决策树

from math import log  
import pandas as pd  
import numpy as np  
def createdata():  
 data = pd.DataFrame(  
 {'water': [1, 1, 1, 0, 0], 'feet': [1, 1, 0, 1, 1], 'survive': ['yes', 'yes', 'no', 'no', 'no']})  
 return data  
def calculateshang(data):  
 names = data[data.columns[-1]]   
 n = len(names)  
 labels = {}  
 for i, j in names.value\_counts().items():  
 labels[i] = j  
 shang = 0  
 for i in labels:   
 pi = labels[i] / n  
 shang -= pi \* log(pi, 2)  
 return shang  
def splitdataSet(data, feature, feature\_value):  
 recvdata = []  
 n = len(data)  
 for i in range(n):   
 if (data.iloc[[i], :][feature].values[0] == feature\_value):  
 temp = data.iloc[[i], :]   
 k = temp.index.values[0]  
 temp\_t = temp.ix[k]  
 tem = temp\_t.drop(feature)  
 recvdata.append(tem)  
 recvDF = pd.DataFrame(recvdata)  
 return recvDF  
def choosebestfeaturetosplit(data):  
 nameFeatures = data.columns  
 baseEntropy = calculateshang(data)   
 bestinfoGain = 0.0   
 bestFeature = -1   
 for Feature in nameFeatures[:-1]:   
 uniquevalue = set(data[Feature])   
 newEntropy = 0.0   
 for value in uniquevalue:  
 subdata = splitdataSet(data, Feature, value)  
 pi = len(subdata) / len(data)  
 newEntropy += pi \* calculateshang(subdata)  
 infoGain = baseEntropy - newEntropy  
 if (infoGain > bestinfoGain):  
 bestinfoGain = infoGain  
 bestFeature = Feature   
 return bestFeature  
def major\_k(classlist):  
 classcount = classlist.value\_counts()  
 result = classcount.sort\_values(ascending=False).index[0]  
 return result  
def createtree(data):  
 labels = data.columns  
 classlist = data[labels[-1]]  
 if (len(classlist.values) == classlist.value\_counts()[0]):   
 return classlist.values[0]  
 if (len(labels) == 1):   
 return major\_k(classlist)  
 bestFeature = choosebestfeaturetosplit(data)  
 myTree = {bestFeature: {}}   
 unique = set(data[bestFeature])  
 for value in unique:  
 myTree[bestFeature][value] = createtree(splitdataSet(data, bestFeature, value)) # 递归创建树  
 return myTree  
def classfiy(myTree, labels, test):  
 firstStr = list(myTree.keys())[0]   
 secondDict = myTree[firstStr]  
 featIndex = labels.index(firstStr)   
 for key in secondDict.keys():  
 if (test[featIndex] == key):  
 if (type(secondDict[key]).\_\_name\_\_ == 'dict'):  
 classlabel = classfiy(secondDict[key], labels, test)  
 else:  
 classlabel = secondDict[key]  
 return classlabel  
def showtree\_pdf(data):  
 from sklearn import tree   
 import pydotplus  
  
 a = data.iloc[:, :-1]   
 b = data.iloc[:, -1]   
 clf = tree.DecisionTreeClassifier()   
 clf.fit(a, b)  
 dot\_data = tree.export\_graphviz(clf, out\_file=None)   
 graph = pydotplus.graph\_from\_dot\_data(dot\_data)  
 graph.write\_pdf("iris1.pdf")   
if \_\_name\_\_ == "\_\_main\_\_":  
 data = createdata()   
 myTree = createtree(data)  
 print(myTree)  
 result = classfiy(myTree, list(data.columns), [1, 0])  
 print(result)  
 showtree\_pdf(data)