Decision Tree

Experiment Description

基本要求

1. 基于 Watermelon-train1数据集（只有离散属性），构造ID3决策树；
2. 基于构造的 ID3 决策树，对数据集 Watermelon-test1进行预测，输出分类精度；

中级要求

1. 对数据集Watermelon-train2，构造C4.5或者CART决策树，要求可以处理连续型属性；
2. 对测试集Watermelon-test2进行预测，输出分类精度；

Data Import

```
import numpy as np
import pandas as pd
import math
import copy
def readfile(name):
    df = pd.read_csv(name, error_bad_lines = False, encoding = 'gbk')
    temp = np.array(df).tolist()
    for i in temp:
        i.pop(0)
    return temp

train1 = readfile("Watermelon-train1.csv")
train2 = readfile("Watermelon-train2.csv")
test1 = readfile("Watermelon-test1.csv")
test2 = readfile("Watermelon-test2.csv")
```

Calculate Information Gain

- Count the frequency of every single samples, and store the data in a dictionary.
- Calculate information entropy
- Return information entropy and classification information

```
def information(data):
    dic = {}
    for i in data:
        current = i[-1] #取出最后的结果
        if current not in dic.keys():
            dic[current] = 1 #创建一个新的类别
        else:
            dic[current] += 1 #原有类别+1
    result = 0.0
    for key in dic:
        prob = float(dic[key]) / len(data)
        result -= prob * math.log(prob,2)
    return result, dic
```

- Classify samples according to an feature, and delete the feature.

```
def split(data, index, kind):
    ls = []
    for temp in data:
        if temp[index] == kind:
            t = temp[0: index]
            t = t + temp[index + 1: ]
            ls.append(t)
    return ls
```

- Calculate information entropy and information gain. Return the best classification method, and conduct the process below:
    1. Extract all possible values of the feature.
    2. Split the dataset into multiple subsets.
    3. Calculate the information entropy of every subsets.
    4. Calculate the information gain.
    5. Choose the feature with the most information gain and return the index of the feature.

```
def chooseBestFeatureToSplit(data):
    base, mm= information(data)
    best = 0
    bestindex = -1
    for i in range(len(data[0]) - 1):
        ls = [index[i] for index in data]
        feture = set(ls)
```

```
        temp = 0.0
        for value in feture:
            datatemp = split(data, i, value)
            prob = len(datatemp) / float(len(data))
            t, mm = information(datatemp)
            temp += prob * t
        infoGain = base - temp
        if infoGain > best:
            best = infoGain
            bestindex = i
    return bestindex
```

Recursively Build ID3 Decision Tree

Generate Decision Tree

1. Calculate the information gain of all features from the root node, and choose the feature with most information gain to split the node. Create child node according to different values of the feature.
2. Invoke the method above recuisively on child nodes until reach the stop condition and generate a decision tree.

Stop Condition for Recursion

1. All samples of current node belongs to the same set.
2. The current node has single values in all features and has no other feature for future split.
3. Reach the maximum depth of the tree.
4. Reach the minimum sample size of leaf node.

Implementation

1. Extract the types of the dataset.
2. Calculate values and numbers of types. If the dataset only has one type, return the type.
3. Calculate the index of the feature for the best split.
4. Initialize the subtree.
5. Delete the feature has been used for classification to avoid reuse.
6. Calculate remaining features.
7. Implement recursion on every branch.

8. Return subtree.

```
def classify1(data, labels):
    typelist = [index[-1] for index in data]
    nothing, typecount = information(data)
    if len(typecount) == 1:
        return typelist[0]
    bestindex = chooseBestFeatureToSplit(data)
    bestlabel = labels[bestindex]
    Tree = {bestlabel: {}}
    temp = labels[:]
    del (temp[bestindex])
    feature = [example[bestindex] for example in data]
    unique = set(feature)
    for i in unique:
        temp = temp[:]
        Tree[bestlabel][i] = classify1(split(data, bestindex, i), temp)
    return Tree
```

Classify Test Dataset Using ID3 Decision Tree

- Extract the root node of the tree.
- Compare input data with the value of root keys.
- Iterate through the keys. If the value of the equivalent key is a value, then it is a leaf node, return the value.
- If it is not a leaf node, traversally query the subtree.

```
def run1(testdata, tree, labels):
    firstStr = list(tree.keys())[0]
    secondDict = tree[firstStr]
    featIndex = labels.index(firstStr)
    result = ''
    for key in list(secondDict.keys()):
         if testdata[featIndex] == key:
            if type(secondDict[key]).__name__ == 'dict':  # 该分支不是叶子节点,
递归
                result = run1(testdata, secondDict[key], labels)
            else:
                result = secondDict[key]
    return result
```

```
生成决策树如下: {'纹理': {'稍糊': {'色泽': {'乌黑': {'敲声': {'浊响': '是', '沉闷': '否'}}, '浅白': '否', '青绿': '否'}}, '清晰': {'根蒂': {'硬挺': '否', '蜷缩': '是', '稍蜷': '是'}}, '模糊': '否'}}
```

Calculate Accuracy

```
def simrate(data, predict):
    num = 0
    for i in range(len(data)):
        if data[i][-1] == predict[i]:
            num +=1
    return format(num / len(data), '.2%')
print("ID3分类器对于test1数据集的准确率是：", simrate(test1, result1))
```

```
ID3分类器对于test1数据集的准确率是：  70.00%
```

It can be seen that, this method has a relatively good result on small dataset.

Build Decision Tree Using $C4.5$ Algorithm

The problem of using information gain as classificaiton critiria:

Information gain tend to choose the feature with more distinct values. For example, every student has a student ID. Thus, if we categorize based on that, every student belongs to a different group, which is meaningless. However, C4.5 Algorithm use information gain ratio to choose features used for classification. This can solve the problem.

Information gain ratio = penalty parameter * information gain: $g_R(D,A) = \frac{g(D,A)}{H_A(D)}. H_A(D)$ uses current feature A as random variable in sample set D to get the empirical entropy.

The essence of information gain ratio is to mutiply information gain with a penalty parameter. When there are more feature values, the panalty parameter is smaller; when there are less feature values, the panalty parameter is bigger.

Panalty parameter: the reciprocal of the entropy of A as random variable in dataset D. That is, put samples with the same feature A values in the same group, and the panalty parameter is $\frac{1}{H_A(D)} = \frac{1}{-\sum_{i=1}^{n} \frac{|D_i|}{|D|} log_2 \frac{|D_i|}{|D|}}$

- Discretization of continuous features

  Use Binary Splitting to arrange data from smallest to largest, and choose the midpoint value for each interval. The left interval is the value smaller than the midpoint value, and the right interval is the value bigger than the midpoint value.

```
def Division(data):
    ls = data[:]
    ls.sort()
    result = []
    for i in range(len(ls) - 1):
        result.append((data[i + 1] + data[i]) / 2)
    return result
```

- Reclassify data base on the index of a feature: method = 0 put data not greater than the midpoint to the left; method = 1 put data greater than the midpoint to the right.

```
def split2(data, index, kind, method):
    ls = []
    if method == 0:
        for temp in data:
            if temp[index] <= kind:
                t = temp[0 : index]
                t = t + temp[index + 1 : ]
                ls.append(t)
    else:
        for temp in data:
            if temp[index] > kind:
                t = temp[0 : index]
                t = t + temp[index + 1 : ]
                ls.append(t)
    return ls
```

  - Calculate information entropy and informatin gain and return the best classification method. Conduct the following process on every feature of the dataset:
    1. Extract all possible values of the feature.
    2. Calculate the information entropy of every feature.
    3. Split the dataset into multiple subsets based on feature values.
    4. Calculate the information entropy of every subsets.
    5. Calculate the information gain ratio.
    6. Choose the feature with greatest information gain ratio and return the index of the feature.
    7. For continuous data, calculate the information gain ratio of every possible split point (midpoint of the intervals) and return the split point with least information gain ratio.

```
def chooseBestFeatureToSplit2(data):
    base, mm= information(data) #original infor gain
    info = []
    for j in range(len(data[0]) - 1):
        dic = {}
        for i in data:
            current = i[j] #extract result
            if current not in dic.keys():
                dic[current] = 1 #build new grp
            else:
                dic[current] += 1 #formal grp + 1
        result = 0.0
        for key in dic:
            prob = float(dic[key]) / len(data)
            result -= prob * math.log(prob,2)
        info.append(result)
    best = 0
    bestindex = -1
    bestpartvalue = None #use for discrete feature
    for i in range(len(data[0]) - 1):
        #extract all values of a feature
        ls = [index[i] for index in data]
        feture = set(ls)
        #cal infor gain
        temp = 0.0
        if type(ls[0]) == type("a"):#determine whether discrete
            for value in feture:
                datatemp = split(data, i, value)
                prob = len(datatemp) / float(len(data))
                t, mm = information(datatemp)
                temp += prob * t
        else:
            ls.sort()
            min = float("inf")
            for j in range(len(ls) - 1):
                part = (ls[j + 1] + ls[j]) / 2 #calculate split
point
                left = split2(data, i, part, 0)
                right = split2(data, i, part, 1)
                temp1, useless = information(left)
                temp2, useless = information(right)
                temp = len(left) / len(data) * temp1 + len(right) /
len(data) * temp2
                if temp < min:
                    min = temp
```

```
                    bestpartvalue = part
            temp = min
        infoGain = base - temp
        #choose by infor gain
        if info[i] != 0:
            if infoGain / info[i] >= best:
                best = infoGain / info[i]
                bestindex = i
    return bestindex, bestpartvalue
```

Build $C4.5$ Decision Tree Traversally

- Build the decision tree.
    1. Calculate the information gain of all features from the root node, and choose the feature with most information gain to split the node. Create child node according to different values of the feature.
    2. Invoke the method above recuisively on child nodes until reach the stop condition and generate a decision tree.
- Iteration stop condition.
    1. All samples of current node belongs to the same set.
    2. The current node has single values in all features and has no other feature for future split.
    3. Reach the maximum depth of the tree.
    4. Reach the minimum sample size of leaf node.
- Implementation.
    1. Extract the types of the dataset.
    2. Calculate values and numbers of types. If the dataset only has one type, return the type.
    3. Calculate the index of the feature for the best split.
    4. Initialize the subtree.
    5. Delete the feature has been used for classification to avoid reuse.
    6. Calculate remaining features.
    7. Implement recursion on every branch.
    8. Return subtree.

```
def classify2(data, labels):
    typelist = [index[-1] for index in data] #extract features
```

```
    nothing, typecount = information(data) #calculate infor gain
    if typecount == len(typelist): #if only 1 grp
        return typelist[0]
    bestindex, part = chooseBestFeatureToSplit2(data)  #the index of best
split feature
    if bestindex == -1:
        return "是"
    if type([t[bestindex] for t in data][0]) == type("a"):#if discrete
        bestlabel = labels[bestindex]
        Tree = {bestlabel: {}}
        temp = copy.copy(labels)
        feature = [example[bestindex] for example in data]
        del (temp[bestindex])  #chosen feature no longer used for
classification
        unique = set(feature)  #every possible values of the split feature
        for i in unique:
            s = temp[:]
            Tree[bestlabel][i] = classify2(split(data, bestindex, i), s)
    else: #continuous features
        bestlabel = labels[bestindex] + "<" + str(part)
        Tree = {bestlabel: {}}
        temp = labels[:]
        del(temp[bestindex])
        leftdata = split2(data, bestindex, part, 0)
        Tree[bestlabel]["是"] = classify2(leftdata, temp)
        rightdata = split2(data, bestindex, part, 1)
        Tree[bestlabel]["否"] = classify2(rightdata, temp)
    return Tree
```

Use $C4.5$ Decision Tree for Classification

- Extract the root node of the current tree.
- Use the root node to check the type of current data
- If it is discrete value:
    1. If the branch queried is a leaf node, return the value.
    2. If the branch queried is not a leaf node, traversally query the subtree.
- If it is non-discrete value:
    1. Extract the value to compare with current value.
    2. If the value is bigger then use "否" as label, else use"是" as label
    3. If the queried branch is a leaf node, return the value.
    4. If it is not leaf node, traversally query the subtree.

```
def run2(data, tree, labels):
    firstStr = list(tree.keys())[0]  # root node
    firstLabel = firstStr
    t = str(firstStr).find('<') # check whether feature continuous
    if t > -1:  # if continuous
        firstLabel = str(firstStr)[ : t]
    secondDict = tree[firstStr]
    featIndex = labels.index(firstLabel)  # node's feature
    result = ''
    for key in list(secondDict.keys()):  # iterate through branches
        if type(data[featIndex]) == type("a"):
            if data[featIndex] == key:
                if type(secondDict[key]).__name__ == 'dict':  # not leaf
node, traversal
                    result = run2(data, secondDict[key], labels)
                else:  # is leaf node, return
                    result = secondDict[key]
        else:
            value = float(str(firstStr)[t + 1 : ])
            if data[featIndex] <= value:
                if type(secondDict['是']).__name__ == 'dict':  # not leaf
node, traversal
                    result = run2(data, secondDict['是'], labels)
                else:  # is leaf node, return
                    result = secondDict['是']
            else:
                if type(secondDict['否']).__name__ == 'dict':  # not leaf
node, traversal
                    result = run2(data, secondDict['否'], labels)
                else:  # is leaf node, return
                    result = secondDict['否']
    return result

def getresult2(train, test, labels):
    ls = []
    tree = classify2(train, labels)
    print("生成决策树如下:", tree)
    for index in test:
        ls.append(run2(index, tree, labels))
    return ls

labels2 = ["色泽", "根蒂", "敲声", "纹理", "密度", "好瓜"]
result2 = getresult2(train2, test2, labels2)
```

```
生成决策树如下: {'纹理': {'稍糊': {'敲声': {'浊响': {'密度<0.56': {'是': '是', '否': '是'}}, '沉闷': {'密度<0.6615': {'是': '是', '否': {'根蒂': {'蜷缩': '是', '稍蜷': '是'}}}}}}, '清晰': {'根蒂': {'硬挺': '是', '蜷缩': {'密度<0.5820000000000001': {'是': '是', '否': {'敲声': {'浊响': {'色泽': {'乌黑': '是', '青绿': '是'}}, '沉闷': {'色泽': {'乌黑': '是', '青绿': '是'}}}}}}, '稍蜷': {'密度<0.3815': {'是': '是', '否': {'色泽': {'乌黑': '是', '青绿': '是'}}}}}}, '模糊': {'密度<0.29400000000000004': {'是': '是', '否': '是'}}}}
```

Calculate accuracy

```
print("C4.5 accuracy: ", simrate(test2, result2))
```

```
C4.5 accuracy:  60.00%
```

It can be seen that the accuracy of C4.5 decision tree is mediocre.