# 文档模板

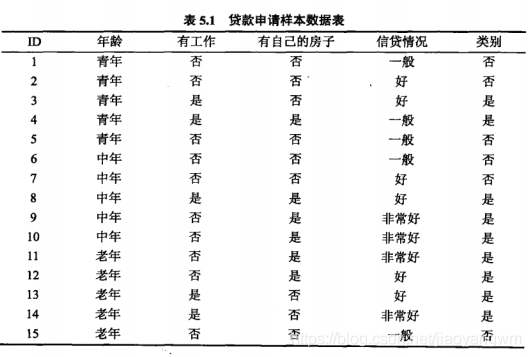
1. 模型概述
2. 概念描述

决策树(Decision Tree）是在已知各种情况发生概率的基础上，通过构成决策树来求取净现值的期望值大于等于零的概率，评价项目风险，判断其可行性的决策分析方法，是直观运用概率分析的一种图解法。由于这种决策分支画成图形很像一棵树的枝干，故称决策树。在机器学习中，决策树是一个预测模型，他代表的是对象属性与对象值之间的一种映射关系。Entropy = 系统的凌乱程度，使用算法ID3, C4.5和C5.0生成树算法使用熵。这一度量是基于信息学理论中熵的概念。

分类树（决策树）是一种十分常用的分类方法。它是一种监督学习，所谓监督学习就是给定一堆样本，每个样本都有一组属性和一个类别，这些类别是事先确定的，那么通过学习得到一个分类器，这个分类器能够对新出现的对象给出正确的分类。这样的机器学习就被称之为监督学习。

1. 公式描述

数据集：



公式：

|  |  |
| --- | --- |
| **经验熵** |  |
| **条件熵** |  |
| **信息增益** | = |

1. 重要代码详解

求解信息熵：按行遍历数据，将每行中最后一个元素即标签值利用字典计数，随后遍历该字典，求解出信息熵。

def get\_empirical\_entropy(data):

    num=len(data)

    Labelcount={}

    for item in data:

        cur=item[-1]

        if cur not in Labelcount:

            Labelcount[cur]=0

        Labelcount[cur]+=1

    empirical\_entropy=0

    for item in Labelcount:

        p=float(Labelcount[item])/num

        empirical\_entropy-=p\*log(p,2)

    return empirical\_entropy

求解信息增益：首先求得经验熵，然后对所有特征进行遍历，计算出每个特征对数据的条件熵，求得信息增益，信息增益最大的为分类的最佳特征。

def choose\_best\_node(data):

    num=len(data[0])-1

    empirical\_entropy=get\_empirical\_entropy(data)

    IG=0

    bestIG=0

    bestfeature=-1

    for i in range(num):

        feaList=[item[i] for item in data]

        Feach\_value=set(feaList)

        conditional\_entropy=0

        for item in Feach\_value:

            subdata=split\_data(data,i,item)

            p=len(subdata)/float(len(data))

            conditional\_entropy+=p\*get\_empirical\_entropy(subdata)

        IG=empirical\_entropy-conditional\_entropy

        if IG>bestIG:

            bestIG=IG

            bestfeature=i

    return bestfeature

求出只具有该特征的数据：服务于信息增益，分离出该特征的所有数据。

def split\_data(data,Findex,value):

    ret=[]

    for line in list(data):

        if line[Findex]==value:

            line=list(line)

            temp=line[:Findex]

            temp.extend(line[Findex+1:])

            ret.append(temp)

    return ret

求出现次数最多的类：首先统计每个类出现的次数，然后根据字典的值降序排列，最后返回字典第一个键值，即出现次数最多的类。

def majorityCnt(classList):

    classCount={}

    for vote in classList:

        classCount[vote] = classCount.get(vote, 0) + 1

    sortedClassCount=sorted(classCount.items(),key=operator.itemgetter(1),reverse=True)

    return sortedClassCount[0][0]

建立决策树：递归解决建立决策树，递归寻找最优特征，直至其类标签全为一类或者其类标签中只有一个元素。

def createTree(data,label,fealabel):

    classList=[item[-1] for item in data]

    if classList.count(classList[0])==len(classList):

        return classList[0]

    #遍历完所有特征时返回出现次数最多的类标签

    if len(data[0])==1:

        return majorityCnt(classList)

    bestFeat=choose\_best\_node(data)

    bestFeatLabel=label[bestFeat]

    fealabel.append(bestFeatLabel)

    myTree={bestFeatLabel:{}}

    del(label[bestFeat])

    #得到训练集中所有最优特征的属性值

    featValues=[example[bestFeat] for example in data]

    #去掉重复的属性值

    uniqueVls=set(featValues)

    #递归建立决策树

    for value in uniqueVls:

        myTree[bestFeatLabel][value]=createTree(split\_data(data,bestFeat,value),label,fealabel)

    return myTree

1. 结果分析

最终建立的决策树为{'有自己的房子': {'是': '是', '否': {'有工作': {'是': '是', '否': '否'}}}}，为字典形式，因为没有使用建树工具所以使用了字典。

根据数据来看的话，该决策树比较合理。

四.源码

import numpy as np

import pandas as pd

from math import log

import operator

path = '决策树\data.txt'

data = np.array(pd.read\_csv(path,names=['年龄', '有工作', '有自己的房子', '信贷情况', '是否给贷款']))

label=['年龄', '有工作', '有自己的房子', '信贷情况', '是否给贷款']

#信息熵求解

def get\_empirical\_entropy(data):

    num=len(data)

    Labelcount={}

    for item in data:

        cur=item[-1]

        if cur not in Labelcount:

            Labelcount[cur]=0

        Labelcount[cur]+=1

    empirical\_entropy=0

    for item in Labelcount:

        p=float(Labelcount[item])/num

        empirical\_entropy-=p\*log(p,2)

    return empirical\_entropy

#求解信息增益，并选取最佳特征

def choose\_best\_node(data):

    num=len(data[0])-1

    empirical\_entropy=get\_empirical\_entropy(data)

    IG=0

    bestIG=0

    bestfeature=-1

    for i in range(num):

        feaList=[item[i] for item in data]

        Feach\_value=set(feaList)

        conditional\_entropy=0

        for item in Feach\_value:

            subdata=split\_data(data,i,item)

            p=len(subdata)/float(len(data))

            conditional\_entropy+=p\*get\_empirical\_entropy(subdata)

        IG=empirical\_entropy-conditional\_entropy

        if IG>bestIG:

            bestIG=IG

            bestfeature=i

    return bestfeature

def split\_data(data,Findex,value):

    ret=[]

    for line in list(data):

        if line[Findex]==value:

            line=list(line)

            temp=line[:Findex]

            temp.extend(line[Findex+1:])

            ret.append(temp)

    return ret

def majorityCnt(classList):

    classCount={}

    #统计classList中每个元素出现的次数

    for vote in classList:

        classCount[vote] = classCount.get(vote, 0) + 1

    #根据字典的值降序排列

    sortedClassCount=sorted(classCount.items(),key=operator.itemgetter(1),reverse=True)

    return sortedClassCount[0][0]

def createTree(data,label,fealabel):

    classList=[item[-1] for item in data]

    if classList.count(classList[0])==len(classList):

        return classList[0]

    #遍历完所有特征时返回出现次数最多的类标签

    if len(data[0])==1:

        return majorityCnt(classList)

    bestFeat=choose\_best\_node(data)

    bestFeatLabel=label[bestFeat]

    fealabel.append(bestFeatLabel)

    myTree={bestFeatLabel:{}}

    del(label[bestFeat])

    #得到训练集中所有最优特征的属性值

    featValues=[example[bestFeat] for example in data]

    #去掉重复的属性值

    uniqueVls=set(featValues)

    #递归建立决策树

    for value in uniqueVls:

        myTree[bestFeatLabel][value]=createTree(split\_data(data,bestFeat,value),label,fealabel)

    return myTree

fealabel=[]

mytree=createTree(data,label,fealabel)

print(mytree)