School of Data and Computer Science, Sun Yat-sen University

E10 Decision Tree

18364066 Lu Yanzuo

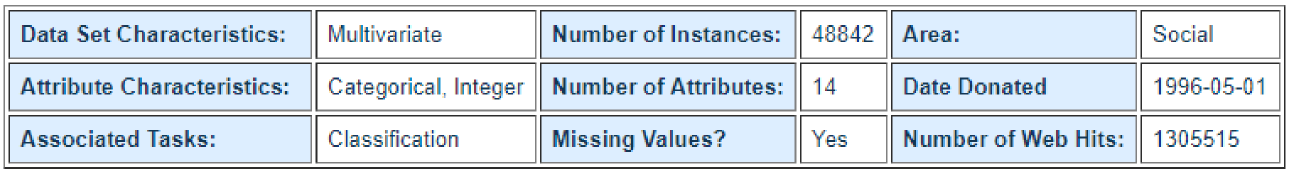
November 18, 2020

Contents

1 Datasets

The UCI dataset (http://archive.ics.uci.edu/ml/index.php) is the most widely used dataset for machine learning. If you are interested in other datasets in other areas, you can refer to https://www.zhihu.com/question/63383992/answer/222718972.

Today's experiment is conducted with the **Adult Data Set** which can be found in http://archive.ics.uci.edu/ml/datasets/Adult.



Data Set Characteristics:	Multivariate	**Number of Instances:**	48842	**Area:**	Social
Attribute Characteristics:	Categorical, Integer	**Number of Attributes:**	14	**Date Donated**	1996-05-01
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	1305515

You can also find 3 related files in the current folder, `adult.name` is the description of **Adult Data Set**, `adult.data` is the training set, and `adult.test` is the testing set. There are 14 attributes in this dataset:

```
>50K, <=50K.


1. age: continuous.
2. workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov,
    Local-gov,
State-gov, Without-pay, Never-worked.
3. fnlwgt: continuous.
4. education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc
    -acdm,
Assoc-voc, 9th, 7th-8th, 12th, Masters, 5. 1st-4th, 10th, Doctorate, 5th
    -6th,
Preschool.
5. education-num: continuous.
6. marital-status: Married-civ-spouse, Divorced, Never-married,
    Separated,
Widowed, Married-spouse-absent, Married-AF-spouse.
7. occupation: Tech-support, Craft-repair, Other-service, Sales,
Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct,
Adm-clerical,Farming-fishing,Transport-moving,Priv-house-serv,Protective
    -serv,
Armed−Forces.
8. relationship: Wife,Own−child,Husband,Not−in−family,Other−relative,
   Unmarried.
9. race: White, Asian−Pac−Islander, Amer−Indian−Eskimo, Other, Black.
10. sex: Female, Male.
11. capital−gain: continuous.
12. capital−loss: continuous.
13. hours−per−week: continuous.
14. native−country: United−States, Cambodia,England,Puerto−Rico,Canada,
   Germany,
Outlying−US(Guam−USVI−etc),India,Japan,Greece, South,China,Cuba,Iran,
   Honduras,
Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland,
    France,
Dominican−Republic,Laos,Ecuador,Taiwan, Haiti, Columbia,Hungary,
   Guatemala,
Nicaragua,Scotland,Thailand,Yugoslavia,El−Salvador, Trinadad&Tobago,Peru
   ,Hong,
Holand−Netherlands.
```

Prediction task is to determine whether a person makes over 50K a year.

2 Decision Tree

2.1 ID3

ID3 (Iterative Dichotomiser 3) was developed in 1986 by Ross Quinlan. The algorithm creates a multiway tree, finding for each node (i.e. in a greedy manner) the categorical feature that will yield the largest information gain for categorical targets. Trees are grown to their maximum size and then a pruning step is usually applied to improve the ability of the tree to generalise to unseen data.

ID3 Algorithm:

1. Begins with the original set S as the root node.
2. Calculate the entropy of every attribute a of the data set S.
3. Partition the set S into subsets using the attribute for which the resulting entropy after splitting is minimized; or, equivalently, information gain is maximum.

4. Make a decision tree node containing that attribute.
5. Recur on subsets using remaining attributes.

Recursion on a subset may stop in one of these cases:

- every element in the subset belongs to the same class; in which case the node is turned into a leaf node and labelled with the class of the examples.
- there are no more attributes to be selected, but the examples still do not belong to the same class. In this case, the node is made a leaf node and labelled with the most common class of the examples in the subset.
- there are no examples in the subset, which happens when no example in the parent set was found to match a specific value of the selected attribute.

ID3 shortcomings:

- ID3 does not guarantee an optimal solution.
- ID3 can overfit the training data.
- ID3 is harder to use on continuous data.

Entropy:

Entropy $H(S)$ is a measure of the amount of uncertainty in the set S.

$$H(S) = \sum_{x \in X} -p(x) \log_2 p(x)$$

where

- S is the current dataset for which entropy is being calculated
- X is the set of classes in S
- $p(x)$ is the proportion of the number of elements in class x to the number of elements in set S.

Information gain:

Information gain $IG(A)$ is the measure of the difference in entropy from before to after the set S is split on an attribute A. In other words, how much uncertainty in S was reduced after splitting set S on attribute A.

$$IG(S, A) = H(S) - \sum_{t \in T} p(t) H(t) = H(S) - H(S \mid A)$$

where

- $H(S)$ is the entropy of set S
- T is the subsets created from splitting set S by attribute A such that $S = \cup_{t \in T} t$
- $p(t)$ is the proportion of the number of elements in t to the number of elements in set S
- $H(t)$ is the entropy of subset t.

2.2 C4.5 and CART

C4.5 is the successor to ID3 and removed the restriction that features must be categorical by dynamically defining a discrete attribute (based on numerical variables) that partitions the continuous attribute value into a discrete set of intervals. C4.5 converts the trained trees (i.e. the output of the ID3 algorithm) into sets of if-then rules. These accuracy of each rule is then evaluated to determine the order in which they should be applied. Pruning is done by removing a rule’s precondition if the accuracy of the rule improves without it.

C5.0 is Quinlan’s latest version release under a proprietary license. It uses less memory and builds smaller rulesets than C4.5 while being more accurate.

CART (Classification and Regression Trees) is very similar to C4.5, but it differs in that it supports numerical target variables (regression) and does not compute rule sets. CART constructs binary trees using the feature and threshold that yield the largest information gain at each node.

3 Tasks

- Given the training dataset `adult.data` and the testing dataset `adult.test`, please accomplish the prediction task to determine whether a person makes over 50K a year in `adult.test` by using ID3 (or C4.5, CART) algorithm (C++ or Python), and compute the accuracy.
  1. You can process the continuous data with **bi-partition** method.
  2. You can use prepruning or postpruning to avoid the overfitting problem.
  3. You can assign probability weights to solve the missing attributes (data) problem.
- Please finish the experimental report named `E10_YourNumber.pdf`, and send it to `ai_2020@foxmail.com`

4 Implementation

4.1 ID3 Algorithm

决策树学习的关键其实就在于选择最优的划分属性，我们总是希望经过划分后，分支节点中某些标签的类别越来越清晰，所占比例也越来越高，如何度量这个所谓的“清晰”也就导致了不同算法的实现方式有所差别。

在 ID3 算法中，我们采用信息增益 IG 这个值来作为度量方式，我们选取使得信息增益最大的特征进行划分，而信息熵是代表随机变量的复杂度，条件熵代表在某一个条件下随机变量的复杂度，信息增益则刚好是信息熵减去条件熵。具体的计算公式已经在该文档的前半部分有详细的说明，下面大致讲述一下在实现中的几个 tricks。

4.2 缺失值处理

根据我们所采用的 adults 数据集我们可以发现，缺失值主要存在于三个方面。

第一个方面是在计算条件熵时，某些样本条目可能缺失某个特征值，此时我们可以直接忽略掉这些条目，将包含该特征值的条目作为一个新的子集计算条件熵，最后再乘以子集在总集中的比例系数作为结果返回即可。

第二个方面是在对训练样本条目进行划分时，某个条目可能缺失该特征值（在数据集中体现为问号?)，经过一些资料的查询和学习，我了解到一个比较实用的方法是对每个条目赋予一个权重值，当某个条目在该特征缺失时，我们可以根据该特征所有条目在所有标签的比例分布，然后在每一个分支节点中都加入这些条目，并赋予这些条目一个与比例分布相同的权重值。

第三个方面是对测试样本条目进行决策时，该测试样本可能不包含某个特征值，这里我使用的方法是和训练样本划分类似的，根据分支节点的比例分布在 [0,1) 范围中随机选取一个值，根据该值在分支节点中进行选取。

4.3 连续值处理

根据该文档中的提示，我们可以使用二分法来对连续值进行划分，具体做法就是对所有训练样本的连续值特征进行去重然后从小到大进行排序，每两个值取中点作为一个划分点进行测试，大于该中点值归为一类，小于该等于该中点值归为另一类，然后选取条件熵最小的那个中点值作为最后的划分点即可。

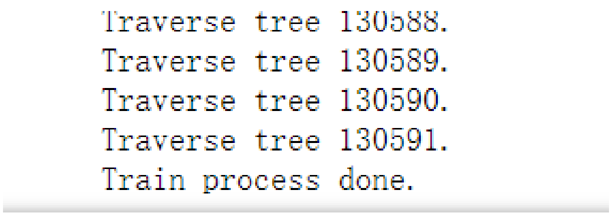
4.4 剪枝

起初我在没有进行剪枝时发现一个比较大的问题就是，决策树扩展了许多无用的节点，在打印的 log 中经常能够看到某个数据集的信息熵已经是 0.0 了，但因为还有属性可以划分，所以这个节点的数据集还会被继续划分直至没有可划分属性，为了解决这个问题我在程序中添加了一个非常简单的判断条件，以此剪枝掉那些无需再划分的节点，判断依据主要是当前节点的信息熵是否已为零，此时代表数据集内所有样本都是同一类别的。除此之外，还有一些预剪枝和后剪枝的技术，由于代码量已经较大没有添加进去。

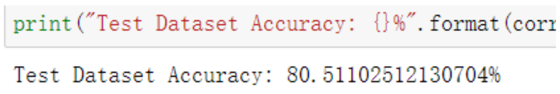
5 Result

已知 adults 训练样本数目是 32561. 测试样本数目是 16282，根据我的 ID3 算法实现，在训练过程中包括空节点和叶节点总共扩展了大约 130000 个子节点，最后在测试集上没有采用交叉验证等方法，只是单纯地根据整棵决策树对每个样本进行决策，需要注意的一点是，有可能会出现下一个节点是

空节点的情况，这里我采用的解决办法是直接使用父节点的结果返回，最后得到的测试集精度约为 80.5%。



```
Traverse tree 130588.
Traverse tree 130589.
Traverse tree 130590.
Traverse tree 130591.
Train process done.
```



```
print("Test Dataset Accuracy: {}%".format(corr
Test Dataset Accuracy: 80.51102512130704%
```

6 Code

```
import copy
import math
import random

import numpy as np


WORKCLASS = ["Private", "Self-emp-not-inc", "Self-emp-inc", "Federal-gov
    ", "Local-gov",
     "State-gov", "Without-pay", "Never-worked"]
EDUCATION = ["Bachelors", "Some-college", "11th", "HS-grad", "Prof-
    school", "Assoc-acdm",
     "Assoc-voc", "9th", "7th-8th", "12th", "Masters", "1st-4th", "10th",
          "Doctorate",
     "5th-6th", "Preschool"]
MARITAL_STATUS = ["Married-civ-spouse", "Divorced", "Never-married", "
    Separated", "Widowed",
     "Married-spouse-absent", "Married-AF-spouse"]
OCCUPATION = ["Tech-support", "Craft-repair", "Other-service", "Sales",
    "Exec-managerial",
    "Prof-specialty", "Handlers-cleaners", "Machine-op-inspct", "Adm-
        clerical",
    "Farming-fishing", "Transport-moving", "Priv-house-serv", "
        Protective-serv",
    "Armed-Forces"]
RELATIONSHIP = ["Wife", "Own-child", "Husband", "Not-in-family", "Other-
    relative",
    "Unmarried"]
RACE = ["White", "Asian-Pac-Islander", "Amer-Indian-Eskimo", "Other", "
    Black"]
SEX = ["Female", "Male"]
NATIVE_COUNTRY = ["United-States", "Cambodia", "England", "Puerto-Rico",
    "Canada",
    "Germany", "Outlying-US(Guam-USVI-etc)", "India", "Japan", "Greece",
        "South",
    "China", "Cuba", "Iran", "Honduras", "Philippines", "Italy", "Poland
        ", "Jamaica",
    "Vietnam", "Mexico", "Portugal", "Ireland", "France", "Dominican-
        Republic", "Laos",
    "Ecuador", "Taiwan", "Haiti", "Columbia", "Hungary", "Guatemala", "
        Nicaragua",
    "Scotland", "Thailand", "Yugoslavia", "El-Salvador", "Trinadad&
        Tobago", "Peru",
    "Hong", "Holand-Netherlands"]
LABEL = ["<=50K", ">50K"]

CONT_ATTR = [0, 2, 4, 10, 11, 12]
SEPE_ATTR = [1, 3, 5, 6, 7, 8, 9, 13]
SEPE_ATTR_LEN = [len(WORKCLASS), len(EDUCATION), len(MARITAL_STATUS),
    len(OCCUPATION),
    len(RELATIONSHIP), len(RACE), len(SEX), len(NATIVE_COUNTRY)]
ATTR_NAME = ["age", "workclass", "fnlwgt", "education", "education_num",
    "marital_status",
        "occupation", "relationship", "race", "sex", "capital_gain", "
           capital_loss", "hours_per_week",
        "native_country"]

class Item(object):
    def __init__(self, age, workclass, fnlwgt, education, education_num,
        marital_status,
        occupation, relationship, race, sex, capital_gain, capital_loss,
            hours_per_week,
        native_country, label, weight=None, test=False):
        self.age = −1 if age == "?" else int(age)
        self.workclass = −1 if workclass == "?" else WORKCLASS.index(
           workclass)
        self.fnlwgt = −1 if fnlwgt == "?" else int(fnlwgt)
        self.education = −1 if education == "?" else EDUCATION.index(
           education)
        self.education_num = −1 if education_num == "?" else int(
           education_num)
        self.marital_status = −1 if marital_status == "?" else
           MARITAL_STATUS.index(marital_status)
        self.occupation = −1 if occupation == "?" else OCCUPATION.index(
           occupation)
        self.relationship = −1 if relationship == "?" else RELATIONSHIP.
           index(relationship)
        self.race = −1 if race == "?" else RACE.index(race)
        self.sex = −1 if sex == "?" else SEX.index(sex)
        self.capital_gain = −1 if capital_gain == "?" else int(
           capital_gain)
        self.capital_loss = −1 if capital_loss == "?" else int(
           capital_loss)
        self.hours_per_week = −1 if hours_per_week == "?" else int(
           hours_per_week)
        self.native_country = −1 if native_country == "?" else
```

```
            NATIVE_COUNTRY.index(native_country)

        self.label = LABEL.index(label) if not test else LABEL.index(
            label[:-1])# 最终结果标签
        self.weight = 1.0 if weight is None else weight # 处理缺失值用的
            权重

    def to_list(self):
        return [self.age, self.workclass, self.fnlwgt, self.education,
            self.education_num,
             self.marital_status, self.occupation, self.relationship,
                self.race, self.sex,
             self.capital_gain, self.capital_loss, self.hours_per_week,
                self.native_country]


class Dataset(object):
    def __init__(self, data, label, weight, attr_name):
        self.data = np.array(data, dtype=np.int_)
        self.label = np.array(label, dtype=np.int_)
        self.weight = np.array(weight)
        self.attr_name = attr_name.copy()

    def __len__(self):
        return self.data.shape[1]

    def split(self):
        entropy = self.cal_Entropy()
        if entropy == 0.0:
            return None, None, None, None

        print("Split dataset start. size: {}".format(len(self.data)))
        print("entropy {}.".format(entropy))
```

```
max_IG = -np.inf
max_attr = None
max_value = None

for attr in range(len(self)):
    # print("Calculate attribute {}.".format(attr))
    if ATTR_NAME.index(self.attr_name[attr]) not in CONT_ATTR:
        # 离散值
        cond_entropy = self.cal_CondEntropy(attr)
        # print("cond_entropy {}.".format(cond_entropy))
        if (entropy - cond_entropy) > max_IG:
            max_IG = entropy - cond_entropy
            max_attr = attr
            max_value = list(range(SEPE_ATTR_LEN[SEPE_ATTR.index
                (ATTR_NAME.index(self.attr_name[attr]))]))
    else:
        # 连续值
        cond_entropy, binary = self.cal_CondEntropy(attr)
        # print("cond_entropy {}.".format(cond_entropy))
        if (entropy - cond_entropy) > max_IG:
            max_IG = entropy - cond_entropy
            max_attr = attr
            max_value = binary

# print("Choose maximum IG attr {}.".format(max_attr))
empty_index = np.where(self.data[:,max_attr] == -1)[0]
empty_data = self.data[empty_index]
empty_label = self.label[empty_index]
empty_weight = self.weight[empty_index]

not_empty = len(self.data) - len(empty_index)

if ATTR_NAME.index(self.attr_name[max_attr]) not in CONT_ATTR:
```

```
# 离散值
split_dataset = []
split_type = ”SEPE”
split_value = max_value

for i in range(SEPE_ATTR_LEN[SEPE_ATTR.index(ATTR_NAME.index
    (self.attr_name[max_attr]))]):
    index = np.where(self.data[:,max_attr] == i)[0]
    temp_data = self.data[index]
    temp_label = self.label[index]
    temp_weight = self.weight[index]

    if not_empty > 0:
        # 缺失值赋予权重
        temp_data = np.concatenate([temp_data, empty_data])
        temp_label = np.concatenate([temp_label, empty_label
            ])
        temp_empty_weight = empty_weight * (len(index) /
            not_empty)
        temp_weight = np.concatenate([temp_weight,
            temp_empty_weight])

    # 删去已被划分的属性
    temp_data = np.delete(temp_data, max_attr, axis=1)

    # 过滤权重为0的情况
    nonzero_weight_index = np.where(temp_weight > 0.0)[0]
    temp_data = temp_data[nonzero_weight_index]
    temp_label = temp_label[nonzero_weight_index]
    temp_weight = temp_weight[nonzero_weight_index]

    attr_name = self.attr_name.copy()
    del attr_name[max_attr]
```

```
            temp_dataset = Dataset(temp_data, temp_label,
                temp_weight, attr_name)
            split_dataset.append(temp_dataset)

        return split_dataset, split_type, max_attr, split_value
else:
    # 连续值
    split_dataset = []
    split_type = "CONT"
    split_value = max_value

    not_empty_index = np.where(self.data[:,max_attr] != -1)[0]
    le_index = np.where(self.data[not_empty_index][:,max_attr]
        <= max_value)[0]
    gt_index = np.where(self.data[not_empty_index][:,max_attr] >
        max_value)[0]

    le_data = self.data[not_empty_index][le_index]
    gt_data = self.data[not_empty_index][gt_index]
    le_label = self.label[not_empty_index][le_index]
    gt_label = self.label[not_empty_index][gt_index]
    le_weight = self.weight[not_empty_index][le_index]
    gt_weight = self.weight[not_empty_index][gt_index]

    if not_empty > 0:
        le_data = np.concatenate([le_data, empty_data])
        gt_data = np.concatenate([gt_data, empty_data])
        le_label = np.concatenate([le_label, empty_label])
        gt_label = np.concatenate([gt_label, empty_label])
        le_weight = np.concatenate([le_weight, empty_weight * (
            len(le_data) / len(not_empty_index))])
        gt_weight = np.concatenate([gt_weight, empty_weight * (
                len(gt_data) / len(not_empty_index))])

            # 删去已被划分的属性
            le_data = np.delete(le_data, max_attr, axis=1)
            gt_data = np.delete(gt_data, max_attr, axis=1)

            # 过滤权重为0的情况
            le_nonzero_weight_index = np.where(le_weight > 0.0)[0]
            le_data = le_data[le_nonzero_weight_index]
            le_label = le_label[le_nonzero_weight_index]
            le_weight = le_weight[le_nonzero_weight_index]

            gt_nonzero_weight_index = np.where(gt_weight > 0.0)[0]
            gt_data = gt_data[gt_nonzero_weight_index]
            gt_label = gt_label[gt_nonzero_weight_index]
            gt_weight = gt_weight[gt_nonzero_weight_index]

            attr_name = self.attr_name.copy()
            del attr_name[max_attr]

            le_dataset = Dataset(le_data, le_label, le_weight, attr_name
                )
            split_dataset.append(le_dataset)
            gt_dataset = Dataset(gt_data, gt_label, gt_weight, attr_name
                )
            split_dataset.append(gt_dataset)

            return split_dataset, split_type, max_attr, split_value


    def cal_Entropy(self):
        res = 0.0
        for label in range(len(LABEL)):
            x = np.where(self.label == label)[0]
            if len(x) > 0:
                x = np.sum(self.weight[x])
                x = x / np.sum(self.weight)
                res -= x * math.log2(x)
        return res

    def cal_CondEntropy(self, index):
        if ATTR_NAME.index(self.attr_name[index]) not in CONT_ATTR:
            # 离散值
            res = 0.0
            # print("SEPE: total num {}.".format(str(SEPE_ATTR_LEN[
                SEPE_ATTR.index(ATTR_NAME.index(self.attr_name[index]))]])
                ))
            for i in range(SEPE_ATTR_LEN[SEPE_ATTR.index(ATTR_NAME.index
                (self.attr_name[index]))]):
                spec_index = np.where(self.data[:,index] == i)[0]
                spec_label = self.label[spec_index]
                spec_res = 0.0
                for label in range(len(LABEL)):
                    x = np.where(spec_label == label)[0]
                    if len(x) > 0:
                        x = np.sum(self.weight[spec_index][x])
                        x = x / np.sum(self.weight[spec_index])
                        spec_res -= x * math.log2(x)
                res += (len(spec_index) / len(self.data)) * spec_res
            # 缺失值不会计入熵，不需要处理，已经在计算中乘以权重
            return res
        else:
            # 连续值
            spec_index = np.where(self.data[:,index] != -1)[0]
            spec_attr = self.data[:,index][spec_index]
            spec_label = self.label[spec_index]
```

```
sepc_weight = self.weight[spec_index]

# 测试多个区间，自动排序从小到大
uniq_attr = np.unique(spec_attr)
# print("CONT: total num {}.".format(str(len(uniq_attr)-1)))
if len(uniq_attr) == 1:
    return 0.0, uniq_attr[0]

res = np.inf
binary = None
for i in range(0, len(uniq_attr)-1):
    mid = (uniq_attr[i] + uniq_attr[i+1]) / 2
    spec_res = 0.0
    le_index = np.where(spec_attr <= mid)[0]
    gt_index = np.where(spec_attr > mid)[0]

    le_label = spec_label[le_index]
    gt_label = spec_label[gt_index]

    le_res = 0.0
    gt_res = 0.0
    for label in range(len(LABEL)):
        x1 = np.where(le_label == label)[0]
        if len(x1) > 0:
            x1 = np.sum(sepc_weight[le_index][x1])
            x1 = x1 / np.sum(sepc_weight[le_index])
            le_res -= x1 * math.log2(x1)

        x2 = np.where(gt_label == label)[0]
        if len(x2) > 0:
            x2 = np.sum(sepc_weight[gt_index][x2])
            x2 = x2 / np.sum(sepc_weight[gt_index])
            gt_res -= x2 * math.log2(x2)
```

```
                spec_res += (len(le_index) / len(self.data)) * le_res
                spec_res += (len(gt_index) / len(self.data)) * gt_res

                if spec_res < res:
                    res = spec_res
                    binary = mid

            return res, binary


class DecisionTree(object):

    SPLIT_TYPE = ["CONT", "SEPE"] # 连续值，离散值

    def __init__(self, root):
        self.pos = 0
        self.tree = []
        self.is_traverse = []

        self.pos += 1
        self.tree.append(root)
        self.is_traverse.append(False)

        self.child = [] # 划分得到的子树编号
        self.split_type = [] # 划分类型连续CONT或离散SEPE
        self.split_attr = [] # 每一次划分的属性
        self.split_value = [] # 划分依据值，离散直接对应数组下标，连续对
            应二分点

        self.child.append([])
        self.split_type.append([])
        self.split_attr.append([])
        self.split_value.append([])

    def train(self):
        while True:
            traverse_index = None
            for i in range(len(self.is_traverse)):
                if not self.is_traverse[i]:
                    traverse_index = i
                    break
            if traverse_index is None:
                break
            print("Traverse tree {}.".format(traverse_index))

            self.is_traverse[traverse_index] = True

            # 叶节点，没有属性可以划分或者只剩下零/一个元素
            if len(self.tree[traverse_index].data) <= 1:
                continue
            elif len(self.tree[traverse_index]) == 0:
                continue

            split_dataset, split_type, split_attr, split_value = self.
               tree[traverse_index].split()
            if split_dataset is None:
                # 交叉熵为0，不必再分了，剪枝
                continue

            self.child[traverse_index] = list(range(self.pos, self.pos+
               len(split_dataset)))
            self.split_type[traverse_index] = split_type
            self.split_attr[traverse_index] = split_attr
            self.split_value[traverse_index] = split_value
            for dataset in split_dataset:
                self.tree.append(dataset)
                self.is_traverse.append(False)

                self.child.append([])
                self.split_type.append([])
                self.split_attr.append([])
                self.split_value.append([])

            self.pos += len(split_dataset)

    def test(self, item):
        pos = 0
        item_list = item.to_list()

        while True:
            child = self.child[pos]
            split_type = self.split_type[pos]
            split_attr = self.split_attr[pos]
            split_value = self.split_value[pos]

            # print(pos, child, split_type, split_attr, split_value,
                item_list[split_attr])

            if len(child) == 0:
                # 叶子节点且不为空节点，可以返回结果
                max_count = 0
                max_label = None
                for label in range(len(LABEL)):
                    count = np.sum(np.where(self.tree[pos].label ==
                        label))
                    if count > max_count:
                        max_count = count
                max_label = label
        return max_label == item.label

    if split_type == "SEPE":
        # 离散值
        if item_list[split_attr] == -1:
            # 缺失值，根据子节点的分布进行随机选取
            probability = []
            for c in child:
                probability.append(len(self.tree[c].data) / len(
                    self.tree[pos].data))
                if len(probability) > 1:
                    probability[-1] += probability[-2]

            random_choice = random.random()
            for p in probability:
                if random_choice < p:
                    random_index = probability.index(p)

            next_pos = child[random_index]
        else:
            next_pos = child[item_list[split_attr]]

        del item_list[split_attr]
    else:
        # 连续值
        if item_list[split_attr] == -1:
            # 缺失值，根据子节点的分布进行随机选取
            p1 = len(self.tree[child[0]].data) / len(self.tree[
                pos].data)
            p2 = len(self.tree[child[1]].data) / len(self.tree[
                pos].data)
```

```
                    random_choice = random.random()
                    if random_choice <= p1:
                        next_pos = child[0]
                    else:
                        next_pos = child[1]
                else:
                    if item_list[split_attr] <= split_value:
                        next_pos = child[0]
                    else:
                        next_pos = child[1]

                del item_list[split_attr]

            # 判定下一个位置是不是空节点，若是取当前节点的结果返回
            if len(self.tree[next_pos].data) == 0:
                # 空节点
                max_count = 0
                max_label = None
                for label in range(len(LABEL)):
                    count = np.sum(np.where(self.tree[pos].label ==
                        label))
                    if count > max_count:
                        max_count = count
                        max_label = label
                return max_label == item.label
            else:
                pos = next_pos


if __name__ == "__main__":
    train_data_path = "../adult.data"
    test_data_path = "../adult.test"
```

```
train_data = []
train_label = []
train_weight = []

line_cnt = 0
with open(train_data_path, "r") as train_data_file:
    line = train_data_file.readline()
    while line.strip():
        line_cnt += 1
        # print("read line " + str(line_cnt))

        line_split = line.strip().split(',')
        args = tuple(list(map(lambda x: x.strip(), line_split)))
        item = Item(*args)

        train_data.append(item.to_list())
        train_label.append(item.label)
        train_weight.append(item.weight)

        line = train_data_file.readline()
print("Read files done.")

root = Dataset(train_data, train_label, train_weight, ATTR_NAME)
print("Build root dataset done.")
tree = DecisionTree(root)
print("Create decision tree done.")
tree.train()
print("Train process done.")

line_cnt = 0
correct_prediction = 0
with open(test_data_path, "r") as test_data_file:
    # 丟掉第一行
    line = test_data_file.readline()
    line = test_data_file.readline()
    while line.strip():
        line_cnt += 1
        print("test read {}".format(line_cnt))

        line_split = line.strip().split(',')
        args = tuple(list(map(lambda x: x.strip(), line_split)))
        item = Item(*args, test=True)
        if tree.test(item):
            correct_prediction += 1

        line = test_data_file.readline()

print("Test Dataset Accuracy: {}%".format(correct_prediction /
   line_cnt * 100))
```