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

E10 Decision Tree

18340013 Conghao Chen

November 18, 2020

Contents

1 Datasets **2**

2 Decision Tree **3**

2.1 ID3 3

2.2 C4.5 and CART 4

3 Tasks **5**

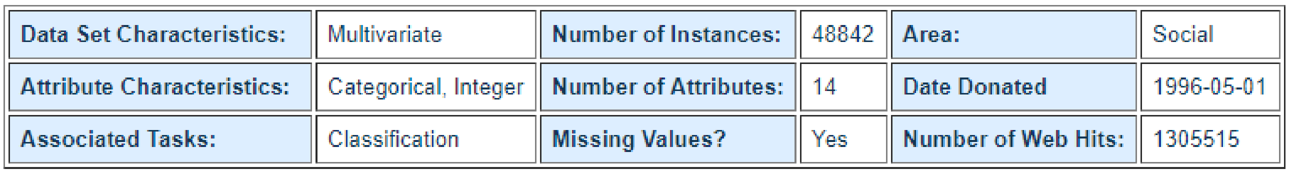
4 Results **5**

5 Codes **6**

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.

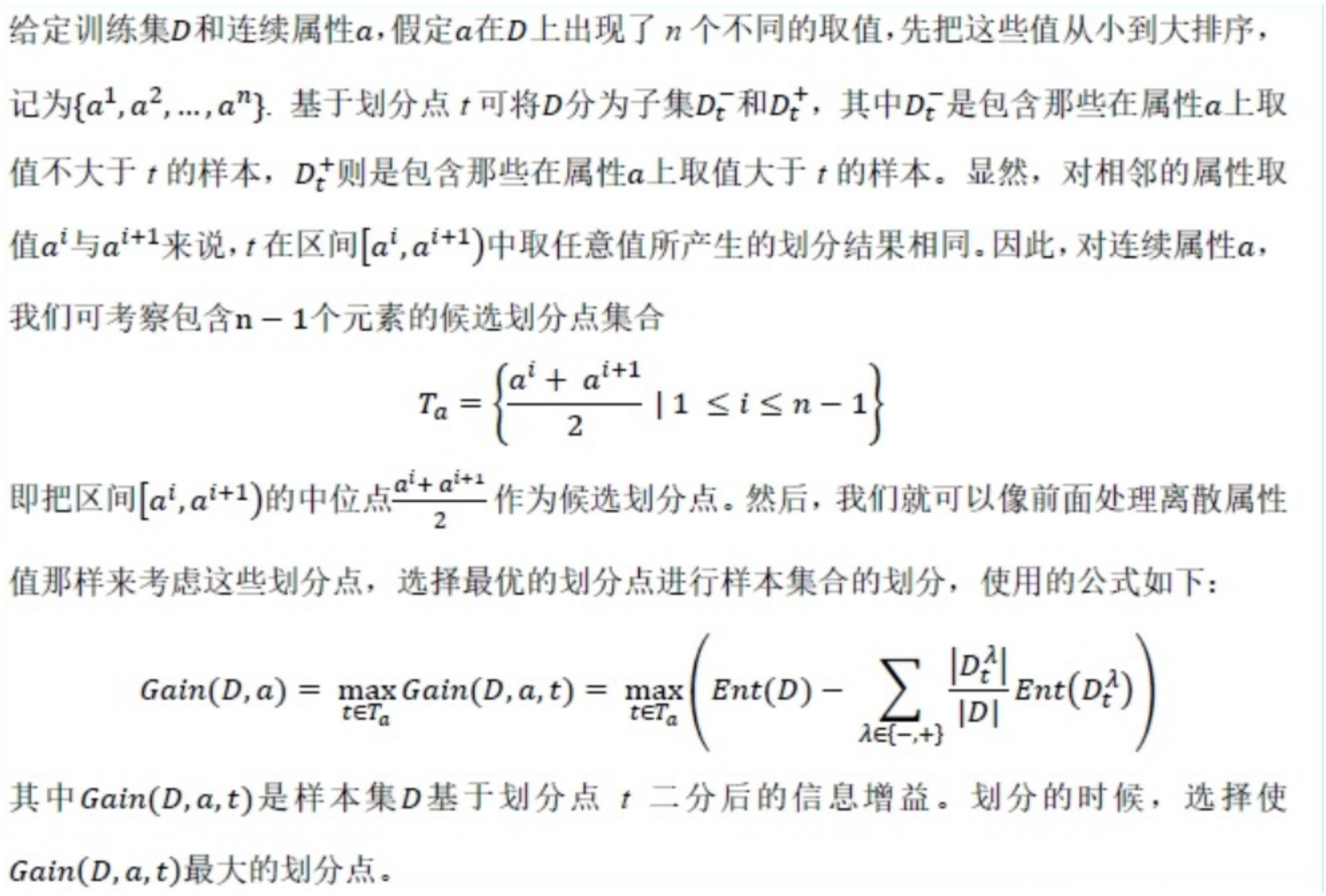
CART (Classification and Regression Trees) is very similar to C4.5, but it differs in that it supports numerical target variables (regression) and does not compute rule sets. CART constructs binary trees using the feature and threshold that yield the largest information gain at each node.

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 Results

本次代码的实现参考了西瓜书上对于 ID3 决策树的介绍；对于丢失的数据，我采取的方法是用这一列的众数将其填进去；对于连续的数据，使用的是二分法，根据查找的资料来实现的代码，算法思路如下图：



给定训练集D和连续属性a，假定a在D上出现了 n 个不同的取值，先把这些值从小到大排序，记为$\{a^1, a^2, \dots, a^n\}$. 基于划分点 t 可将D分为子集D_t^-和D_t^+，其中D_t^-是包含那些在属性a上取值不大于 t 的样本，D_t^+则是包含那些在属性a上取值大于 t 的样本。显然，对相邻的属性取值a^i与a^{i+1}来说，t 在区间$[a^i, a^{i+1})$中取任意值所产生的划分结果相同。因此，对连续属性a，我们可考察包含$\mathbf{n-1}$个元素的候选划分点集合

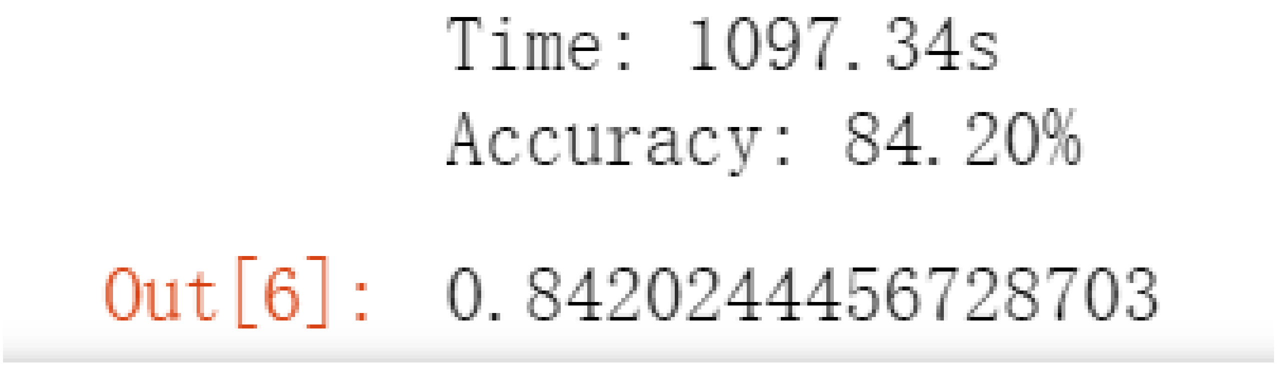
$$T_a = \left\{ \frac{a^i + a^{i+1}}{2} \mid 1 \leq i \leq n-1 \right\}$$

即把区间$[a^i, a^{i+1})$的中位点$\frac{a^i + a^{i+1}}{2}$作为候选划分点。然后，我们就可以像前面处理离散属性值那样来考虑这些划分点，选择最优的划分点进行样本集合的划分，使用的公式如下：

$$Gain(D,a) = \max_{t \in T_a} Gain(D,a,t) = \max_{t \in T_a} \left(Ent(D) - \sum_{\lambda \in \{-,+\}} \frac{|D_t^\lambda|}{|D|} Ent(D_t^\lambda) \right)$$

其中$Gain(D,a,t)$是样本集D基于划分点 t 二分后的信息增益。划分的时候，选择使$Gain(D,a,t)$最大的划分点。

经过长时间的等待后终于训练完成, 在 log 文件里可以看到整个过程, 最终准确度为 84.20%：



```
Time: 1097.34s
Accuracy: 84.20%

Out[6]: 0.8420244456728703
```

5 Codes

代码是在 Jupyter 下完成，所以先导出.py 文件，代码如下：

```
# coding: utf-8
# In[1]:
import logging
```

```
logger = logging.getLogger(__name__)
logger.setLevel(level = logging.INFO)
handler = logging.FileHandler("DT-prepruning.log")
handler.setLevel(logging.INFO)
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s -
    %(message)s')
handler.setFormatter(formatter)
logger.addHandler(handler)
logger.info("Depth 5, no shuffle, 0.9 train data")



# In[2]:
import pandas as pd
import numpy as np

attrlist = {"age":0, "workclass":1, "fnlwgt":0, "education":1,
    "education-num":0, "marital-status":1, "occupation":1,
    "relationship":1, "race":1, "sex":1, "capital-gain":0,
    "capital-loss":0, "hours-per-week":0, "native-country":1,
    "salary":0} # 0: continuous, 1: discrete

traindata = pd.read_csv("../data/adult/adult.data",
    names=attrlist.keys(), index_col=False)
testdata = pd.read_csv("../data/adult/adult.test",
    names=attrlist.keys(),
index_col=False, header=0)

def preprocessing(data):
    attributes = list(attrlist.keys())
    return data[attributes]

def fulldata(data):
    #填充缺失数据，方法是选择该列出现最多的数据填入该位置。
    for a in attrlist:
        if attrlist[a]: # 离散
            data.loc[data[a] == " ?",a] =
               data[a].value_counts().argmax()
        else: # 连续
            pass

# 得到数据集
traindata = preprocessing(traindata)
testdata = preprocessing(testdata)
fulldata(traindata)
fulldata(testdata)

# 9: 1分配训练集与验证集
breakpoint = int(0.9 * len(traindata))
# breakpoint = int(len(traindata))
traindata, valdata = traindata[:breakpoint], traindata[breakpoint:]



# In[3]:
def entropy(p):
    #计算熵
    if p.ndim == 1:
        nextp = p[p != 0]
        return -np.sum(nextp * np.log2(nextp))
    else:
        return -np.sum(p * np.log2(p),axis=1)

def information_gain(D,a,discrete_flag=False):
    #计算信息增益
    pk = D["salary"].value_counts(normalize=True).values
    if discrete_flag: # 离散
```

```
        perDv = D[a].value_counts(normalize=True).values
        probDv = np.array([D.loc[D[a] ==
            av]["salary"].value_counts(normalize=True).get(" >50K",0)
            for av in D[a].unique()])
        hstack = np.column_stack((perDv,probDv))
        hstack = hstack[(hstack[:,1] != 0) & (hstack[:,1] != 1)]
        perDv = hstack[:,0]
        probDv = hstack[:,1]
        probDv_not = 1 - probDv
        return (entropy(pk) - np.sum(perDv *
            entropy(np.column_stack((probDv,probDv_not))))), a)
    else: # 连续
        newa = sorted(D[a].unique())
        T_a = [(newa[i] + newa[i+1]) / 2 for i in range(len(newa)-1)]
        entmin, tmin = 0x3f3f3f3f, newa[0]
        for t in T_a:
            perDv = len(D[D[a] < t]) / len(D)
            perDv = np.array([perDv,1-perDv])
            probDv_smaller = D[D[a] <
                t]["salary"].value_counts(normalize=True).get(" >50K",0)
            probDv_bigger = D[D[a] >=
                t]["salary"].value_counts(normalize=True).get(" >50K",0)
            probDv = np.array([[probDv_smaller,1-probDv_smaller],
            [probDv_bigger,1-probDv_bigger]])
            probDv = probDv[(probDv[:,0] != 0) & (probDv[:,1] != 0)]
            if len(probDv) == 0:
                sumup = 0
            else:
                sumup = np.sum(perDv * entropy(probDv))
            if entmin > sumup:
                entmin = sumup
                tmin = t
        return (entropy(pk) - entmin, tmin)
```

```
# In[4]:
class Node:

    def __init__(self):
        self.branch = {}

    def setLeaf(self,catagory,cnt=1):
        logger.info("{} - Create leaf: {}".format(cnt,catagory))
        if cnt % 10 == 0:
            print("{} - Create leaf:
                {}".format(cnt,catagory),flush=True)
        self.label = "Leaf"
        self.catagory = catagory

    def setBranch(self,attr,value,node,branch_value=None):
        logger.info("Create branch: {} ({})".format(attr,value))
        self.label = "Branch"
        self.attr = attr
        self.branch[value] = node
        if branch_value != None:
            self.branch_value = branch_value


# In[5]:
import time,sys

class ID3:
    # ID3算法实现
    def __init__(self,trainset=None,validationset=None,testset=None,
    attrlist=None):
        self.trainset = trainset
        self.validationset = validationset
        self.testset = testset
        self.attrlist = attrlist

    def
        TreeGenerate(self, dataset, attributes, depth, cnt_leaves=0, root=None):
        catagory = dataset["salary"].unique()
        node = Node() if root == None else root
        cnt_leaves += 1

        # 1) All samples in `dataset` belongs to the same catagory
        if len(catagory) == 1:
            node.setLeaf(catagory[0], cnt_leaves)
            return node

        # 2) `attributes` is empty, or the values of `dataset` on
            `attributes` are the same
        if len(attributes) == 0 or np.array([len(dataset[a].unique())
            == 1 for a in attributes]).all() == True:
            node.setLeaf(dataset["salary"].value_counts().argmax(),
            cnt_leaves)
            return node

        # without partition
        node.setLeaf(dataset["salary"].value_counts().argmax(), cnt_leaves)
        accuracy_without_partition = self.validation()

        # with partition
        # 找带来最大增益的属性
        max_gain = (-0x3f3f3f3f, None)
        for a in attributes:
            gain = information_gain(dataset, a, self.attrlist[a])
            if gain[0] > max_gain[0]:
```

```
            aright , max_gain = a, gain
    num_leaves = 0
    if self.attrlist[aright]: # 离散
        num_leaves = len(self.trainset[aright].unique())
        for av in self.trainset[aright].unique():
            Dv = dataset[dataset[aright] == av]
            cnt_leaves += 1
            leafnode = Node()
            if len(Dv) == 0:
                leafnode.setLeaf
                (dataset["salary"].value_counts().argmax(),cnt_leaves)
            else:
                leafnode.setLeaf
                (Dv["salary"].value_counts().argmax(),cnt_leaves)
            node.setBranch(aright ,av,leafnode)
    else: # 连续
        num_leaves = 2
        for flag in ["Smaller","Bigger"]:
            Dv = dataset[dataset[aright] < max_gain[1]] if flag ==
               "Smaller" else dataset[dataset[aright] >=
               max_gain[1]]
            cnt_leaves += 1
            leafnode = Node()
            if len(Dv) == 0:
                leafnode.setLeaf
                (dataset["salary"].value_counts().argmax(),cnt_leaves)
            else:
                leafnode.setLeaf
                (Dv["salary"].value_counts().argmax(),cnt_leaves)
            node.setBranch(aright ,flag ,leafnode ,
            branch_value=max_gain[1])
    accuracy_with_partition = self.validation()
```

```
        # prepruning
        if depth > 5 and accuracy_without_partition >=
            accuracy_with_partition:
            cnt_leaves -= num_leaves
            print("Prune at {}: {} (without) >= {}
                (with)".format(aright,accuracy_without_partition,
            accuracy_with_partition))
            logger.info("Prune at {}: {} (without) >= {}
                (with)".format(aright,accuracy_without_partition,
            accuracy_with_partition))
            node.setLeaf(dataset["salary"].value_counts().argmax())
            return node
        elif depth > 5:
            print(aright,accuracy_without_partition,
            accuracy_with_partition)

        if self.attrlist[aright]: # 离散
            for av in self.trainset[aright].unique():
                Dv = dataset[dataset[aright] == av]
                # 3) `Dv` is empty, which can not be partitioned
                if len(Dv) != 0:
                    node.setBranch(aright,av,self.TreeGenerate(Dv,
                    attributes[attributes !=
                        aright],depth+1,cnt_leaves))
        else: # 连续
            for flag in ["Smaller","Bigger"]:
                Dv = dataset[dataset[aright] < max_gain[1]] if flag ==
                    "Smaller" else dataset[dataset[aright] >=
                    max_gain[1]]
                if len(Dv) != 0:
                    node.setBranch(aright,flag,self.TreeGenerate(Dv,
                    attributes,depth+1,cnt_leaves),
                    branch_value=max_gain[1])
        return node

    def train(self,trainset=None):
        # 训练
        if trainset != None:
            self.trainset = trainset
        start_time = time.time()
        self.root = Node()
        self.root =
            self.TreeGenerate(self.trainset,self.trainset.columns.values
        [self.trainset.columns.values !=
            "salary"],depth=1,root=self.root,
        cnt_leaves=0)
        logger.info("Time: {:.2f}s".format(time.time()-start_time))
        print("Time: {:.2f}s".format(time.time()-start_time))

    def validation(self,validationset=None):
        # 验证
        if validationset != None:
            self.validationset = validationset
        accuracy = 0
        for i,row in self.validationset.iterrows():
            p = self.root
            while p.label != "Leaf":
                if self.attrlist[p.attr]: # 离散
                    p = p.branch[row[p.attr]]
                else: # 连续
                    p = p.branch["Smaller"] if row[p.attr] <
                        p.branch_value else p.branch["Bigger"]
            if p.catagory == row["salary"]:
                accuracy += 1
        accuracy /= len(self.validationset)
        return accuracy
    def test(self,testset=None):
        # 测试
        if testset != None:
            self.testset = testset
        accuracy = 0
        for i,row in self.testset.iterrows():
            p = self.root
            while p.label != "Leaf":
                if self.attrlist[p.attr]: # 离散
                    p = p.branch[row[p.attr]]
                else: # 连续
                    p = p.branch["Smaller"] if row[p.attr] <
                        p.branch_value else p.branch["Bigger"]
            if p.catagory == row["salary"][:-1]:
                accuracy += 1
        accuracy /= len(self.testset)
        logger.info("Accuracy: {:.2f}%".format(accuracy * 100))
        print("Accuracy: {:.2f}%".format(accuracy * 100))
        return accuracy


# In[6]:
dt = ID3(trainset=traindata,validationset=valdata,testset=testdata,
attrlist=attrlist)
dt.train()
dt.test()
```