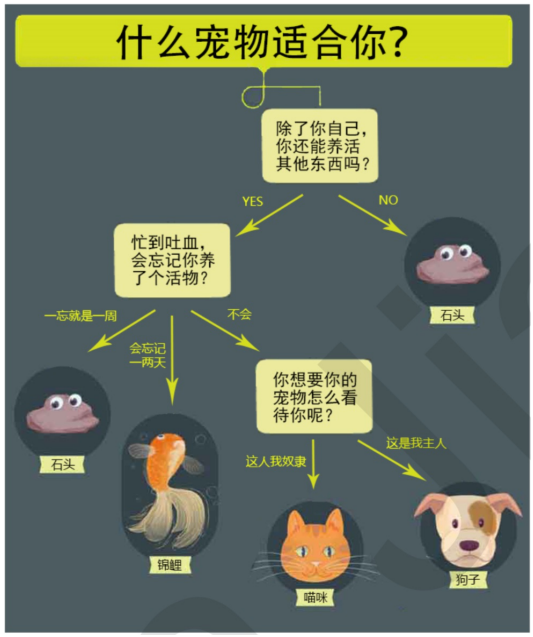
Table of Contents

决策树的概述

什么是决策树

一种对实例进行分类的树形结构，也可以看作是if-then规则的集合。if then规则集合又一个重要的性质：互斥且完备，即每一条路径都被一条规则所覆盖，而且只被这一条规则覆盖

接下来以选宠物为例，进行决策树的构建



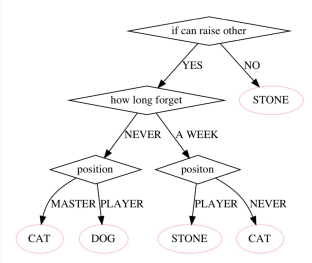
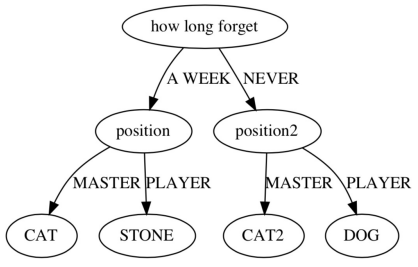
数据集

```
import pandas as pd
import numpy as np
original_data = pd.read_csv('recommendpet.csv')
original_data.drop(['id'],axis=1,inplace=True)
original_data
```

	if can raise other	how long will forget your pet	your pet's position in your heart	the pet suitable to you
0	NO	A WEEK	MY PLAYER	STONE
1	YES	A WEEK	MY PLAYER	STONE
2	YES	A WEEK	MY PLAYER	STONE
3	YES	A WEEK	MY MASTER	CAT
4	YES	A WEEK	MY PLAYER	STONE
5	YES	NEVER	MY MASTER	CAT
6	YES	NEVER	MY PLAYER	DOG

选择不同的特征建树

如果我们仅仅是根据上面的数据来建树的画，可以建树很多种树

if can raise	how long	position
		

可以看出选择的特征的顺序不同，构建出来的树的复杂程度不同

```
#画图的代码留一下
from graphviz import Digraph

dot = Digraph(comment='the root node is if can raise other')
dot.edge('position','Cat','MASTER')
dot.edge('position','how long forget','PLAYER')
dot.edge('how long forget','Dog','NEVER')
dot.edge('how long forget','Stone','A WEEK')
dot.view()
#dot.render('oneway', view=True)
```

```
'Digraph.gv.pdf'
```

特征选择

特征选择目的在于选取出对训练数据有分类能力的特征，提高决策树的学习效率。如果利用一个特征进行分类的结果与随机分类没有很大区别，那么这个特征是没有分类能力的

一般来说，随着划分的进行，我们希望决策树的分枝节点所包含的样本尽可能属于同一类别，也就是结点的纯度越来越高。（我的理解是，从这个结点中随机抽取一个样本，能很有把握判断这个样本属于哪个类别）

结点的样本分布很纯净：比如前两幅图，如果这个结点下的样本大体上都是属于同一个类别，那么随机从中抽取出一个样本，可以很有把握猜测出样本的类别

结点的样本分布不纯：如图三所示，这个结点下的样本一半属于类别1，一般属于类别2，那么随机从中抽取出一个样本，也就无法很有把握判断出抽取样本的类别

在特征选择时应用的标准有很多，常见的有GINI系数，增益率，entropy(信息熵)，选取不同的标准也就对应了不同的决策树的构建方法

信息熵

熵表示随机变量不确定性的度量

信息熵的计算公式

$$ Ent(D)=-\sum\limits_{i=1}^np(x_i)log_2p(x_i) $$
n表示样本集合D中一共有n类样本

比如在这个选宠物的栗子中

cat	2
dog	1
stone	4

我们来计算他的信息熵
$$ Ent(D)=-\frac{2}{7}log_2(\frac{2}{7})-\frac{1}{7}log_2(\frac{1}{7})--\frac{4}{7}log_2(\frac{4}{7})
$$

计算结果为1.3787834934861753

看到B站中有一个理解信息熵的视频
https://www.bilibili.com/video/av41539594?t=209

计算信息熵的python代码

```
'''
函数说明：计算信息熵
参数：dataSet
返回：信息熵
modify:2019-05-17
'''
def calEnt(dataSet):
    n = dataSet.shape[0]
    p = dataSet.iloc[:,-1].value_counts()/n
    Ent = (-p*np.log2(p)).sum()
    return Ent
```

```
#可以用刚才的计算结果来验证我们的函数
calEnt(original_data)
```

```
1.3787834934861753
```

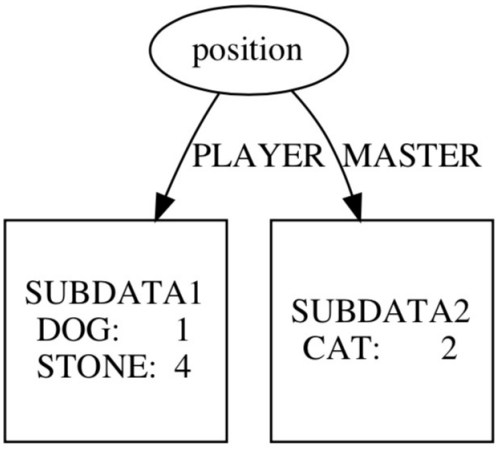
熵越高，信息的不纯度就越高，也就是混合的数据就越多

信息增益

我的理解是，在确定了某个特征后，对确定类别的种类提供的信息

所以信息增益的公式就为：
$$Gain(D,sub_D)=Ent(D)-Ent(sub_D)$$

对于子集的Ent的计算，以position为例



$$
\begin{align}
Ent(position) &= P_{player}Ent(position|player)+p_{master}Ent(position|master)\\
&= \frac{5}{7}Ent(SUBDATA1)+\frac{2}{7}Ent(SUNDATA2)
\end{align}
$$

故如果选择postion作为结点的话，提供的信息增益是
$$Gain(D,D_position)=Ent(D)-Ent(D_position)$$

如果可以确定position的值的话，信息增益是0.8631205685666308

数据集最佳切分函数

划分数据集最大的准则是选择最大的信息增益

```
'''
函数说明：让数据集根据某一特征值进行切分
参数：dataSet
返回：axis-数据集最佳切分列的索引
'''
def bestSplit(dataSet):
    baseEnt = calEnt(dataSet)
    m = dataSet.shape[0]
    bestGain = 0
    bestaxis = -1
    for i in range(dataSet.shape[1]-1):
        label = dataSet.iloc[:,i].value_counts().index #取出i特征下的类别
        sub_ent = 0
        for j in label:
            subData = dataSet[dataSet.iloc[:,i] == j]
            childEnt = calEnt(subData)
            sub_ent += subData.shape[0]/m*childEnt
        Gain = baseEnt - sub_ent
       # print(f'the gain is {Gain} when feature is {dataSet.columns[i]}')
        if Gain > bestGain:
            bestGain = Gain
            bestaxis = i
    return i
```

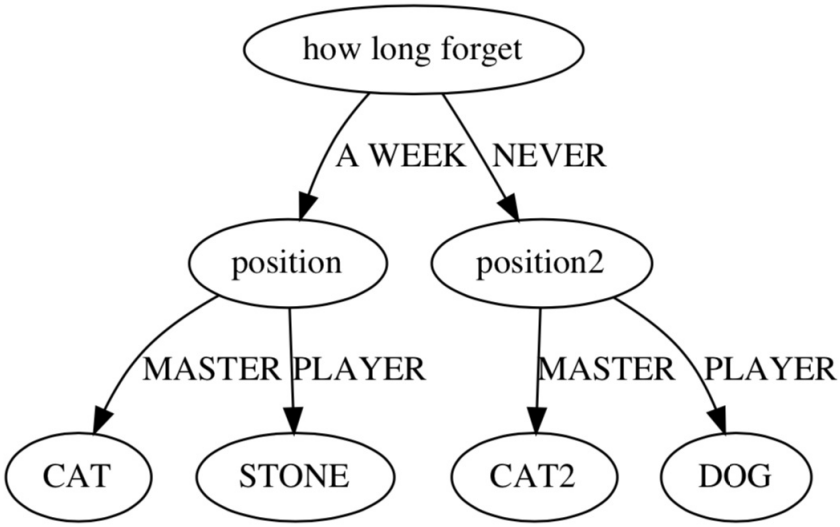
```
axis = bestSplit(original_data)
axis
```

```
2
```

特征 postion 的信息增益和刚才手算的结果也是一致的，验证函数正确

给定列切分数据集函数

以下图为例



如果我们确定根节点是how long forget后，接下来会分成Never和a week两个分支，对这个两个分支分别建树

比如在Never下选择下一个最佳的分类结点时，此时传入的数据集应是how long forget都是Never，同时，how long forget这个特征已经用掉，所以这个特征要被删除后的数据

即数据集

	if can raise other	how long will forget your pet	your pet's position in your heart	the pet suitable to you
0	NO	A WEEK	MY PLAYER	STONE
1	YES	A WEEK	MY PLAYER	STONE
2	YES	A WEEK	MY PLAYER	STONE
3	YES	A WEEK	MY MASTER	CAT
4	YES	A WEEK	MY PLAYER	STONE
5	YES	NEVER	MY MASTER	CAT
6	YES	NEVER	MY PLAYER	DOG

```
'''
函数说明：在给定列和标签后，对数据进行切分
参数：
    dataSet
```

```
    axis-给定的列
    label-给定的标签
返回:
    redata-切好后的数据
'''
def splitData(dataSet,axis,label):
    feature = dataSet.columns[axis]
    redata = dataSet[dataSet.iloc[:,axis] == label]
    redata = redata.drop(feature,axis=1)
    return redata
```

```
a = splitData(original_data,1,'NEVER')
a
```

	if can raise other	your pet's position in your heart	the pet suitable to you
5	YES	MY MASTER	CAT
6	YES	MY PLAYER	DOG

函数输出的结果和我们想要的数据是一样～开心

递归建树

构造决策树的算法很多，比如ID3，C4.5，CART，基于《机器学习实战》，我们先学习ID3

关于建树所需要的自功能模块已经全部写好

- def calEnt
- def bestSplit
- def splitData

工作原理:

1. 基于原始数据集最好的属性值划分数据集
2. 划分后将数据集传递到树的分支的下一个节点，在这个节点上，我们可以再次划分数据集
3. 直到遍历完所有的特征列，或者每个分支下的所有实例都具有相同的分类。

编写代码构建决策树

```
a={}
type(a)
a['test']='one'
a
```

```
{'test': 'one'}
```

```
'''
函数说明：基于最大的信息增益切分数据集，递归构造决策树
参数说明：dataSet
返回：mytree-字典形式的树
'''
def createTree(dataSet):
    if dataSet.shape[1]==1 or dataSet.iloc[:,-1].value_counts().index.shape[0]==1:
        return dataSet.iloc[:,-1].value_counts().index[0]
    bestfeature = bestSplit(dataSet)
    featurename = dataSet.columns[bestfeature]
    labellist = set(dataSet.iloc[:,bestfeature])
    lis = {}
    for label in labellist:
        subData = splitData(dataSet,bestfeature,label)
        lis[label] = createTree(subData)
    mytree = {featurename:lis}
    return mytree
```

```
mytree = createTree(original_data)
mytree
```

```
{'your pet’s position in your heart': {'MY MASTER': 'CAT',
  'MY PLAYER': {'how long will forget your pet': {'NEVER': 'DOG',
    'A WEEK': 'STONE'}}}}
```

存储决策树

使用的是numpy里面的save()函数，它可以直接把字典形式的数据保存为.npy文件，调用时用load()函数即可

```
#树的存储
```

```
np.save('mytree.npy',mytree)
```

```
#树的读取
read_mytree = np.load('mytree.npy').item()
read_mytree
```

```
{'your pet’s position in your heart': {'MY MASTER': 'CAT',
  'MY PLAYER': {'how long will forget your pet': {'NEVER': 'DOG',
    'A WEEK': 'STONE'}}}}
```

使用决策树分类

对一组数据进行分类

假设有一组测试数据

```
testData = original_data.iloc[6,:-1]
testData
```

```
if can raise other                          YES
how long will forget your pet              NEVER
your pet’s position in your heart    MY PLAYER
Name: 6, dtype: object
```

```
'''
函数说明：对一个测试样例进行分类
参数说明：
    inputTree-已经生成的决策树
    labels:特征标签
    testVec:测试数据列表，顺序对应原始数据集
返回：
    classLabel:分类结果
'''
def classify(inputTree,labels,testVec):
    feature = next(iter(inputTree)) #获取树的第一个结点
    secondDict = inputTree[feature]
    index_feature=labels.index(feature)
    f = testVec[index_feature]
    for key in secondDict:
```

```
        if f == key:
            if type(secondDict[key]) == dict:
                return classify(secondDict[key],labels,testVec)
            else:
                return secondDict[key]
```

```
a = classify(mytree,list(original_data.columns),testData)
a
```

```
'DOG'
```

对测试集进行预测，并返回预测后的结果

```
'''
函数说明：对测试集进行预测，并返回预测后的结果
参数说明：
    train:训练集
    test:测试集
返回：
    test:预测好分类的测试集
'''
def acc_classify(train,test):
    tree = createTree(train)
#     print(tree)
    m = test.shape[0]
    res = []
    for i in range(m):
        pre = classify(tree,list(train.columns),test.iloc[i,:])
        res.append(pre)
    test['predict'] = res
    acc = (test.iloc[:,-1] == test.iloc[:,-2]).mean()
    print(acc)
    return test
```

```
train = original_data
test = original_data.iloc[:,:]
list(train.columns)
acc_classify(train,test)
```

```
1.0
```

	if can raise other	how long will forget your pet	your pet’s position in your heart	the pet suitable to you	predict
0	NO	A WEEK	MY PLAYER	STONE	STONE
1	YES	A WEEK	MY PLAYER	STONE	STONE
2	YES	A WEEK	MY PLAYER	STONE	STONE
3	YES	A WEEK	MY MASTER	CAT	CAT
4	YES	A WEEK	MY PLAYER	STONE	STONE
5	YES	NEVER	MY MASTER	CAT	CAT
6	YES	NEVER	MY PLAYER	DOG	DOG

使用sklearn中的包完成决策树的预测

下个文档中再写～