Decision Trees

葉建華

jhyeh@mail.au.edu.tw

http://jhyeh.csie.au.edu.tw/

Twenty Questions

(1) 你會把自己比喻成哪種花香
濃郁的花香 -- 去第2題
清淡的花香 -- 去第3 題

(2) 你會選擇哪種香味的潤唇膏？
水果味 -- 去第4題
薄荷味 -- 去第5題

(3) 你會把自己比喻為哪種花束？
紅色系的花束(如紅色/粉紅色/橙色) -- 去第2題
非紅色系的花束(如白色/藍色/紫色) -- 去第5題

(4) 你跟意中人首次約會用什麼香水？
帶有甜味的花香 -- 去第6題
清爽的水果香 -- 去第7題

(5) 你較喜歡哪種味道？
盛夏乾燥的草味 -- 去第4題
雨後濕淋淋的草味 -- 去第7題

(6) 玫瑰和百合，你較喜歡哪種香味？
玫瑰 -- 去第8題
百合 -- 去第9題

(7)你剛發現一瓶新款洗頭水，你十分喜歡它的味道，那瓶子的形狀是怎樣的？
圓形 -- 去第6題
長身形 -- 去第10題

(8)當你情緒低落時，哪種味道最能撫慰你的心靈？
花香 -- 去第11題
森林的味道 -- 去第12題

Decision Trees

- One of the most commonly used technique
- Decision blocks, terminal blocks

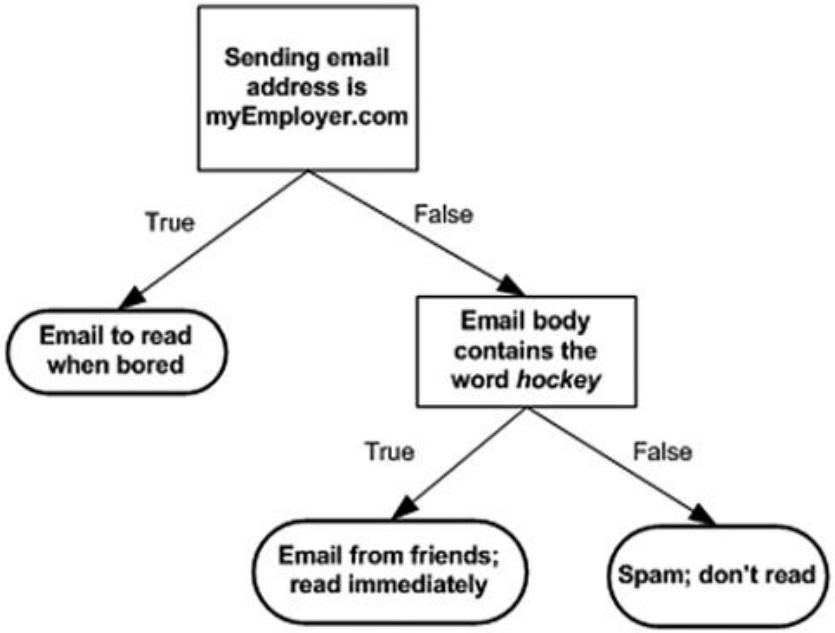

Figure 3.1 A decision tree in flowchart form

Compare with kNN

- Features of kNN
  - Good job on classification
  - Has no insights about the data
  - No model created, re-calculate all data again and again
- Decision trees
  - Pros: Computationally cheap to use, easy to understand, missing values OK, can deal with irrelevant features
  - Cons: Prone to overfitting

Tree Construction

Check if every item in the dataset is in the same class:

If so return the class label

Else

find the best feature to split the data

split the dataset

create a branch node

for each split

call createBranch and add the result to the branch node

return branch node

General Approach

General approach to decision trees

1. Collect: Any method.
2. Prepare: This tree-building algorithm works only on nominal values, so any continuous values will need to be quantized.
3. Analyze: Any method. You should visually inspect the tree after it is built.
4. Train: Construct a tree data structure.
5. Test: Calculate the error rate with the learned tree.
6. Use: This can be used in any supervised learning task. Often, trees are used to better understand the data.

Fish/Not Fish

	Can survive without coming to surface?	Has flippers?	Fish?
1	Yes	Yes	Yes
2	Yes	Yes	Yes
3	Yes	No	No
4	No	Yes	No
5	No	Yes	No

Table 3.1 Marine animal data

Information Gain: Entropy

$$I(x_i) = \log_2 p(x_i)$$

$$H = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

Shannon Entropy

Listing 3.1 Function to calculate the Shannon entropy of a dataset

```
from math import log

def calcShannonEnt(dataSet):
    numEntries = len(dataSet)
    labelCounts = {}
    for featVec in dataSet:
        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)
    return shannonEnt
```

1 Create dictionary of all possible classes

2 Logarithm base 2

Dataset Splitting

Listing 3.2 Dataset splitting on a given feature

```
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
```

1 Create separate list

2 Cut out the feature split on

Dataset Splitting by Best Feature

Listing 3.3 Choosing the best feature to split on

```
def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1
    baseEntropy = calcShannonEnt(dataSet)
    bestInfoGain = 0.0; bestFeature = -1
    for i in range(numFeatures):
        featList = [example[i] for example in dataSet]
        uniqueVals = set(featList)
        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
        if (infoGain > bestInfoGain):
            bestInfoGain = infoGain
            bestFeature = I
    return bestFeature
```

1. Create unique list of class labels
2. Calculate entropy for each split
3. Find the best information gain

Recursively Building

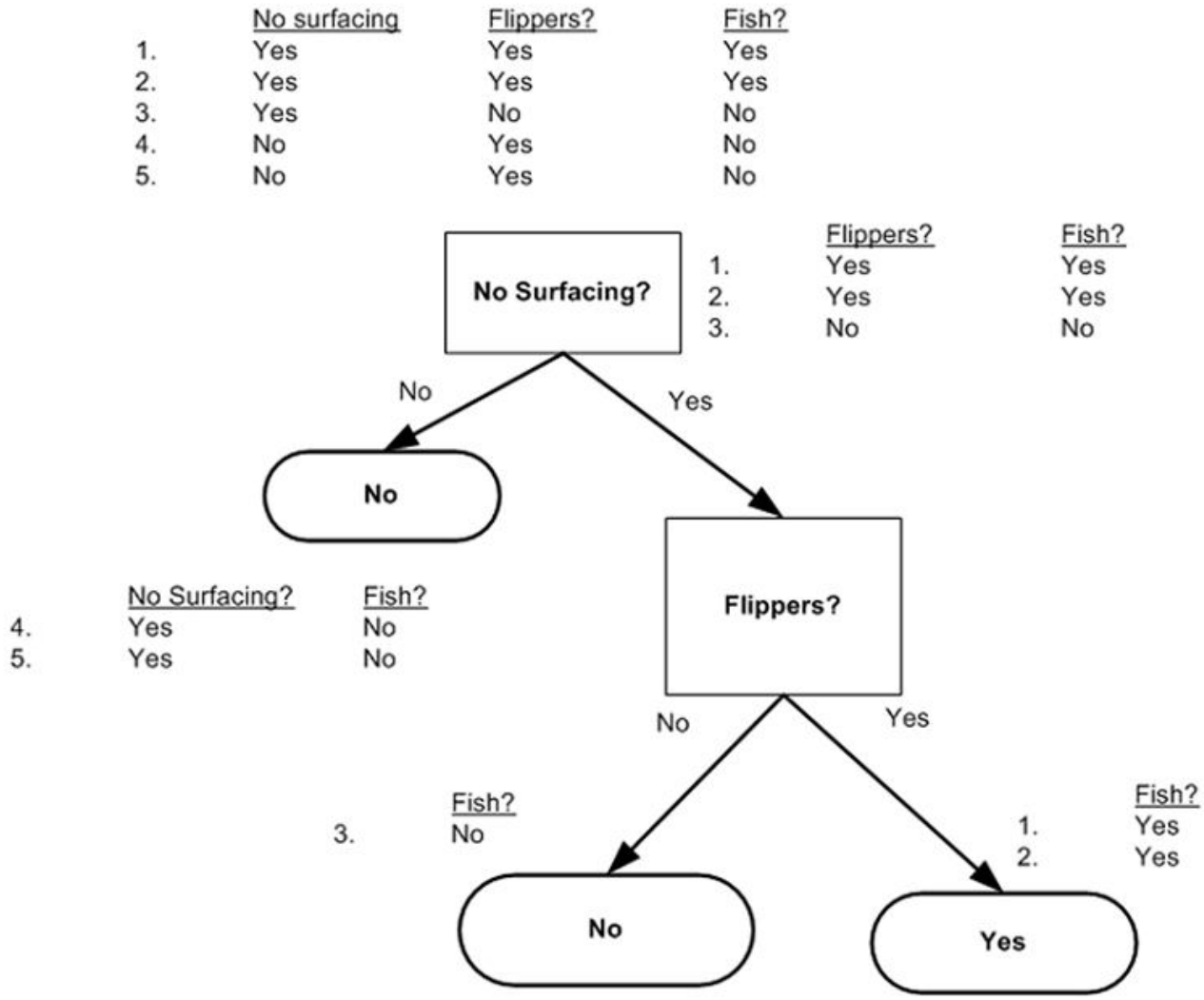

Figure 3.2 Data paths while splitting

Tree Building Code

Listing 3.4 Tree-building code

```
def createTree(dataSet,labels):
    classList = [example[-1] for example in dataSet]
    if classList.count(classList[0]) == len(classList):
        return classList[0]
    if len(dataSet[0]) == 1:
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)
    bestFeatLabel = labels[bestFeat]
    myTree = {bestFeatLabel:{}}
    del(labels[bestFeat])
    featValues = [example[bestFeat] for example in dataSet]
    uniqueVals = set(featValues)
    for value in uniqueVals:
        subLabels = labels[:]
        myTree[bestFeatLabel][value] = createTree(splitDataSet\
                        (dataSet, bestFeat, value),subLabels)
    return myTree
```

1. Stop when all classes are equal
2. When no more features, return majority
3. Get list of unique values

Test Decision Tree

Listing 3.8 Classification function for an existing decision tree

```
def classify(inputTree,featLabels,testVec):
    firstStr = inputTree.keys()[0]
    secondDict = inputTree[firstStr]
    featIndex = featLabels.index(firstStr)
    for key in secondDict.keys():
        if testVec[featIndex] == key:
            if type(secondDict[key]).__name__=='dict':
                classLabel = classify(secondDict[key],featLabels,testVec)
            else:   classLabel = secondDict[key]
    return classLabel
```

1 Translate label string to index

Example: Contact Lens Type Prediction

Example: using decision trees to predict contact lens type

1. Collect: Text file provided.
2. Prepare: Parse tab-delimited lines.
3. Analyze: Quickly review data visually to make sure it was parsed properly. The final tree will be plotted with `createPlot()`.
4. Train: Use `createTree()` from section 3.1.
5. Test: Write a function to descend the tree for a given instance.
6. Use: Persist the tree data structure so it can be recalled without building the tree; then use it in any application.

Tree Visualization

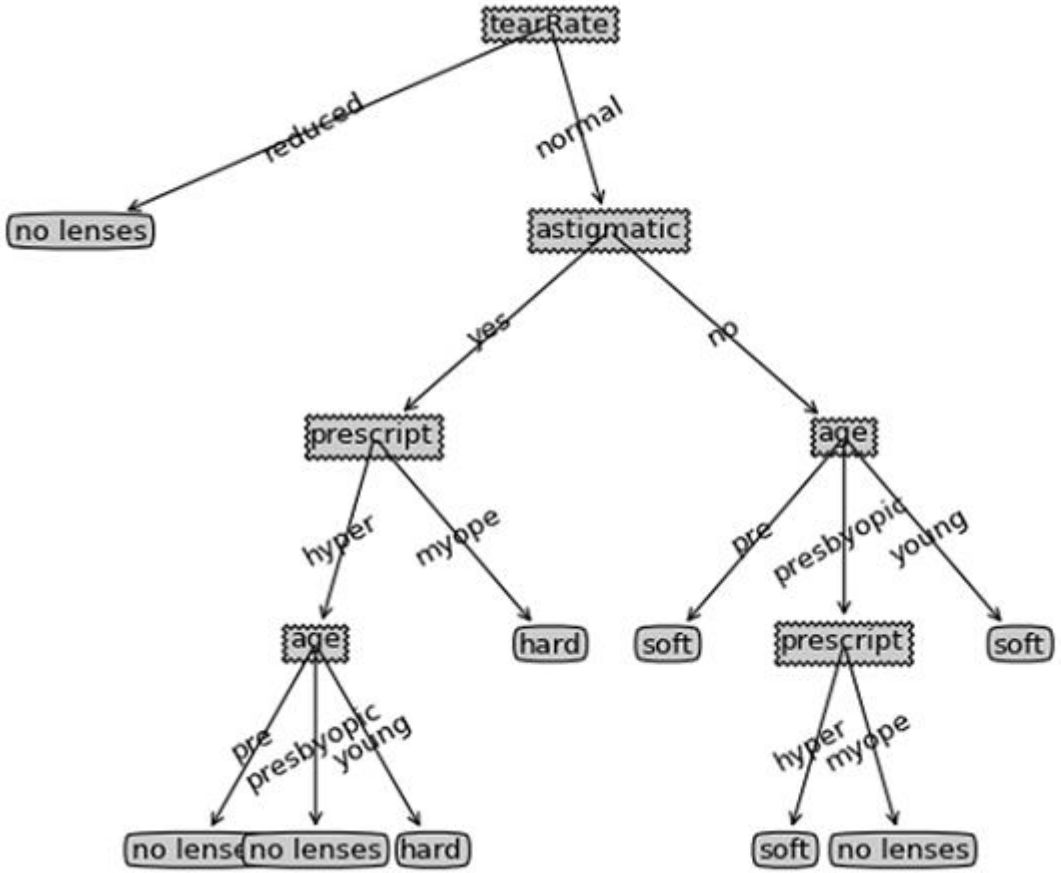

Figure 3.8 Decision tree generated by the ID3 algorithm