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

E10 Decision Tree

18340215 张天祎

2020 年 11 月 25 日

目录

1 Datasets **2**

2 Decision Tree **3**

2.1 ID3 3

2.2 C4.5 and CART 4

3 Tasks **5**

4 Codes and Results **5**

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 Codes and Results

```python
import math
import copy
import random
from collections import Counter

# 超参数
MAX_LEFT_DEPTH = 6

# 读入并格式化数据
def load(file_name):
    f = open(file_name,'rt')
    adult_list = f.readlines()
    adult_msg = []
    for i in adult_list:
        curr = i.split(',')
        for j in range(1, len(curr)):
            curr[j] = curr[j][1:] # 去空格
        adult_msg.append(curr)
    # 检查是否有不合规的数据，数据只有最后一行多了一个回车，测试集还有第一行不是数据
    for i in adult_msg:
```

```python
        if len(i) != 15:
            # print(i)
            adult_msg.remove(i)
    return [i[:-1] for i in adult_msg], [i[-1][:-1] for i in adult_msg] # 这里标签删去了回车符

class node:
    def __init__(self, leaf=False, label=None, attribute=None, son=[]):
        self.leaf = leaf
        self.attribute = attribute
        self.label = label
        self.son = son

    def add_son(self, son):
        if son[1] != None:
            self.son.append(son)

    # for test
    def printinf(self):
        if self.leaf:
            # print("leaf label: "+self.label, end=" ")
            print("leaf", end=" ")
        else:
            print(self.attribute, end=" ")

class DecisionTree:
    def __init__(self, train_data, train_label):
        # 预处理连续值数据
        self.con_attlist = [0, 2, 4, 10, 11, 12]
        # 预处理离散值数据
        tem_dis_list = [1, 3, 5, 6, 7, 8, 9, 13]
        self.dis_attlist = {}
        for j in tem_dis_list:
            self.dis_attlist[j] = list(set([i[j] for i in train_data]))
        # 预处理标签
        self.labelist = list(set(train_label))

    def treeLearn(self, data, label, attributes, entropy=1, option='TD3'):
```

```
        # 数据集是否空
        if data == []:
            # 数据缺失时，需要分配叶节点防止出现决策时找不到的情况
            # 策略一，随机，TD3:0.80
            return node(True, self.labelist[random.randint(0,len(self.labelist)
                -1)])
            # 策略二，硬分配一个，TD3:0.79~0.81
            # return node(True, self.labelist[0])
        # 是否全为同一标签
        same = True
        for i in label[1:]:
            if i != label[0]:
                same = False
                break
        if same:
            return node(True, label[0])
        # 是否没有属性
        if option == 'C4.5':
            # if len(attributes) < 7: # 同ID3
            if len(attributes) < MAX_LEFT_DEPTH:
                return node(True, Counter(label).most_common(1)[0][0])
        else: # TD3变为是否无离散值属性:
            flag = False
            for i in attributes:
                if i in self.dis_attlist:
                    flag = True
            if not flag:
                return node(True, Counter(label).most_common(1)[0][0])
        # 一般情况
        Entropys = [(i, self.calEntropy(data, label, i, entropy, option)) for i
            in attributes]
        if option == 'C4.5': # C4.5 比较增益率
            pickatt, entropy = max(Entropys, key=lambda x: x[1][2])
        else: # 比较信息熵
            pickatt, entropy = min(Entropys, key=lambda x: x[1][0])
        mid_val = entropy[1]
        curr_node = copy.deepcopy(node(False, attribute=pickatt)) # 这里有个，需要
            深复制bug
```

```
        par_map = self.partition(data, label, pickatt, mid_val)
        tem_attributes = copy.deepcopy(attributes)
        tem_attributes.remove(pickatt)
        for i in par_map:
            curr_node.add_son((i, self.treeLearn(par_map[i][0], par_map[i][1],
                tem_attributes, entropy[0], option)))
        return curr_node

    def calEntropy(self, data, label, attribute, fa_entropy, option='TD3'):
        att_data = [i[attribute] for i in data]
        length = len(data)
        true_label = self.labelist[0]
        sum = 0
        IV = 0
        if attribute in self.con_attlist:
            # print(option)
            if option=='C4.5':
                # 二分法
                att_data = [int(i) for i in att_data]
                max_val = max(att_data)
                min_val = max_val
                for i in att_data:
                    # 考虑到该数据集中有大量无用的，做了一点优化，效果不大0
                    if i != 0 and i < min_val:
                        min_val = i
                mid_val = (max_val+min_val)//2
                true_count = false_count = [0, 0]
                for i in zip(att_data, label):
                    ind = i[0] > mid_val
                    if i[1] == true_label:
                        true_count[ind] += 1
                    else:
                        false_count[ind] += 1
                for i in range(2):
                    ci = true_count[i] + false_count[i]
                    pi = ci / length
                    if true_count[i]:  # 防止出现数学错，这里不会影响熵的结果
```

```
                        px = true_count[i] / ci
                        sum += pi * (-px * math.log(px))
                IV -= pi * math.log2(pi) if pi != 0 else 0
                gain_ratio = (fa_entropy - sum) / IV if IV != 0 else 2 # IV 小是较好的分类
                return sum, mid_val, gain_ratio # 2 这里只是一个相对较大的数
            # td3 下连续值属性不考虑，直接返回一个较大的熵 2
            return 2, None, None
        else:
            true_count = dict([(i, 0) for i in self.dis_attlist[attribute]])
            false_count = dict([(i, 0) for i in self.dis_attlist[attribute]])
            for i in zip(att_data, label):
                if i[1] == true_label:
                    true_count[i[0]] += 1
                else:
                    false_count[i[0]] += 1
            for i in self.dis_attlist[attribute]:
                ci = true_count[i] + false_count[i]
                pi = ci / length
                if true_count[i]: # 防止出现数学错，这里不会影响熵的结果
                    px = true_count[i] / ci
                    sum += pi * (-px*math.log2(px))
                IV -= pi * math.log2(pi) if pi != 0 else 0
                gain_ratio = (fa_entropy - sum) / IV if IV != 0 else 2
            return sum, None, gain_ratio

    def partition(self, data, label, attribute, mid_val):
        par_map = {}
        if attribute in self.dis_attlist:
            for i in self.dis_attlist[attribute]:
                par_map[i] = [[], []]
            for i in range(len(data)):
                par_map[data[i][attribute]][0].append(data[i])
                par_map[data[i][attribute]][1].append(label[i])
        else:
            ind1 = ">"+str(mid_val)
            ind2 = "<="+str(mid_val)
```

```
                par_map[ind1] = [[], []]
                par_map[ind2] = [[], []]
                for i in range(len(data)):
                    ind = ind1 if int(data[i][attribute])>mid_val else ind2
                    par_map[ind][0].append(data[i])
                    par_map[ind][1].append(label[i])
            return par_map

# 没有考虑测试集出现训练集中未出现值的情况
    def test(self, root, test_data):
        rst = []
        for i in test_data:
            curr = root
            while(not curr.leaf):
                att = curr.attribute
                val = i[att]
                if att in self.dis_attlist:
                    for j in curr.son:
                        if j[0] == val:
                            curr = j[1]
                            break
                else:
                    num = int(curr.son[0][0][1:])
                    if int(val)>num:
                        curr = curr.son[0][1]
                    else:
                        curr = curr.son[1][1]
            rst.append(curr.label)
        return rst

    # for test
    def printinf(self,root):
        print("root:",end=" ")
        que = []
        que.append(root)
        while(len(que)):
            tem = []
```

```
            while(len(que)):
                curr = que[0]
                que.pop(0)
                curr.printinf()
                if not curr.leaf:
                    # print(curr.attribute, end=" ")
                    for i in curr.son:
                        tem.append(i[1])
                # else:
                    # print("leaf",end=" ")
            que = tem
            print("")

def main():
    # pre_train
    train_data, train_label = load('adult.data')
    test_data, test_label = load('adult.test')
    test_label = [i[:-1] for i in test_label] # test_label 后面多了一个'.'
    tree = DecisionTree(train_data, train_label)

    # train
    root_1 = tree.treeLearn(train_data, train_label, list(range(14)))
    root_2 = tree.treeLearn(train_data, train_label, list(range(14)),option='C4.5')

    # test
    # DecisionTree.printinf(root)
    rst_label1 = tree.test(root_1, test_data)
    rst_label2 = tree.test(root_2, test_data)
    testlen = len(test_label)
    count1 = count2 = 0
    for i in range(testlen):
        if rst_label1[i] == test_label[i]:
            count1 += 1
        if rst_label2[i] == test_label[i]:
            count2 += 1
    print("  ID3 accuracy: "+str(count1/testlen*100)+"%")
```

```
    print(" C4.5 accuracy: "+str(count2/testlen*100)+"%")
    print("with MAX_DEPTH:", 14 - MAX_LEFT_DEPTH)

if __name__ == "__main__":
    main()
```

结果如下：

```
  ID3 accuracy: 80.38204041520791%
 C4.5 accuracy: 81.46919722375775%
with MAX_DEPTH: 8
```

- ID3 只使用了离散型的属性，效果也还不错。
- C4.5 相比 ID3 有很多改进，但总体上是类似的，改进难度不大。(1) 使用了连续型的属性，在不考虑大量无意义的 0 的前提下使用了二分的方法进行分割。(2) 因为连续属性二分的熵增益在数学上显然比离散属性的熵增益要低，为了合理的评估属性选取，引入（熵）增益率的概念，以此为基础进行属性选取。(3) 树太深很容易产生过拟合，需要人为设定超参数树深。
- CART 是一棵二叉树，代码修改量比较大，这里没有再做尝试。