Homework 3

Yihui He

yihuihe@fomail.com

May 9, 2016

Contents

1 problem 1

(a) $(3+1) \times (3+1) \times (2+1) \times (2+1) = 144$

(b) $Talent = Don't\ care \wedge AveHeight = NBA\_tall \wedge GreatCoach = Don't\ care \wedge TeamChemistry = Great$

(c) $True$, Because there's no negative example.

(d) $True$, All features have different values.

2 problem 2

2.1 decision tree

Decision Tree is shown in Fig 1

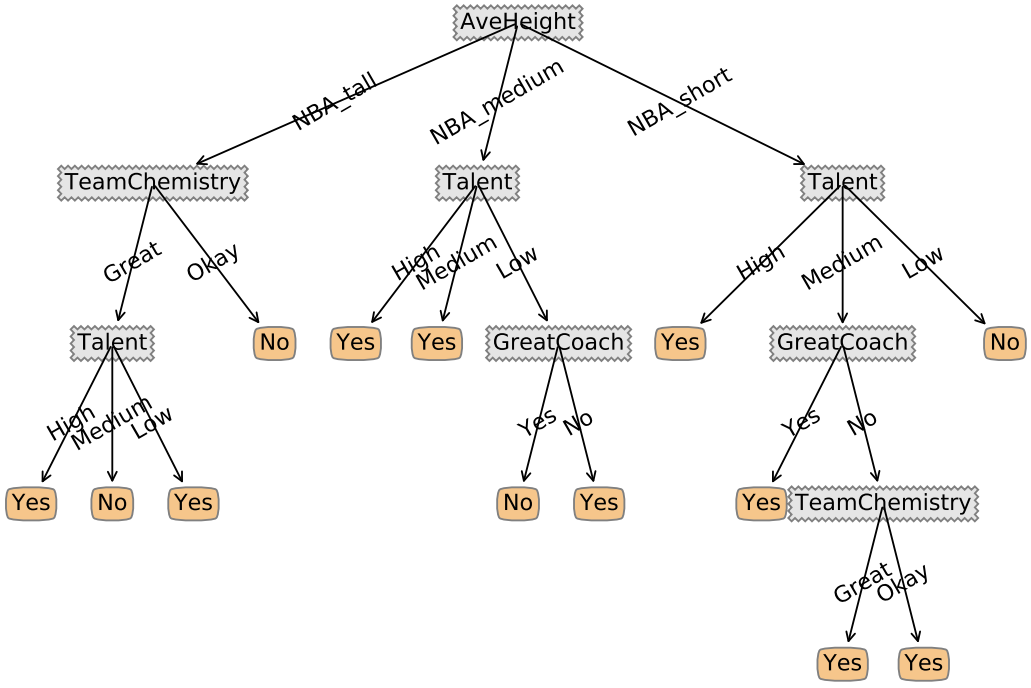

Figure 1: Decision Tree

2.2 Decision making

Below I show how decision is made (Depth first traverse)

```
==========================================================================
root
--------------------------------------------------------------------------
Talent impurity
High 0.721928094887
Medium 0.970950594455
Low 0.918295834054
```

```
information Gain 0.115313880099
Total impurity 0.87338552819

AveHeight impurity
NBA_tall 0.970950594455
NBA_medium 0.650022421648
NBA_short 0.970950594455
information Gain 0.138096878636
Total impurity 0.850602529652

GreatCoach impurity
Yes 0.954434002925
No 1.0
information Gain 0.011482406826
Total impurity 0.977217001462

TeamChemistry impurity
Great 0.863120568567
Okay 0.991076059838
information Gain 0.0536038758816
Total impurity 0.935095532407

I choose AveHeight
================================================================================
root->AveHeight=NBA_tall
--------------------------------------------------------------------------------
Talent impurity
High 1.0
Medium 0.0
Low 1.0
information Gain 0.170950594455
Total impurity 0.8

GreatCoach impurity
Yes 0.0
No 0.811278124459
information Gain 0.321928094887
Total impurity 0.649022499567

TeamChemistry impurity
Great 0.918295834054
Okay 0.0
information Gain 0.419973094022
Total impurity 0.550977500433

I choose TeamChemistry
================================================================================
root->AveHeight=NBA_tall->TeamChemistry=Great
--------------------------------------------------------------------------------
```

```
Talent impurity
High 0.0
Medium 0.0
Low 0.0
information Gain 0.918295834054
Total impurity 0.0

GreatCoach impurity
Yes 0.0
No 1.0
information Gain 0.251629167388
Total impurity 0.666666666667

I choose Talent
================================================================================
root->AveHeight=NBA_medium
--------------------------------------------------------------------------------
Talent impurity
High 0.0
Medium 0.0
Low 1.0
information Gain 0.316689088315
Total impurity 0.333333333333

GreatCoach impurity
Yes 0.811278124459
No 0.0
information Gain 0.109170338676
Total impurity 0.540852082973

TeamChemistry impurity
Great 0.0
Okay 0.918295834054
information Gain 0.190874504621
Total impurity 0.459147917027

I choose Talent
================================================================================
root->AveHeight=NBA_medium->Talent=Low
--------------------------------------------------------------------------------
GreatCoach impurity
Yes 0.0
No 0.0
information Gain 1.0
Total impurity 0.0

TeamChemistry impurity
Great 0.0
Okay 0.0
```

```
information Gain 1.0
Total impurity 0.0

I choose GreatCoach
================================================================================
root->AveHeight=NBA_short
--------------------------------------------------------------------------------
Talent impurity
Medium 0.918295834054
Low 0.0
information Gain 0.419973094022
Total impurity 0.550977500433

GreatCoach impurity
Yes 0.918295834054
No 1.0
information Gain 0.019973094022
Total impurity 0.950977500433

TeamChemistry impurity
Great 0.0
Okay 1.0
information Gain 0.170950594455
Total impurity 0.8

I choose Talent
================================================================================
root->AveHeight=NBA_short->Talent=Medium
--------------------------------------------------------------------------------
GreatCoach impurity
Yes 0.0
No 1.0
information Gain 0.251629167388
Total impurity 0.666666666667

TeamChemistry impurity
Okay 0.918295834054
information Gain 0.0
Total impurity 0.918295834054

I choose GreatCoach
================================================================================
root->AveHeight=NBA_short->Talent=Medium->GreatCoach=No
--------------------------------------------------------------------------------
TeamChemistry impurity
Okay 1.0
information Gain 0.0
Total impurity 1.0
```

```
I choose TeamChemistry
```

2.3 Evaluation

These are incorrectly classified exmaples. Also error rate.

```
train.xlsx
    Talent  AveHeight GreatCoach TeamChemistry WinTitle?
15  Medium  NBA_short         No          Okay        No
error rate 0.0625
HW3 - Test data set 1.xlsx
    Talent   AveHeight GreatCoach TeamChemistry WinTitle?
1      Low  NBA_medium         Yes          Great        Yes
2      Low   NBA_short         Yes          Great        Yes
3   Medium   NBA_short         Yes           Okay         No
4   Medium    NBA_tall         Yes          Great        Yes
7      Low    NBA_tall         Yes          Great         No
8      Low  NBA_medium         Yes           Okay        Yes
10  Medium    NBA_tall         Yes           Okay        Yes
16  Medium   NBA_short          No           Okay         No
20     Low    NBA_tall          No          Great         No
error rate 0.45
HW3 - Test data set 2.xlsx
    Talent   AveHeight GreatCoach TeamChemistry WinTitle?
2      Low   NBA_short         Yes          Great        Yes
7     High   NBA_short         Yes           Okay         No
10  Medium  NBA_medium          No           Okay         No
13  Medium    NBA_tall          No          Great        Yes
15     Low  NBA_medium         Yes          Great        Yes
18     Low    NBA_tall          No          Great         No
error rate 0.3
```

3 problem 3

3.1 ranking tree

Ranking tree is shown in Fig 2

3.2 ranks

Below I show numbers of examples in each rank, errors and error rate.

```
sorted probabilities
[ 0.8         0.66666667  0.66666667  0.66666667  0.66666667  0.66666667
```

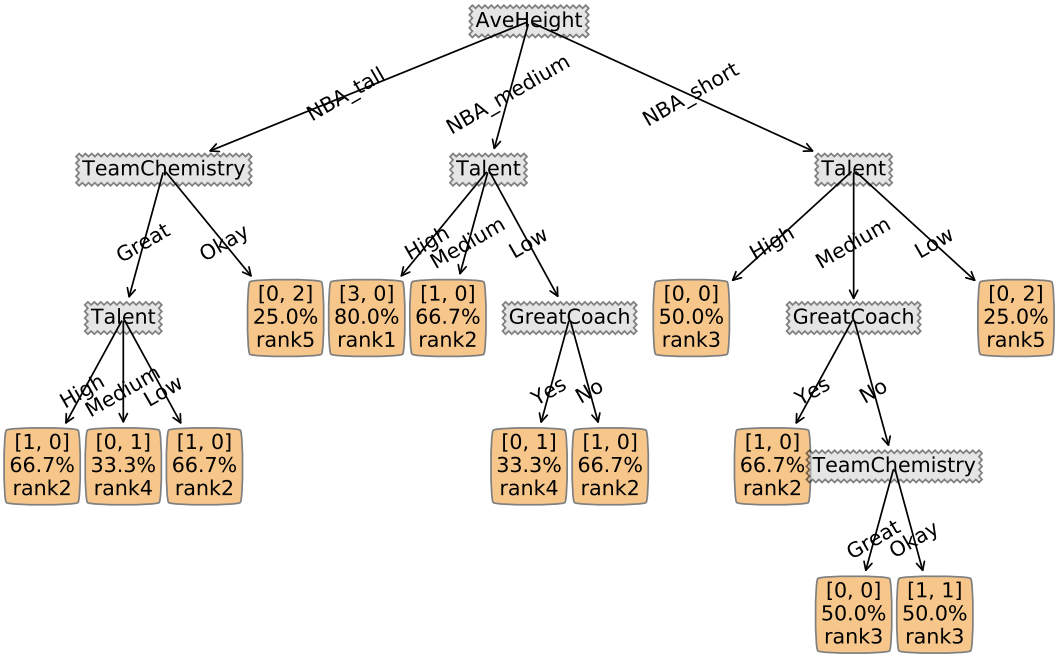

Figure 2: Ranking Tree

```
  0.5          0.5          0.5          0.33333333   0.33333333   0.25         0.25        ]
train.xlsx
error each level [0.0, 0.0, 0.5, 0.0, 0.0] total 0.5
        Yes  No
1.0       3   0
2.0       5   0
3.0       1   1
4.0       0   2
5.0       0   4
Total     9   7
error rate 0.00793650793651
accuracy 0.992063492063

HW3 - Test data set 1.xlsx
error each level [0.0, 24.0, 6.0, 3.5, 4.0] total 37.5
        Yes  No
1.0       2   0
2.0       2   3
3.0       2   1
4.0       3   1
5.0       2   4
Total    11   9
```

```
error rate 0.378787878788
accuracy 0.621212121212

HW3 - Test data set 2.xlsx
error each level [0.0, 16.0, 4.0, 2.0, 1.0] total 23.0
       Yes  No
1.0      3   0
2.0      6   2
3.0      2   1
4.0      2   1
5.0      1   2
Total   14   6
error rate 0.27380952381
accuracy 0.72619047619
```

4 supplimentary

4.1 Tree functions

```
from math import log
import operator
import numpy as np

leavesCnt=0
def entropy(dataSet):
    numEntries = len(dataSet)
    labelCounts = {}
    for featVec in dataSet:
        currentLabel = featVec[-1]
        if currentLabel not in labelCounts.keys():
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1
    ent = 0.0
    for key in labelCounts:
        prob = float(labelCounts[key])/numEntries
        imp=-prob*log(prob,2)
        ent += imp #log base 2
    return ent

def splitDataSet(dataSet, axis, value):
    retDataSet = []
    for featVec in dataSet:
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]
            reducedFeatVec.extend(featVec[axis+1:])
            retDataSet.append(reducedFeatVec)
```

```
    return retDataSet

def chooseBestFeatureToSplit(dataSet,labels):
    numFeatures = len(dataSet[0]) - 1
    #the last column is used for the labels
    baseEntropy = entropy(dataSet)
    bestInfoGain = 0.0; bestFeature = 0
    for i in range(numFeatures):
        #iterate over all the features
        featList = [example[i] for example in dataSet]
        #create a list of all the examples of this feature
        uniqueVals = set(featList)
        #get a set of unique values
        newEntropy = 0.0
        print labels[i],'impurity'
        for value in uniqueVals:
            subDataSet = splitDataSet(dataSet, i, value)
            prob = len(subDataSet)/float(len(dataSet))

            sub_ent=entropy(subDataSet)
            print value,sub_ent
            newEntropy +=prob*sub_ent
        infoGain = baseEntropy - newEntropy
        print 'information Gain',infoGain
        print 'Total impurity', newEntropy,'\n'
        if (infoGain > bestInfoGain):
            #compare this to the best gain so far
            bestInfoGain = infoGain
            #if better than current best, set to best
            bestFeature = i
    return bestFeature,bestInfoGain
    #returns an integer

def findClass(classList,rank):
    global leavesCnt
    classCount={}
    for i in ['Yes','No']:
        classCount[i]=0
    for vote in classList:
        if vote not in classCount.keys():
            classCount[vote] = 0
        classCount[vote] += 1
    clas='Yes'
    for vote in classList:
        if classCount[vote]>classCount[clas]:
            clas=vote
    infos=[classCount['Yes'],classCount['No'],\
            (classCount['Yes']+1.0)/(classCount['Yes']+classCount['No']+2),\
            leavesCnt]
```

```
    leavesCnt+=1
    rank.append(infos)
    return clas,infos

def sortRankingTree(rankingTree,rank):
    sorted_rank=np.array(rank).T[-2:]
    idx=(-sorted_rank[0]).argsort()
    sorted_idx=sorted_rank[1][idx]
    #print sorted_idx
    sorted_prob=sorted_rank[0][idx]
    print "sorted probabilities"
    print sorted_prob
    new_ranks=[1]
    for i in range(1,len(sorted_prob)):
        if sorted_prob[i]==sorted_prob[i-1]:
            new_ranks.append(new_ranks[-1])
        else:
            new_ranks.append(new_ranks[-1]+1)
    #print new_ranks
    ret_idx=sorted_idx.argsort().astype(int)
    ret_ranks=np.array(new_ranks)[ret_idx]
    return ret_ranks

def rankingTree2string(rankTree):
    firstStr = rankTree.keys()[0]
    secondDict = rankTree[firstStr]
    for valueOfFeat in secondDict:
        item=secondDict[valueOfFeat]
        if isinstance(item, dict):
            rankingTree2string(item)
        else:
            string=str(item[:2])+'\n'
            string+="{:2.1f}".format(item[-2]*100.0)+'%\n'
            string+='rank'+str(item[-1])
            secondDict[valueOfFeat]=string


def Rank(rankTree,rank):
    firstStr = rankTree.keys()[0]
    secondDict = rankTree[firstStr]
    for valueOfFeat in secondDict:
        item=secondDict[valueOfFeat]
        if isinstance(item, dict):
            Rank(item,rank)
        else:
            secondDict[valueOfFeat][-1]=rank[item[-1]]

def getUniqueVals(dataSet,labels):
    uniqueVals=dict()
```

```
    i=0
    for feature in dataSet.T[:-1]:
        uniqueVals[labels[i]]=set(feature)
        i+=1
    return uniqueVals

def createTree(dataSet,labels,ValsSet,node='root',rank=None):
    #check impurity and if we've run out of features
    classList = [example[-1] for example in dataSet]
    if classList.count(classList[0]) == len(classList) or len(dataSet[0])==1:
        return findClass(classList,rank)
    print '=============================================================================='
    print node
    print '------------------------------------------------------------------------------'
    bestFeat,bestInfoGain = chooseBestFeatureToSplit(dataSet,labels)
    #if bestInfoGain==0:
    #    return findClass(classList,rank)
    bestFeatLabel = labels[bestFeat]
    print 'I choose',bestFeatLabel
    node+='->'+bestFeatLabel+'='
    myTree = {bestFeatLabel:{}}
    rankTree={bestFeatLabel:{}}
    del(labels[bestFeat])
    #print bestFeat
    featValues = [example[bestFeat] for example in dataSet]
    #print featValues
    uniqueVals = set(featValues)
    if len(uniqueVals)!=len(ValsSet[bestFeatLabel]):
       # some value have no example
       for val in ValsSet[bestFeatLabel]:
           if val not in uniqueVals:
               l,votes=findClass(classList,rank)
               myTree[bestFeatLabel][val]='Yes'
               rank[-1]=[0,0,.5,votes[-1]]
               rankTree[bestFeatLabel][val]=rank[-1]

    for value in uniqueVals:
        #print value
        subLabels = labels[:]
        myTree[bestFeatLabel][value],rankTree[bestFeatLabel][value]\
                = createTree(splitDataSet(dataSet, bestFeat, value),
                        subLabels,ValsSet,node=node+value,rank=rank)
    return myTree,rankTree #also return a ranking tree

def classify(inputTree,featLabels,testVec,ranking=False):
    firstStr = inputTree.keys()[0]
    secondDict = inputTree[firstStr]
    featIndex = featLabels.index(firstStr)
    key = testVec[featIndex]
```

```
    valueOfFeat = secondDict[key]
    if isinstance(valueOfFeat, dict):
        classLabel = classify(valueOfFeat, featLabels, testVec,ranking)
    else:
        if ranking==False:
            classLabel=valueOfFeat
        else:
            classLabel=valueOfFeat[-1]
    return classLabel


#def storeTree(inputTree,filename):
#    import pickle
#    fw = open(filename,'w')
#    pickle.dump(inputTree,fw)
#    fw.close()
#
#def grabTree(filename):
#    import pickle
#    fr = open(filename)
#    return pickle.load(fr)
```

4.2 Tree Evaluation

```
from trees import *
from treePlotter import *
import numpy as np
import pandas as pd
import sys
sys.stdout = open("decision.txt", "w")
rank=[]
decision=False
def getData(filename):
    raw_train=pd.read_excel(filename,index_col=0)

    dataset=np.array(raw_train,dtype=str).tolist()

    label=np.array(raw_train.T.index,dtype=str)[:-1].tolist()

    obj=np.array(raw_train.T.index,dtype=str)[-1]

    return dataset,label,obj
dataset,label,obj=getData('train.xlsx')
u=getUniqueVals(np.array(dataset),label)
myTree,rankTree=createTree(dataset,label,u,rank=rank)

def DTmetric(myTree=myTree):
```

```
    sys.stdout = open("classify.txt", "w")
    for i in range(3):
        name='HW3 - Test data set '+str(i)+'.xlsx'
        if i==0:
            name='train.xlsx'
        dataset,label,obj=getData(name)
        print name
        res=[]
        idx=[]
        for j,cnt in zip(dataset,range(1,len(dataset)+1)):
            clas=classify(myTree,label,j)
            #print clas
            if clas != j[-1]:
                idx.append(cnt)
                res.append(j)

        res=pd.DataFrame(res)
        res.columns=label+[obj]
        res.index=idx
        print res
        print 'error rate',float(len(idx))/len(dataset)


def RTmetric(rankTree=rankTree):
    global rank
    sys.stdout = open("ranking.txt", "w")
    rank=sortRankingTree(rankTree,rank)
    #print rank
    Rank(rankTree,rank)
    for i in range(3):
        name='HW3 - Test data set '+str(i)+'.xlsx'
        if i==0:
            name='train.xlsx'
        dataset,label,obj=getData(name)
        print name
        ranks=[[],[]]
        for j,cnt in zip(dataset,range(1,len(dataset)+1)):
            clas=classify(rankTree,label,j,True)
            if j[-1]=='Yes':
                ranks[0].append(clas)
            else:
                ranks[1].append(clas)
        ranks=pd.DataFrame(ranks).T
        ranks=ranks.apply(pd.value_counts).fillna(0).astype(int)
        ranks.columns=['Yes','No']
        err=0
        err_each_level=[]
        idx=ranks.index
        for level in range(len(ranks)):
```

```
            neg=ranks.ix[idx[level]]['No']
            err_in_this_level=0
            for sublevel in range(level,len(ranks)):
                pos=ranks.ix[idx[sublevel]]['Yes']
                if idx[sublevel]==idx[level]:
                    pos*=.5
                err_in_this_level+=neg*pos
                err+=neg*pos
            err_each_level.append(err_in_this_level)
        print 'error each level',err_each_level,'total',err
        err_rate=float(err)/ranks.sum().prod()
        print ranks.append(pd.Series(ranks.sum(),name='Total'))
        print 'error rate',err_rate
        print 'accuracy',1-err_rate,'\n'

print '\n'
if decision==False:
    RTmetric(rankTree)
    rankingTree2string(rankTree)
    createPlot(rankTree)
else:
    DTmetric(myTree)
    createPlot(myTree)
```