the root node then provide information to internal nodes (also called decision nodes). Both nodes perform evaluations based on available capabilities to form homogeneous subsets, which are represented by leaf nodes or terminal nodes. Leaf nodes represent all possible outcomes within the dataset. Only decision trees for classification are discussed here.

Decision tree classification: starting from the root node, test a feature of the instance, and assign the instance to its child nodes according to the test results. At this time, each child node corresponds to a value of the feature, and the instance is tested recursively. And allocate until the leaf node is reached, and finally the instance is divided into the class of the leaf node.

### 1.2 Principle

Decision tree learning adopts a "one-by-one" strategy, performing a greedy search to identify the best split points within the decision tree. This splitting process is then repeated in a top-down regression fashion until all or most records are marked with a specific class label. Whether or not to classify all data points into homogeneous subsets depends largely on the complexity of the decision tree. Smaller decision trees are more likely to get leaf nodes that cannot be split, i.e. data points within a single class. However, as decision trees grow larger, it becomes increasingly difficult to maintain this purity, and often results in too little data falling within a given subtree. This situation is known as data fragmentation, and often leads to data overfitting. Therefore, small decision trees are usually chosen, which is consistent with the "simple and efficient principle" of Occam's razor, that is, "don't increase entities if they are not necessary". In other words, we should only increase the complexity of decision trees when necessary, because the simplest is often the best. To reduce complexity and prevent overfitting, pruning algorithms are often employed; this process removes branches of less important features. Then, the fit of the model is evaluated by cross-validation.

### 1.3 Construction of decision tree

The algorithm of decision tree learning is usually a process of recursively selecting the optimal attribute and dividing the training data according to the attribute, so that each sub-data set has a best classification process. This process corresponds to the division of the attribute space and the construction of the decision tree.

1. Start: build the root node, put all the training data on the root node, select an optimal attribute, and divide the training data set into subsets according to this attribute, so that each subset has the best classification under the current conditions.
2. If these subsets have been basically correctly classified, then construct leaf nodes and assign these subsets to the corresponding leaf nodes.
3. If there are still subsets that cannot be correctly classified, then select new optimal attributes for these subsets, continue to segment them, build corresponding nodes, and recursively until all training data

subsets are basically correctly classified , or until there is no suitable attribute.
4. Each subset is divided into leaf nodes, that is, has a clear class, thus generating a decision tree.

## 1.4 How to choose the optimal attribute on each node

While many methods can be used to select the optimal attribute at each node, the two methods information gain and Gini impurity are the most commonly used split criteria for decision tree models. They help to assess the quality of each test condition as well as the sample classification ability.

### Information gain

It is difficult to explain information gain without discussing information entropy (Shannon entropy). Information entropy is a concept derived from information theory and is used to measure the impurity of sample values. It is defined by the following formula:

$$Ent(D) = -\sum_{k=1}^{|y|} p_k log_2 p_k$$

where:

- $D$ represents the data set for which information entropy is to be calculated
- $k$ represents the class in the set $D$, there are a total of $|y|$ classes
- $p_k$ represents the proportion of the $k$ class samples in the set $D$

The entropy value is between 0 and 1. If all samples in dataset S belong to one class, the entropy value is zero. If half the samples belong to one class and the other half belong to the other class, the entropy value is the highest value of 1. In order to choose the best attribute to split on and determine the best decision tree, the attribute with the smallest entropy value must be used.

Information gain represents the difference in entropy values before and after splitting a given attribute. The split on the attribute with the largest information gain works best because it works best when classifying according to the target class of the training data. Information gain is usually expressed by the following formula:

$$Gain(D, a) = Ent(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} Ent(D^v)$$

where:

- $a$ represents a specific feature, which has $V$ possible values
- $D^v$ represents all samples in $D$ with feature $a$ valued as $a^v$
- $\frac{|D^v|}{|D|}$ represents the proportion of $D^v$ in $D$

### Gini Impurity

Gini impurity refers to the probability of misclassification when random data points are labeled according to the class distribution of a dataset. Similar to information entropy, if dataset S cannot be further split (i.e. belongs to a class), then its impurity is zero. The formula is as follows:

$$Gini(D) = 1 - \sum_{k=1}^{|y|} p_k^2$$

## 1.5 Advantages and disadvantages of decision trees

**Advantage:**

- **easy to explain**: The Boolean logic and visual representation of decision trees are helpful to understand and use. The hierarchical structure of decision trees also makes it easier to spot the most important features, which other algorithms don't make clear, such as neural networks.
- **no data preparation required**: Decision trees have many features and are more flexible than other classification methods. It can handle various data types such as discrete or continuous, and continuous values can be converted to categorical values by using thresholds. In addition, it can handle data with missing values that can cause difficulties for other classification methods such as Naive Bayes.
- **more flexible**: Decision trees can be used for classification and regression tasks and are more flexible than some other algorithms. It is also not sensitive to potential relationships between features; this means that if two variables are extremely correlated, the algorithm will only choose one of the features to split on.

---

**Disadvantages**

- **easy to overfit**: Complex decision trees tend to overfit and do not work well with new data. This situation can be avoided by pre-pruning or post-pruning. Pre-pruning stops the development of a decision tree when there is insufficient data, while post-pruning removes subtrees with insufficient data after the decision tree is formed.
- **predictors have high variance**: Small changes in the data can produce very different decision trees. The bagging algorithm or the average of the estimates can reduce the variance of the decision tree. However, this approach is of limited use and produces highly correlated predictors.
- **higher cost**: The decision tree adopts the greedy search method in the construction process, and its training cost will be higher than other algorithms.

# 2. Example

Here, we will illustrate the use of decision trees with an example of a bank loan decision.

## 2.1 Dataset explanation

The dataset will be shown in section 2.2. This simple dataset has only 15 pieces of data, and the columns of each piece of data represent the applicant's age, whether they have a job, whether they have their own house, their credit status, and their category (loan or not). The resulting decision tree is required to decide whether to grant a loan based on the applicant's age, whether they have a job, whether they have a house, and their credit status. The actual meaning of the value of each attribute of the dataset:

- age: 0 for youth, 1 for middle age, 2 for old age
- haveWork: 0 for no, 1 for yes
- haveHouse: 0 for no, 1 for yes
- credit: 0 is fair, 1 is good, 2 is very good
- category: no means no, yes means yes

## 2.2 sklearn

Python has many excellent machine learning libraries, including many commonly used machine learning algorithms, which allows us to implement these algorithms without having to start from scratch, which brings us a lot of convenience. sklearn is one such tool, which we will use to build decision trees. It has the following characteristics:

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Official website: [click here (https://scikit-learn.org/stable/)](https://scikit-learn.org/stable/)

## 2.3 Import required packages

In [ ]:

```
from sklearn import tree
import pandas as pd
```

## 2.4 Prepare the dataset

In [ ]:

```
def createDataSet():
    dataSet = [[0, 0, 0, 0, 'no'],
               [0, 0, 0, 1, 'no'],
               [0, 1, 0, 1, 'yes'],
               [0, 1, 1, 0, 'yes'],
               [0, 0, 0, 0, 'no'],
               [1, 0, 0, 0, 'no'],
               [1, 0, 0, 1, 'no'],
               [1, 1, 1, 1, 'yes'],
               [1, 0, 1, 2, 'yes'],
               [1, 0, 1, 2, 'yes'],
               [2, 0, 1, 2, 'yes'],
               [2, 0, 1, 1, 'yes'],
               [2, 1, 0, 1, 'yes'],
               [2, 1, 0, 2, 'yes'],
               [2, 0, 0, 0, 'no']]
    labels = ['age', 'haveWork', 'haveHouse', 'credit']
    return dataSet, labels


if __name__ == '__main__':
    dataSet, Labels = createDataSet()
    print(dataSet)
```

## 2.5 Extract the categories for each set of data

In [ ]:

```
data_target = []
for each in dataSet:
    data_target.append(each[-1])
print(data_target)
```

## 2.6 Convert data to pandas.DataFrame

In [ ]:

```
data_list = []
data_dict = {}
for each_label in Labels:
    for each in dataSet:
        data_list.append(each[Labels.index(each_label)])
    data_dict[each_label] = data_list
    data_list = []
print(data_dict)
data_pd = pd.DataFrame(data_dict)
print(data_pd)
```

## 2.7 Build a decision tree

Here we use DecisionTreeClassifier to build the decision tree with default parameters. You can also modify the parameters according to your needs. For the meaning of the parameters of DecisionTreeClassifier, refer to the official documentation: [click here (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)

In [ ]:

```
clf = tree.DecisionTreeClassifier()
clf = clf.fit(data_pd.values.tolist(), data_target)
```

## 2.8 Visualization

In [ ]:

```
tree.plot_tree(clf,feature_names = Labels);
```

## 2.9 Predict

Using the decision tree generated above, the decision result of the loan application for an applicant who is middle-aged, has a job, has a house, and has an average credit situation is yes.

In [ ]:

```
print(clf.predict([[1, 1, 1, 0]]))
```

## 3. Exercise: determine the type of contact lenses a patient needs to wear

The data set is given below, where the columns represent age, prescript, whether astigmatism, number of tears, and classification labels (glasses type). The dataset is given below, with columns for age, symptoms, astigmatism, number of tears, and a classification label (glasses type). Please combine the knowledge learned above to build a decision tree and visualize it.

**Tips:**The dataset attribute field is of type string, if you want to use sklearn to build a decision tree, you need to encode it first.

In [ ]:

```
dataSet=[['young', 'myope', 'no', 'reduced', 'no lenses'],
        ['young', 'myope', 'no', 'normal', 'soft'],
        ['young', 'myope', 'yes', 'reduced', 'no lenses'],
        ['young', 'myope', 'yes', 'normal', 'hard'],
        ['young', 'hyper', 'no', 'reduced', 'no lenses'],
        ['young', 'hyper', 'no', 'normal', 'soft'],
        ['young', 'hyper', 'yes', 'reduced', 'no lenses'],
        ['young', 'hyper', 'yes', 'normal', 'hard'],
        ['pre', 'myope', 'no', 'reduced', 'no lenses'],
        ['pre', 'myope', 'no', 'normal', 'soft'],
        ['pre', 'myope', 'yes', 'reduced', 'no lenses'],
        ['pre', 'myope', 'yes', 'normal', 'hard'],
        ['pre', 'hyper', 'no', 'reduced', 'no lenses'],
        ['pre', 'hyper', 'no', 'normal', 'soft'],
        ['pre', 'hyper', 'yes', 'reduced', 'no lenses'],
        ['pre', 'hyper', 'yes', 'normal', 'no lenses'],
        ['presbyopic', 'myope', 'no', 'reduced', 'no lenses'],
        ['presbyopic', 'myope', 'no', 'normal', 'no lenses'],
        ['presbyopic', 'myope', 'yes', 'reduced', 'no lenses'],
        ['presbyopic', 'myope', 'yes', 'normal', 'hard'],
        ['presbyopic', 'hyper', 'no', 'reduced', 'no lenses'],
        ['presbyopic', 'hyper', 'no', 'normal', 'soft'],
        ['presbyopic', 'hyper', 'yes', 'reduced', 'no lenses'],
        ['presbyopic', 'hyper', 'yes', 'normal', 'no lenses']]

labels=['age', 'prescript', 'astigmatic', 'tearRate']
```