ID3(Play Tennis)

November 15, 2023

Module “Apprentissage automatique” MST IASD/S1 2023-2024 (M. AIT KBIR)

ID3 appliqué aux données avec attributs Discrets (voir le livre “Machine learning in action” page 39)

Désactiver les commentaires pour voir les résultats intermédaires

[1]:
```python
from math import log
import treePlotter as tpl
import operator
import csv

def createDataSet():
    with open('PlayTennisV1.csv', 'r') as f:
        reader = csv.reader(f,delimiter=';')
        dataSet = list(reader)
    return dataSet[1:], dataSet[0]
```

[2]:
```python
a,b=createDataSet()
print(a)
```

```
[['sunny', 'hot', 'high', 'weak', 'no'], ['sunny', 'hot', 'high', 'strong',
'no'], ['overcast', 'hot', 'high', 'weak', 'yes'], ['rain', 'mild', 'high',
'weak', 'yes'], ['rain', 'cool', 'normal', 'weak', 'yes'], ['rain', 'cool',
'normal', 'strong', 'no'], ['overcast', 'cool', 'normal', 'strong', 'yes'],
['sunny', 'mild', 'high', 'weak', 'no'], ['sunny', 'cool', 'normal', 'weak',
'yes'], ['rain', 'mild', 'normal', 'weak', 'yes'], ['sunny', 'mild', 'normal',
'strong', 'yes'], ['overcast', 'mild', 'high', 'strong', 'yes'], ['overcast',
'hot', 'normal', 'weak', 'yes'], ['rain', 'mild', 'high', 'strong', 'no']]
```

[3]:
```python
def calcShannonEnt(dataSet):
    numEntries = len(dataSet)
    labelCounts = {}
    for featVec in dataSet:               # Le nombre d'occurrences pour chaque␣
↪attribut
        currentLabel = featVec[-1]
        if currentLabel not in labelCounts.keys():
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1
```

```
    shannonEnt = 0.0
    for key in labelCounts:
        prob = float(labelCounts[key])/numEntries
        shannonEnt -= prob * log(prob,2)  # log base 2
    return shannonEnt
```

[4]:
```
calcShannonEnt(a)
```

[4]: 0.9402859586706309

[5]:
```
def splitDataSet(dataSet, axis, value):
    retDataSet = []
    for featVec in dataSet:
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]     # Mettre dehors l'axe utilisé␣
 ↪pour la subdivision
            reducedFeatVec.extend(featVec[axis+1:])
            retDataSet.append(reducedFeatVec)
    return retDataSet
```

[6]:
```
c = splitDataSet(a,0,'overcast')
print(len(c),c)
```

```
4 [['hot', 'high', 'weak', 'yes'], ['cool', 'normal', 'strong', 'yes'], ['mild',
'high', 'strong', 'yes'], ['hot', 'normal', 'weak', 'yes']]
```

[7]:
```
splitDataSet(c,2,'weak')
```

[7]: [['hot', 'high', 'yes'], ['hot', 'normal', 'yes']]

[8]:
```
def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1       # Dernière colonne contient la␣
 ↪classe d'appartenance
    baseEntropy = calcShannonEnt(dataSet)
    bestInfoGain = 0.0; bestFeature = -1
    for i in range(numFeatures):            # Pour chaque attribut
        featList = [example[i] for example in dataSet] # Liste des valeurs
        uniqueVals = set(featList)          # Liste des valeurs sans doublons
        newEntropy = 0.0
        for value in uniqueVals:
            subDataSet = splitDataSet(dataSet, i, value)
            prob = len(subDataSet)/float(len(dataSet))
            newEntropy += prob * calcShannonEnt(subDataSet)
        infoGain = baseEntropy - newEntropy  # Calculer le gain
        if (infoGain > bestInfoGain):        # Garder le meilleur gain
            bestInfoGain = infoGain
            bestFeature = i
    return bestFeature                       # rang de l'attribut: entier
```

[9]:
```python
ra = chooseBestFeatureToSplit(a)
print(ra)
```

```
0
```

[10]:
```python
c
```

[10]:
```
[['hot', 'high', 'weak', 'yes'],
 ['cool', 'normal', 'strong', 'yes'],
 ['mild', 'high', 'strong', 'yes'],
 ['hot', 'normal', 'weak', 'yes']]
```

[11]:
```python
rc=chooseBestFeatureToSplit(c)
print(rc)
```

```
-1
```

[12]:
```python
def majorityCnt(classList):     # Retourner la classe avec la majorité de vote
    classCount={}
    for vote in classList:
        if vote not in classCount.keys(): classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.items(), key=lambda item: item[1],␣
 ↪reverse=True)
    #print(sortedClassCount)
    return sortedClassCount[0][0]
majorityCnt(['no','no','no','yes'])
```

[12]:
```
'no'
```

Apprentissage

[13]:
```python
def createTree(dataSet,labels):
    classList = [example[-1] for example in dataSet]
    if classList.count(classList[0]) == len(classList):
        return classList[0]            # Arrêter la decomposition (Exemples de␣
 ↪la même classe)
    if len(dataSet[0]) == 1:
        return majorityCnt(classList)   # Arrêter s'il ne reste qu'un seul␣
 ↪attribut

    bestFeat = chooseBestFeatureToSplit(dataSet)
    bestFeatLabel = labels[bestFeat]
    print(bestFeatLabel)

    myTree = {bestFeatLabel:{}}         # Initialiser le dictionnaire

    del(labels[bestFeat])
```

```
    featValues = [example[bestFeat] for example in dataSet]
    uniqueVals = set(featValues)
    for value in uniqueVals:
        subLabels = labels.copy()           # Copier chaque fois les noms des␣
↪attributs
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet,␣
↪bestFeat, value),subLabels)
    return myTree
```

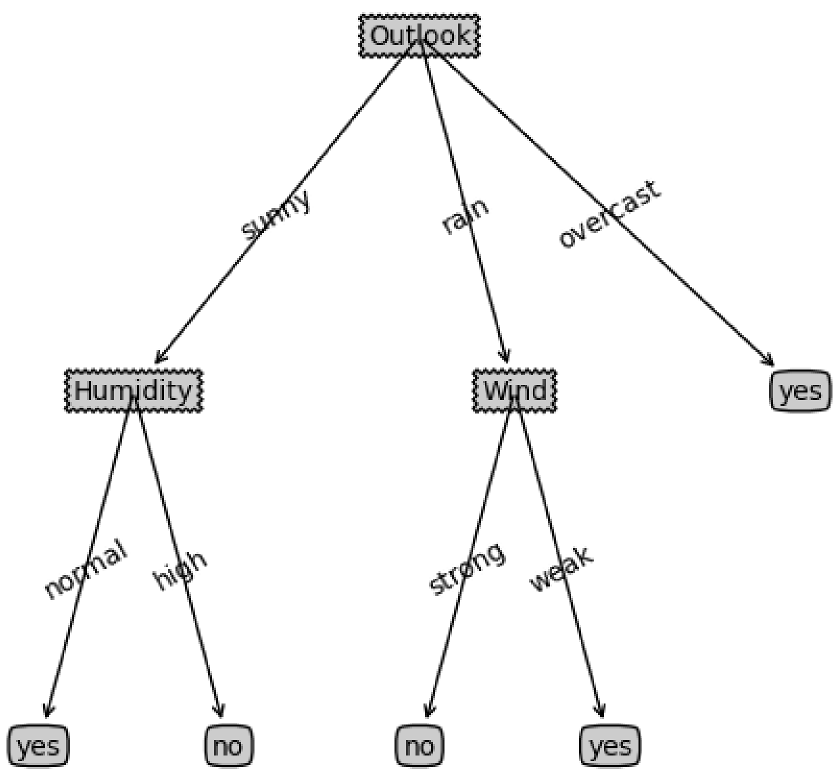
Test

[14]:
```
dataSet, labels=createDataSet()
tr=createTree(dataSet, labels)

print(tr)
```

```
Outlook
Humidity
Wind
{'Outlook': {'sunny': {'Humidity': {'normal': 'yes', 'high': 'no'}}, 'rain':
{'Wind': {'strong': 'no', 'weak': 'yes'}}, 'overcast': 'yes'}}
```

[15]:
```
tpl.createPlot(tr)
```



Généralisation

[16]:
```
def classify(inputTree,featLabels,testVec):
    firstStr = list(inputTree.keys())[0]
    secondDict = inputTree[firstStr]
    featIndex = featLabels.index(firstStr)
    key = testVec[featIndex]
    valueOfFeat = secondDict[key]
    #print(key,valueOfFeat)
    if isinstance(valueOfFeat, dict):
        classLabel = classify(valueOfFeat, featLabels, testVec)
    else: classLabel = valueOfFeat
    return classLabel

classify(tr,['Outlook','Temperature','Humidity','Wind'],['sunny', 'hot',␣
↪'high', 'weak'])
```

[16]: 'no'