

# Machine Learning

## Decision Tree

### 1. Information theory

entropy:

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

where  $n$  is the number of classes.

information gain:

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_A \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

### 2. Model

input:

$$\text{dataset: } \{x^{(1)}, \dots, x^{(m)}\}, x^{(i)} \in \mathbb{R}^{n+1}$$

the last item in  $x^{(i)}$  is the class label.

choosing the best feature:

```

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

```

## majority:

```

def majorityCnt(classList):
    classCount={}
    for vote in classList:
        if vote not in classCount.keys(): classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.iteritems(),
        key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]

```

## create tree:

```

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

```

## classify:

```
def classify(inputTree, featLabels, testVec) :
    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)
    else: classLabel = valueOfFeat
    return classLabel
```

## 3. TODO

1. gini
2. continuity