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

E11 Decision Tree

19214808 Yikun Liang

2019 年 11 月 24 日

目录

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`.

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 FF precondition if the accuracy of the rule improves without it.

C5.0 is Quinlan FF 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 `E11_YourNumber.pdf`, and send it to `ai_201901@foxmail.com`

4 Codes and Results

Codes mainly contain four parts:

- Pretreatment
- Create Dataset
- Build up decision tree model
- Test accuracy on test datas

The first part is pretreatment. This part deals with origin datas , replaces unknown datas and descretizes datas.

```
import pandas as pd
import numpy as np

#数据预处理

def createDataSet(path):
    matrix =pd.read_csv(path,header = None )
    matrix = matrix.as_matrix().tolist()
    dataset = []
    labels = ["age", "workclass", "fnlwgt", 'education', 'education-num', \
              'marital-status', 'occupation', 'relationship', 'race', 'sex', \
              'capital-gain', 'capital-loss', 'hours-per-week', 'native-country']
    for line in matrix:
        dataset.append(line)
    return dataset,labels
```

```
#第一步构建初始数据集
myData, labels = createDataSet("adult.data")
test_data, test_labels = createDataSet("adult.test")


#然后获取每列最频繁出现的元素，用于填补？的空缺
def get_most_frequent(dataset):
    listDict = [{} for i in range(15)]

    for row in dataset:
        for i in range(15):
            if row[i] not in listDict[i].keys():
                listDict[i][row[i]] = 1
            else:
                listDict[i][row[i]] += 1
    most_frequent_list = []
    for i in range(15):
        maxcount = 0
        most_frequent = ""
        for key, count in listDict[i].items():
            if (count > maxcount):
                maxcount = count
                most_frequent = key
        most_frequent_list.append(most_frequent)
    return most_frequent_list


train_most_frequent_list = get_most_frequent(myData)
test_most_frequent_list = get_most_frequent(test_data)

#将所有？替换为最经常出现的项
def replace_unknown_data(dataset,most_frequent_list):
    for row in dataset:
        for i in range(15):
            if(str(row[i]).find("?") != -1):
                #print("haha,find you!",row)
                row[i] = most_frequent_list[i]


replace_unknown_data(myData,train_most_frequent_list)
replace_unknown_data(test_data,test_most_frequent_list)
```

```
#此函数将连续的变量离散化。label表示要离散化的列，classnum表明了分类的个数，默认为5.
def dealData(dataset,label, classnum = 5):
    import copy
    newdataset = copy.deepcopy(dataset)
    list = [dataset[i][label]for i in range(len(dataset))]
    newlist = sorted(list)
    length = len(list)
    each = length // classnum

    for i, num in enumerate(list):
        ind = newlist.index(num)
        list[i] = ind // each
    for i in range(length):
        newdataset[i][label] = list[i]
    return newdataset
    #fnlwgts[]


#此函数对之前的数据集所有连续变量进行离散化，并将最终结果保存到filename的位置。
def preDeal(dataset,filename):
    print(dataset[:10])
    ageData = dealData(dataset,0,5)
    fnlwgtData = dealData(ageData,2,5)
    edunumData = dealData(fnlwgtData,4,5)
    capgainData = dealData(edunumData,10,5)
    caplossData = dealData(capgainData,11,5)
    hoursData =  dealData(caplossData,12,5)
    myData = hoursData
    #myData = ageData
    print(myData[:10])
    import csv
    with open (filename,'w',newline= '')as f:
        writer = csv.writer(f)
        for line in myData:
            writer.writerow(line)
        f.close()

preDeal(myData,"adult_after_deal.data")
preDeal(test_data,"adult_after_deal.test")
```

The Second part is creating Dataset. This part creates dataset used for training model. The part also includes several operation on datasets, written by function in the code.

```
import pandas as pd
import numpy as np

#构建经过初始化的数据集
def createDataSet(path):
    matrix =pd.read_csv(path,header = None )
    matrix = matrix.as_matrix().tolist()
    dataset = []
    labels = ["age", "workclass", "fnlwgt", 'education', 'education-num', \
              'marital-status', 'occupation', 'relationship', 'race', 'sex', \
              'capital-gain', 'capital-loss', 'hours-per-week', 'native-country']
    for line in matrix:
        dataset.append(line)
    return dataset,labels

#train_data, train_labels = createDataSet("adult.data")
test_data, test_labels = createDataSet("adult_after_deal.test")
myData, labels = createDataSet("adult_after_deal.data")

#myData = myData[5200:20000]


from math import log
#calc ent

#对于一个数据集，计算其信息熵
def calcEnt(dataSet):
    num = len(dataSet)
    labelcount = {}
    for vec in dataSet:
        Lable = vec[-1]
        if Lable not in labelcount.keys():
            labelcount[Lable] =  0
        labelcount[Lable] += 1
    ent = 0.
    for key in labelcount.keys():
        prob = float(labelcount[key]/num)    #计算key的出现概率得到p(t)
```

```
        ent -= prob * log(prob,2)        #熵是 -p(t)*log(p(t))的累加值
    return ent


#返回数据集根据第几个特征axis值为value的分类结果
def splitData(dataSet,axis, value):
    newDataSet = []
    for vec in dataSet:
        if (vec[axis]==value):
            reduceVec = vec[:axis]
            reduceVec.extend(vec[axis+1:])
            newDataSet.append(reduceVec)
    return newDataSet

#根据各个特征选择所得到的增益信息熵得到最佳的划分数据集的特征
def chooseBestSplit(dataset):
    numAxis = len(dataset[0]) - 1
    originEnt = calcEnt(dataset)
    bestAxis = -1
    bestinformgain = 0
    for i in range(numAxis):
        axisList = [axis[i] for axis in dataset]
        uniqvals = set(axisList)
        newEnt = 0
        for val in uniqvals: #对每一个特征进行计算
            subDataset = splitData(dataset,i,val)
            prob = len(subDataset)/float(len(dataset)) #按此特征分类的对应子树概率
            newEnt += prob *calcEnt(subDataset) #对各个子数据集的信息熵进行求和
        informGain = originEnt - newEnt #增益信息熵
        if(informGain>bestinformgain):
            bestinformgain = informGain
            bestAxis = i
    return bestAxis
```

The third part is building up decision tree model.

```
import numpy as np
import pandas
from create_dataset import *

#define buildtree 需要注释
#decision tree learning : find the most important attribute first, then classify and
                                                create a dictionary.


def majorClass(classlist): #返回出现最多的一个分类
    classCount = {}
    for i in classlist:
        if(i not in classCount.keys()):
            classCount[i] = 1
        else:
            classCount[i] += 1
    maxnum = 0
    ans = 0
    for i in classCount.keys():
        if(classCount[i]>maxnum):
            maxnum = classCount[i]
            ans = i
    return ans

'''
originlabels = ["age", "workclass", "fnlwgt", 'education', 'education-num', \
               'marital-status', 'occupation', 'relationship', 'race', 'sex', \
               'capital-gain', 'capital-loss', 'hours-per-week', 'native-country']
'''

#生成决策树函数
def decision_tree_learning(dataset,labels,depth):
    classlist = [example[-1] for example in dataset]
    # 1.类别全部一致
    if(len(set(classlist))==1):
        return classlist[0]

    # 2.labels为空，此时直接返回出现最多的类
    if(len(labels) == 0):
        return majorClass(classlist)
```

```
    # 3.dataset只剩余单个列项，而数据取值不尽相同，需要投票返回出现最多的类
    if(len(dataset[0])==1):
        return majorClass(classlist)

    ##以上是终止条件，下面构造树

    A = chooseBestSplit(dataset)          #选择最大信息熵的特征，返回特征编号
    bestLabel = labels[A]
    tree  = {bestLabel:{}}                #初始化结果树，下面将递归构建
    valuelistA = [axis[A] for axis in dataset]
    del (labels[A]) #labels中删除此项选出的特征
    valuelistA = set(valuelistA)
    for value in valuelistA:
        subdataset = splitData(dataset,A,value)      #根据该特征将数据集划分为子数据集，
                                                     从而构建子树
        sublabels = labels[:]
        subtree = decision_tree_learning(subdataset,sublabels,depth+1) #对所有子节点进行
                                                     递归调用，产生子树
        tree[bestLabel][value] = subtree     #将决策树的此特征对应的值赋值为子节点
    return tree


tree = decision_tree_learning(myData,labels,0)
print(tree)


#树的存储
def storeTree(tree, filename):
    import pickle
    fw = open(filename,'wb')
    pickle.dump(tree,fw)
    fw.close()
storeTree(tree,"classifier2.txt")



#树的输出
# dict example {'no surfacing': {0: 'no', 1: {'flippers': {0: 'no', 1: 'yes'}}}}
def printtree(tree):
    bestaxis = list(tree.keys())[0]
```

```
    print(bestaxis)
    nextdict = tree[bestaxis]
    for key in nextdict.keys():
        if(type(nextdict[key]).__name__ =="dict" ):
            printtree(nextdict[key])
        else:
            print (key,nextdict[key])

#printtree(tree)

#draw the tree
import drawtree
from drawtree import createPlot

#createPlot(tree)
```

The fourth part is testing accuracy on test datas.

```
from create_dataset import createDataSet
import random

def getTree(filename):
    import pickle
    fr  = open(filename,'rb')
    return pickle.load(fr)

#获取决策树
tree = getTree("classifier2.txt")

#决策树分类函数，接收一个vector，预测它的分类
def classify(tree,labels,testData):  # 决策树，labels，测试向量
    bestAxis = list(tree.keys())[0] #获取树的第一个分类属性（即最佳分类特征）
    nextDict = tree[bestAxis]   #获取这个分类属性的字典
    index = labels.index(bestAxis)  #获取这个属性在labels中的位置
    for key in nextDict.keys():
        if(testData[index] == key):
            #递归进行classify操作,如果递归对象是字典，继续递归，否则直接返回分类。
            if(type(nextDict[key]).__name__ =='dict'):
                return classify(nextDict[key],labels,testData)
```

```
            else:
                return nextDict[key]
    #set default value
    return random.choice([" <=50K"," >50K"])

test_data, test_labels = createDataSet("adult_after_deal.test")

#测试集答案
test_ans = [test_data[i][-1] for i in range(len(test_data))]
#print(test_ans)

acc = 0

for i,vec in enumerate(test_data):
    t = classify(tree, test_labels, vec)
    acc += (t == test_ans[i])

acc = acc/len(test_data)
print("acc:",acc)
```

And here is the test result:

图 1: Acc Result

And there is also an extra service: drawing out the trees. Codes are as follow:

```
import matplotlib.pyplot as plt

from pylab import *
mpl.rcParams['font.sans-serif'] = ['SimHei'] #否则中文无法正常显示

decisionNode=dict(boxstyle='sawtooth',fc='0.8') #决策点样式
leafNode=dict(boxstyle='round4',fc='0.8') #叶节点样式
arrow_args=dict(arrowstyle='<-') #箭头样式
```

```
def plotNode(nodeTxt,centerPt,parentPt,nodeType):
    createPlot.ax1.annotate(nodeTxt,xy=parentPt,xycoords='axes fraction',
                            xytext=centerPt,textcoords='axes fraction',
                            va='center',ha='center',bbox=nodeType,arrowprops=arrow_args)

def createPlot():
    fig=plt.figure(1,facecolor='white')
    fig.clf()
    createPlot.ax1=plt.subplot(111,frameon=False)
    plotNode('决策节点',(0.5,0.1),(0.1,0.5),decisionNode)
    plotNode('叶节点',(0.8,0.1),(0.3,0.8),leafNode)
    plt.show()

#测试
#获取叶节点数量（广度）
def getNumLeafs(myTree):
    numLeafs=0
    firstStr=list(myTree.keys())[0]#'dict_keys' object does not support indexing
    secondDict=myTree[firstStr]
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':
            numLeafs+=getNumLeafs(secondDict[key])
        else:numLeafs+=1
    return numLeafs

#获取树的深度的函数（深度）
def getTreeDepth(myTree):
    maxDepth=0
    firstStr=list(myTree.keys())[0]
    secondDict=myTree[firstStr]
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':
            thisDepth=1+getTreeDepth(secondDict[key])
        else: thisDepth=1
        if thisDepth > maxDepth:
            maxDepth=thisDepth
    return maxDepth
#定义一个预先创建树的函数
def retrieveTree(i):
    listOfTrees=[{'no surfacing': {0: 'no', 1: {'flippers': {0: 'no', 1: 'yes'}}}},
```

```
                {'no surfacing': {0: 'no', 1: {'flippers': {0: {'head':{0:'no', 1: 'yes
                                                                '}},1:'no'}}}}
                ]
    return listOfTrees[i]

#定义在父子节点之间填充文本信息的函数
def plotMidText(cntrPt,parentPt,txtString):
    xMid=(parentPt[0]-cntrPt[0])/2+cntrPt[0]
    yMid=(parentPt[1]-cntrPt[1])/2+cntrPt[1]
    createPlot.ax1.text(xMid,yMid,txtString)

#定义树绘制的函数
def plotTree(myTree,parentPt,nodeTxt):
    numLeafs=getNumLeafs(myTree)
    depth=getTreeDepth(myTree)
    firstStr=list(myTree.keys())[0]
    cntrPt=(plotTree.xOff+(1.0+float(numLeafs))/2/plotTree.totalW,plotTree.yOff)
    plotMidText(cntrPt,parentPt,nodeTxt)
    plotNode(firstStr,cntrPt,parentPt,decisionNode)
    secondDict=myTree[firstStr]
    plotTree.yOff=plotTree.yOff -1/plotTree.totalD
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':
            plotTree(secondDict[key],cntrPt,str(key))
        else:
            plotTree.xOff=plotTree.xOff+1.0/plotTree.totalW
            plotNode(secondDict[key],(plotTree.xOff,plotTree.yOff),cntrPt,leafNode)
            plotMidText((plotTree.xOff,plotTree.yOff),cntrPt,str(key))
    plotTree.yOff=plotTree.yOff+1/plotTree.totalD

 #定义主函数，来调用其它函数
def createPlot(inTree):
    fig=plt.figure(1,facecolor='white')
    fig.clf()
    axprops=dict(xticks=[],yticks=[])
    createPlot.ax1=plt.subplot(111,frameon=False,**axprops)
    plotTree.totalW=float(getNumLeafs(inTree))
    plotTree.totalD=float(getTreeDepth(inTree))
    plotTree.xOff=-0.5/plotTree.totalW;plotTree.yOff=1.0;
    plotTree(inTree,(0.5,1.0),'')
    plt.show()
```