机器学习笔记

# 机器学习基础

1. 概念：何为机器学习，将无序的数据转换成有用的信息
2. 数据获取：譬如可以在人们手机上装app，通过许多手机的磁力计得到信息
3. 术语：
   * 专家系统
   * 属性/特征
   * 分类
   * 目标变量(类别)
   * 训练数据和测试数据
4. 任务：
   1. 监督学习(知道预测什么，即目标变量的分类信息)
      * 分类：将实例数据划分到合适的分类中，譬如数据拟合曲线
      * 回归：主要用于预测数值型数据
   2. 无监督学习(数据无类别信息，不给目标值)
      * 聚类：数据集合分成类似对象组成的集合
      * 密度估计：寻找数据统计值的过程
      * 无监督学习还可以减少数据特征的维度，以便我们可以使用二维或三维图形展示数据
5. 开发步骤
   1. 收集数据
   2. 准备输入数据
   3. 分析输入数据
   4. 训练算法
   5. 测试算法
   6. 使用算法

# k-近邻算法(kNN)

## 概述

* 简单而言，k-近邻算法采用测量不同特征值之间的距离方法进行分类
* 工作原理：
  1. 存在一个样本数据集合(训练样本集)，其中每个数据存在标签
  2. 输入没有标签的新数据后，将新数据的每个特征与样本集中数据对应的特征进行比较
  3. 算法提取样本集中数据中最相似(最相邻)的分类标签，一般而言，我们只选择样本数据前k个最相似数据
  4. k即为k-近邻算法中k的出处，一般k取值不大于20

## 示例

### 场景一

收集到约会数据，保存在 'datingTestSet.txt' 中，每个样本数据占据一行，总共有1000行。 样本包含以下3种特征以及1种label：

* 每年获得的飞行常客里程数
* 玩视频游戏所耗时间百分比
* 每周消费冰淇淋公升数
* label分为三种：很喜欢，一般喜欢，不喜欢

#### 考虑

1. 算法采用 k 近邻算法
2. label可以依次用数字1,2,3代表很喜欢，一般喜欢以及不喜欢，假设这样的数据文件变为了 'datingTestSet2.txt'
3. 数据样本的前10%可以作为测试样本，剩余样本可以作为训练样本

#### 实现

1. 将数据读入，保存为一个 array 待用

* from numpy import \*  
    
   def file2matrix(filename):  
   fr = open(filename)  
   arrayOLines = fr.readlines()  
   m = len(arrayOLines)  
   returnMat = zeros((m,3))  
   classLabelVector = []  
   index = 0  
   for line in arrayOLines:  
   line = line.strip()  
   listFromLines = line.split('\t')  
   returnMat[index,:] = listFromLines[0:3]  
   classLabelVector.append(int(listFromLines[-1]))  
   index += 1  
   return returnMat, classLabelVector

1. 三种数据的数值范围差别过大，会导致对label的影响不一致，因此，需要对三种数据进行归一化

* from numpy import \*  
    
   def autoNorm(dataSet):  
   minVals = dataSet.min(0)  
   maxVals = dataSet.max(0)  
   ranges = maxVals - minVals  
   dataSetSize = dataSet.shape[0]  
   dataSet = dataSet - tile(minVals, (dataSetSize,1))  
   normDataSet = dataSet/tile(ranges, (dataSetSize,1))  
   return normDataSet, ranges, minVals

1. 算法实现

* from numpy import \*  
   import operator  
    
   def classify0(inX, dataSet, labels, k):  
   # 计算距离  
   m = dataSet.shape[0]  
   diffMat = tile(inX, (m, 1)) - dataSet  
   sqDiffMat = diffMat\*\*2  
   sqDistances = sqDiffMat.sum(axis=1)  
   distances = sqDistances\*\*0.5  
   # 将距离进行排序  
   # argsort() 函数，可以将 array 中的数值，按从小到大编号  
   sortedDistIndices = distances.argsort()  
   classCount = {}  
   for i in range(k):  
   # 前k个离测试点最近的label  
   voteIlabel = labels[sortedDistIndices[i]]  
   classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1  
   sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1), reverse=True)  
   return sortedClassCount[0][0]

1. 进行测试

* from numpy import \*  
    
   def datingClassTest():  
   # 前10%作为测试样本  
   hoRatio = 0.1  
    
   # 导入数据并归一  
   dataFile = '/home/curiousbull/Workspace/Python/machinelearninginaction/Ch02/datingTestSet2.txt'  
   datingDataMat, datingLabels = file2matrix(dataFile)  
   normMat, ranges, minVals = autoNorm(datingDataMat)  
    
   m = normMat.shape[0]  
   numTestVecs = int(m\*hoRatio)  
   errorCount = 0.0  
   for i in range(numTestVecs):  
   classifierResult = classify0(normMat[i,:], normMat[numTestVecs:m,:], \  
   datingLabels[numTestVecs:m], 3)  
   print("the classifier came back with: %d, the real answer is: %d" %(classifierResult, datingLabels[i]))  
   if(classifierResult != datingLabels[i]): errorCount += 1.0  
   print("the total error rate is: %f" %(errorCount/float(numTestVecs)))

### 场景二

目录digits下，分别有两个目录，testDigits和trainingDigits，代表用于测试的数据和训练的数据， 样本名包含需要识别的数字，也就是label，样本中是需要识别的手写数字数据集，每个单独文件都是32x32 的矩阵形式，包含 0 和 1

#### 考虑

1. 使用 k 近邻算法
2. 将文件名进行分割，得到对应的 labels 向量
3. 将每个文件内的32x32的0,1数字存储到一个vector，也就是一个1x1024的向量，作为识别的输入

#### 实现

1. 得到labels向量

* from numpy import \*  
   from os import listdir  
    
   def file2labels(dirName):  
   trainingFileList = listdir(dirName)  
   numberOfFiles = len(trainingFileList)  
   hwLabels = []  
   for i in range(numberOfFiles):  
   strTrainingFiles = trainingFileList[i].split('.')[0]  
   hwLabels.append(int(strTrainingFiles.split('\_')[0]))  
   trainingFileList[i] = dirName + trainingFileList[i]  
   return hwLabels, trainingFileList

1. 将img(32x32矩阵)转换为1x1024向量形式

* from numpy import \*  
   from os import listdir  
    
   def img2vec(filename):  
   fr = open(filename)  
   vecImg = zeros((1,1024))  
   for row in range(32):  
   strLine = fr.readline()  
   for col in range(32):  
   vecImg[0, 32\*row+col] = int(strLine[col])  
   return vecImg

1. 实现kNN算法

* from numpy import \*  
   from os import listdir  
   import operator  
    
   def classify0(inX, dataSet, labels, k):  
   dataSetSize = dataSet.shape[0]  
   diffMat = tile(inX, (dataSetSize, 1))-dataSet  
   sqDiffMat = diffMat\*\*2  
   sqDistances = sqDiffMat.sum(axis=1)  
   distances = sqDistances\*\*0.5  
   sortedDistIndicies = distances.argsort()  
   countLabels = {}  
   errorCount = 0.0  
   for i in range(k):  
   voteIlabel = labels[sortedDistIndicies[i]]  
   countLabels[voteIlabel] = countLabels.get(voteIlabel, 0) + 1  
   sortedCountLabels = sorted(countLabels.items(), key=operator.itemgetter(1),\  
   reverse=True)  
   return sortedCountLabels[0][0]

1. 测试样本

* def hwTest():  
   testDirName = 'digits/testDigits/'  
   trainingDirName = 'digits/trainingDigits/'  
   hwLabels, trainingFiles = file2labels(trainingDirName)  
   numberOfTrainingFiles = len(trainingFiles)  
   trainingDataSet = zeros((numberOfTrainingFiles, 1024))  
   for i in range(numberOfTrainingFiles):  
   trainingDataSet[i,:] = img2vec(trainingFiles[i])  
   testHwLabels, testFiles = file2labels(testDirName)  
   numberOfTestFiles = len(testFiles)  
   errorCount = 0.0  
   for i in range(numberOfTestFiles):  
   testVec = img2vec(testFiles[i])  
   classifierLabel = classify0(testVec, trainingDataSet, hwLabels, 3)  
   print("the classifer come back to: %d while the true value is: %d"\  
   %(classifierLabel, testHwLabels[i]))  
   if(classifierLabel != testHwLabels[i]): errorCount += 1.0  
   print("the total error classifiers number is: %f" %(errorCount/float(numberOfTestFiles)))  
   print("the error rate is: %f" %(errorCount/float(numberOfTestFiles)))

#### 注意点

1. 要使用 listdir() 函数，需要从 os 模块导入，但是不要将 os 模块中将所有内容都导入，以下代码

* from os import \*
* 会导致 open() 函数不可用

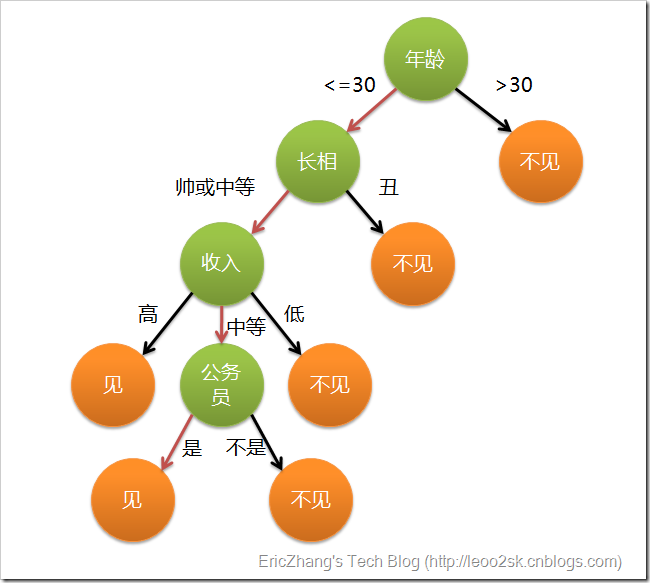
1. 注意 'labels' 从字符串中导入时，将字符串强转为 int 型

# 决策树

## 概述

#### 定义

决策树的分类思想类似姑娘找对象，相亲前，通过类似年龄、长相、收入，职业等将男人分为见或不见两类。类似下图：



从上图，可以总结出决策树的定义，即： 决策树（decision tree）是一个树结构（可以是二叉树或非二叉树）。其 每个非叶节点表示一个特征属性上的测试，每个分支代表这个特征属性在某个值域上的输出，而每个叶节点存放一个类别。 使用决策树进行决策的过程就是从根节点开始，测试待分类项中相应的特征属性，并按照其值选择输出分支，直到到达叶 子节点，将叶子节点存放的类别作为决策结果。

#### 构造

关键步骤在于分裂属性，即在某个节点处按照某一特征属性的不同划分构造不同的分支，目标是让各个分裂子集尽可能地 “纯”，即可能纯就是尽量让一个分裂子集中待分类项属于同一类别。分裂属性分三种情况：

1. 属性离散且不要求生成二叉树，此时用属性的每一个划分作为一个分支
2. 属性离散且要求生成二叉树，此时使用属性划分的一个子集作为测试，以“属于子集”和“不属于子集”分成两个分支
3. 属性值连续，确定一个值作为分裂点(split-point)，以 <split-point 和 >split-point 生成两个分支

#### ID3 算法(仅对于属性离散且不要求生成二叉树的情况讨论)

如果待分类的事物可能划分在多个分类中，譬如，设D为用类别对训练元组进行的划分

1. 符号 信息的定义：

* 其中表示第i个类别在整个训练元组出现的概率

1. D的熵(entropy)定义

* 熵的实际意义表示D中元组的类标号所需要的平均信息量。

1. 假设训练元组D按属性A划分，则该划分的熵计算如下：
2. 将D按属性A划分后，信息增益为两者的差值
3. 将D按所有属性进行划分，计算各种划分的信息增益，选择增益最大的属性进行划分，之后递归至
   * 划分后的所有分支内的元素都属于同一类别
   * 可用于划分的属性已使用完

## 示例

### 场景

有数据文件 'lenses.txt'，其中包含了如何预测患者需要佩戴的隐形眼镜类型数据。分为5列， 前四列为特征属性，最后一列为隐形眼镜的类型，也就是我们的 labels 特征依次为 '年龄(age)' '处方(prescript)' '是否散光(astigmatic)' '眼泪多少(tearRate)'

#### 考虑

1. 使用算法决策树解决问题
2. 数据导入(createDataSet())，返回数据(dataSet)和标签(labels)
3. 数据分割(splitDataSet())：按特征属性划分数据的时候，需要处理的数据(dataSet)是对应 该特征值的某个值的子dataSet
4. 考虑能够得到最大信息增益的特征，因此需要计算不同分割方法的熵(entropy)
5. 算法实现(createTree())：利用递归的方法，终止条件有两种：
   1. 按照当前特征划分的分支内所有元素都是同一类，即标签相同
   2. 用于划分的特征属性已经用完
6. 对应用于划分数据的特征属性用完后，如果某个分支内还有不同标签，处理方法可以效仿 KNN 算法，取该分支内占最多的label作为该分支的label
7. 测试

#### 实现

1. 数据导入 定义 createDataSet() 函数，返回 dataSet 和 labels 待用

* def createDataSet(filename):  
   fr = open(filename)  
   dataSet = [inst.strip().split('\t') for inst in fr.readlines()]  
   labels = ['age', 'prescript', 'astigmatic', 'tearRate']  
   return dataSet, labels

1. 数据分割 需要将数据中用于划分的列刨除

* def splitDataSet(dataSet, axis, value):  
   retDataSet = []  
   # 逐行输入  
   for featVec in dataSet:  
   if featVec[axis] == value:  
   reducedVec = featVec[0:axis]  
   reducedVec.extend(featVec[axis+1:])  
   retDataSet.append(reducedVec)  
   return retDataSet

1. 熵计算 对输入的dataSet，统计其所有事例数，统计不同label统计事例数 不同label统计事例数/总事例数=该label对应概率

* from math import log  
    
   def calcShannonEnt(dataSet):  
   numEntries = len(dataSet)  
   labelCount = {}  
   for featVec in dataSet:  
   currentLabel = featVec[-1]  
   labelCount[currentLabel] = labelCount.get(currentLabel, 0)+1  
   info = 0.0  
   for keys in labelCount.keys():  
   prob = float(labelCount[keys])/numEntries  
   info -= prob\*log(prob, 2)  
   return info

1. 挑选最佳分割路线，即确定最大信息增益的分割方式

* def chooseBestFeatToSplit(dataSet):  
   # 可用分割的特征数  
   numFeat = len(dataSet[0]) - 1  
    
   # 原始熵值  
   baseEntropy = calcShannonEnt(dataSet)  
    
   # 遍历可用特征，计算得到最大增益和与之匹配的特征编号  
   maxInfoGain = 0.0; bestFeat = -1  
   for i in range(numFeat):  
   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 > maxInfoGain:  
   maxInfoGain = infoGain  
   bestFeat = i  
    
   # 返回最大信息增益对应的特征编号  
   return bestFeat

1. 算法实现 利用熵计算，得到最佳的按特征分割方式，终止条件分为两种，如果是第二种终止条件，对该分支内 的元素采取类似kNN的做法，取多数的label作为该分支label
   1. 终止条件如果是用于分割数据的特征用完，需要定义一个处理分支内元素有不同label情况的函数
   * def majCnt(classList):  
      classCount = {}  
      for vote in classList:  
      classCount[vote] = classCount.get(vote, 0) + 1  
      # 注意，此处sortedLabelCount是一个tuple  
      sortedLabelCount = sorted(labelCount.items(), key=operator.itemgetter(1), reverse=True)  
      return sortedLabelCount[0][0]
   1. 利用递归实现 createTree() 方法，利用包含 dict 的 dict 的形式来存放 tree
   * def createTree(dataSet, labels):  
      # 首先应该考虑传入的数据是否还需要分割  
      # 获取数据集中所有标签  
      classList = [example[-1] for example in dataSet]  
      # 1. 分支内所有元素标签相同  
      if classList.count(classList[0]) == len(dataSet):  
      return dataSet  
       
      # 2. 所有可用分割特征使用结束  
      if len(dataSet[0]) == 1:  
      majCnt(classList)  
       
      # 以上终止条件不满足，对dataSet进行划分  
       
      # 最佳分割特征对应编号  
      bestFeat = chooseBestFeatToSplit(dataSet)  
      bestFeatLabel = labels[bestFeat]  
       
      # 利用特征定义树  
      myTree = {bestFeatLabel:{}}  
       
      # 将labels中此次最佳分割特征去除，在剩余特征中重新选择分割特征  
      del(labels[bestFeat])  
       
      # 该特征对应取值  
      featValues = [example[bestFeat] for example in dataSet]  
      uniqueValues = set(featValues)  
       
      # 按特征属性值进行分割  
      for value in uniqueValues:  
      subLabels = labels[:] # 尤其注意这句话的含义？  
      myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), subLabels)  
       
      # 返回树  
      return myTree

## 使用 matplotlib 画树图

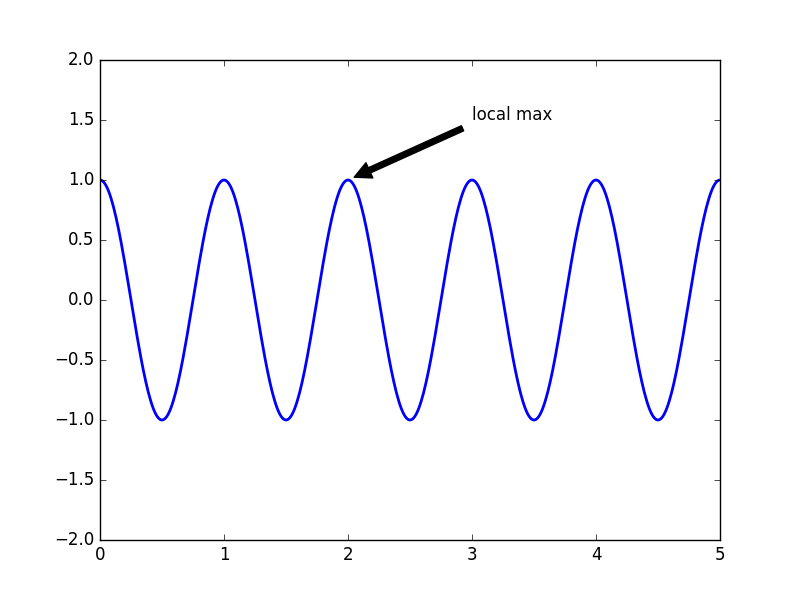
### 关于注释

#### 注释文字 (Annotation Text)

text() 函数可以设置在哪个位置坐标系任意位置添加文本。 而 annotate() 函数可以在添加文本 的基础上，提供更多丰富的功能，使得添加注释更加容易。 在注释中，最重要的是两个参数，一个是参数 'xy'，它代表需要被注释的点所在位置；另一个参数是 'xytext'，代表注释文本所在位置。

import numpy as np  
 import matplotlib.pyplot as plt  
  
 fig = plt.figure()  
 ax = fig.add\_subplot(111)  
  
 t = np.arange(0.0, 5.0, 0.01)  
 s = np.cos(2\*np.pi\*t)  
 line, = ax.plot(t, s, lw=2)  
  
 ax.annotate('local max', xy=(2, 1), xytext=(3, 1.5),  
 arrowprops=dict(facecolor='black', shrink=0.05),  
 )  
  
 ax.set\_ylim(-2,2)  
 plt.show()

上述代码会产生如下图所示效果：



#### 坐标系 (coordinate system)

在上述代码中， annotate() 函数没有指定坐标系，默认 'xy' 'xytext' 所代表的位置为数据坐标系， 实际应用中，我们可以按需求任意指定坐标系，可用的坐标系可见下表：

| argument | coordinate system |
| --- | --- |
| 'figure points' | points from the lower left corner of the figure |
| 'figure pixels' | pixels from the lower left corner of the figure |
| 'figure fraction' | 0,0 is lower left of figure and 1,1 is upper right |
| 'axes points' | points from lower left corner of axes |
| 'axes pixels' | pixels from lower left corner of axes |
| 'axes fraction' | 0,0 is lower left of axes and 1,1 is upper right |
| 'data' | use the axes data coordinate system |

譬如， 注释位置用 'fractional axes’ coordinates，可以

ax.annotate('local max', xy=(3,1), xycoords='data', xytext=(0.8, 0.95),\  
 textcoords='axes fraction', arrowprops=dict(facecolor='black', shrink=0.05),\  
 horizontalalignment='right',verticalalignment='top')

#### 箭头性质 (arrow properties)

箭头性质也可以用下表的参数进行指定

| arrowprops key | description |
| --- | --- |
| width | the width of the arrow in points |
| frac | the fraction of the arrow length occupied by the head |
| headwidth | the width of the base of the arrow head in point |
| shrink | move the tip and base some percent away from the annotated point and text |
| \*\*kwargs | any key for matplotlib.patches.Polygon, e.g., facecolor |

#### 线条性质

最简单方式，利用以下代码查看可用线条性质设置

lines = plt.plot([1,2,3])  
 plt.setp(lines)

#### 注释轴 (Annotation Axes)

##### Annotating with Text with Box

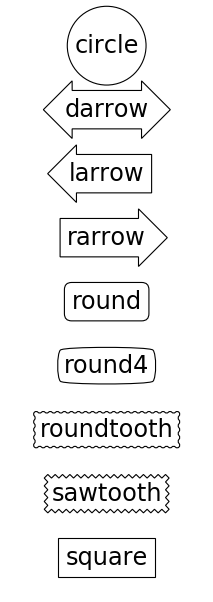
text() 会接受 'bbox' 关键词参数，当接受到 'bbox' 参数时，注释文字会被一个 'box' 包围

bbox\_props = dict(boxstyle='rarrow, pad=0.3', fc='cyan', ec='b', lw=2)  
 t = ax.text(0, 0, "Direction", ha='center', va='center', rotation=45, size =15,\  
 bbox=bbox\_props,)

下表是 'box' 可接受的参数与属性值

| Class | Name | Attrs |
| --- | --- | --- |
| Circle | circle | pad=0.3 |
| DArrow | darrow | pad=0.3 |
| LArrow | larrow | pad=0.3 |
| RArrow | rarrow | pad=0.3 |
| Round | round | pad=0.3 |
| Round4 | round4 | pad=0.3,roundingsize=Non |
| Roundtooth | roundtooth | pad=0.3,toothsize=None |
| Sawtooth | sawtooth | pad=0.3,toothsize=None |
| Square | square | pad=0.3 |

与上表对应的图如下：

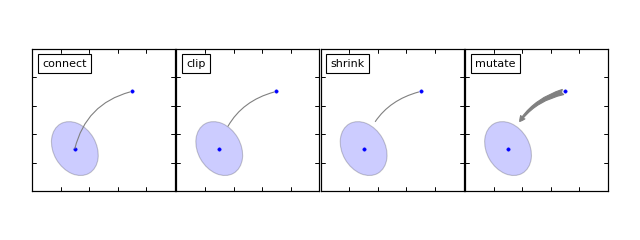


##### Anonotating with Arrow

annotate() 函数在 pyplot 中是用来连接两个点的 (注释点与被注释点)。 画 'arrow' 主要分为以下几步：

1. 确定两点之间的连接路径，可以通过 connectionsytle 来控制
2. 假如 'patch object' (画块) 给定 (patchA & patchB)，路径会被裁剪，避开画块
3. 路径按照给定的 'pixels' 数值进行 'shrunk'
4. 路径变形为箭头状，通过 arrowstyle 控制属性

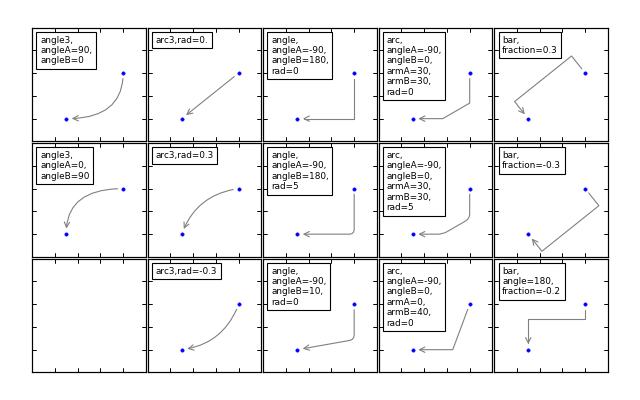
以上步骤总结在下图中： 注意：



1. connectionstyle 可用的选项如下表：

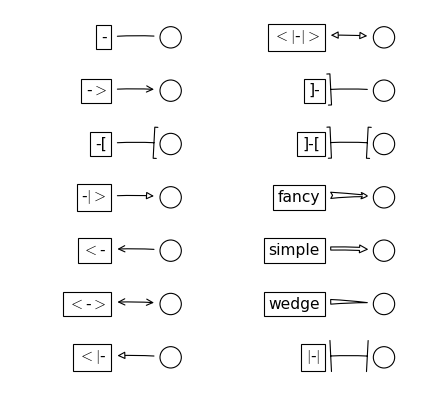
| name | Attrs |
| --- | --- |
| angle | angleA=90, angleB=0, rad=0.0 |
| angle3 | angleA=90, angleB=0 |
| arc | angleA=0, angleB=0, armA=None, armB=None, rad=0.0 |
| arc3 | rad=0.0 |
| bar | armA=0.0,armB=0.0,fraction=0.3,angle=None |

其中 'angle3' 和 'arc3' 中的 3 表示其指定的路径样式为二次样条线段，有 3 个控制点，上述格式对应下图：



1. arrowstyle 可用选项如下：

| Name | Attrs |
| --- | --- |
| - | None |
| -> | headlength=0.4,headwidth=0.2 |
| -[ | widthB=1.0, lengthB=0.2, angleB=None |
| |-| | widthA=1.0, widthB=1.0 |
| -|> | headlength=0.4, headwidth=0.2 |
| <- | headlength=0.4, headwidth=0.2 |
| <-> | headlength=0.4, headwidth=0.2 |
| <|- | headlength=0.4, headwidth=0.2 |
| <|-|> | headlength=0.4, headwidth=0.2 |
| fancy | headlength=0.4, headwidth=0.4, tailwidth=0.4 |
| simple | headlength=0.5, headwidth=0.5, tailwidth=0.2 |
| wedge | tailwidth=0.3, shrinkfactor=0.5 |

* 具体样式见下图：
* 
* 有些 arrowstyle 仅与能生成二次样条线段的 connectionstyle 配合，这些 arrowstyle 是 'fancy', 'simple', 'wedge'

### 树节点构造

终止块(叶节点)使用 boxstyle=round4 决策节点用 boxstyle=sawtooth

import matplotlib.pyplot as plt  
  
 # 决策节点样式  
 decisionNode = dict(boxstyle='sawtooth', fc='0.8')  
  
 # 叶节点  
 leafNode = dict(boxstyle='round4', fc="0.8")  
  
 # arrow样式  
 arrow\_args = dict(arrowstyle='<-')  
  
 def plotNode(nodeTxt, centerPt, parentPt, nodeType):  
 createPlot.ax1.annotate(nodeTxt, xy=parentPt, xycoords='axes fraction',\  
 xytext=centerPt, textcoords='axes fraction',\  
 va='center', ha='center', bbox=nodeType, arrowprops=arrow\_args)  
  
 def createPlot():  
 fig = plt.figure(1, facecolor='white')  
 fig.clf()  
 createPlot.ax1 = plt.subplot(111, frameon=False)  
 plotNode(U'决策节点', (0.5, 0.1), (0.1, 0.5), decisionNode)  
 plotNode(U'叶节点', (0.8, 0.1), (0.3, 0.5), leafNode)  
 plt.show()

### 树大小确定

1. 宽度：与叶节点数目有关

* def getNumLeaf(myTree):  
   numLeafs = 0  
    
   # 使用递归方式获取叶节点  
   firstStr = myTree.keys()[0]  
   secondDict = myTree[firstStr]  
   for key in secondDict.keys():  
   # 创建myTree的时候，终止条件下，返回的是 dataSet 或 label  
   # 这里递归取 dict[key]，终止条件是取到 label  
   if type(secondDict[key]).\_\_name\_\_ == 'dict':  
   numLeafs += getNumLeaf(secondDict[key])  
   else: numLeafs += 1  
   return numLeafs

1. 高度：与决策节点数目有关

* def getTreeDepth(myTree):  
   maxDepth = 0  
   firstStr = myTree.keys()[0]  
   secondDict = myTree[firstStr]  
   for key in secondDict.keys():  
   if type(secondDict[key]).\_\_name\_\_ == 'dict':  
   thisDepth = 1 + getTreeDepth(secondDict[key])  
   else: thisDepth = 1  
   return maxDepth

### 绘制树

# 父子节点直接插入文本  
 def plotMidText(cntrPt, parentPt, txtString):  
 xMid = ((parentPt[0]-cntrPt[0])/2.0) + cntrPt[0]  
 yMid = ((parentPt[1]-cntrPt[1])/2.0) + cntrPt[1]  
 createPlot.ax1.text(xMid, yMid, txtString)  
  
 # 绘制树  
 def plotTree(myTree, parentPt, nodeTxt):  
 # 计算树的宽高  
 numLeafs = getNumLeafs(myTree)  
 depth = getTreeDepth(myTree)  
 firstStr = myTree.keys()[0]  
 cntrPt = (plotTree.xOff + (1.0+float(numLeafs))/2.0/plotTree.totalW,\  
 plotTree.yOff)  
 plotMidText(cntrPt, parentPt, nodeTxt)  
 plotNode(firstStr, cntrPt, parentPt, dicisionNode)  
 secondDict = myTree[firstStr]  
 plotTree.yOff = plotTree.yOff - 1.0/plotTree.totalD  
 for key in secondDict.keys():  
 if type(secondDict[key]).\_\_name\_\_ == 'dict':  
 plotTree(secondDict[key], cntrPt, str(key))  
 else:  
 plotTree.xOff = plotTree.xOff + 1.0/plotTree.totalW  
 plotNode(secondDict[key], (plotTree.xOff, plotTree.yOff), \  
 cntrPt, leafNode)  
 plotMidText((plotTree.xOff, plotTree.yOff), cntrPt, str(key))  
 plotTree.yOff = plotTree.yOff + 1.0/plotTree.totalD  
  
 def createPlot(inTree):  
 fig = plt.figure(1, facecolor='white')  
 fig.clf()  
 axprops = dict(xticks = [], yticks=[])  
 createPlot.ax1 = plt.subplot(111, frameon=False, \*\*axprops)  
 plotTree.totalW = float(getNumLeafs(inTree))  
 plotTree.totalD = float(getTreeDepth(inTree))  
 plotTree.xOff = -0.5/plotTree.totalW; plotTree.yOff = 1.0  
 plotTree(inTree, (0.5, 1.0), '')  
 plt.show()

# 基于概率论的分类方法：朴素贝叶斯

## 概述

### 概念

事物有许多属性，譬如一个事物 $X$，可以定义其一系列属性

\begin{equation}  
 $X = (x\_1, x\_2, x\_3, \ldots)$  
 \end{equation}

作为其描述，而不同事物可能归属不同种类，可以用集合 作为各种不同种类的描述，

\begin{equation}  
 Y = y\_1, y\_2, \ldots, y\_m  
 \end{equation}

当给出任意一个事物 的时候，我们需要预测这个事物到底属于哪个种类 $y′$。 朴素贝叶斯理论做了一个假设，认为任意一个实例不同属性之间相互独立，这样，可以计算一个实例，其处于 不同种类之间的概率，选择概率最大的一个种类作为该实例种类的预测。如果定义 作为 的 后验概率，表示实例 属于种类 的概率， 称为 的先验概率， 通过计算不同 的后验概率，其中数值最大对应 可以作为 种类的预测。

实际我们可以直接得到的，一般是 的后验概率 (相当于 的属性出现在 类的概率总和) 和 的先验概率，但是，通过贝叶斯公式，我们可以求得 的先验概率：

\begin{equation}  
 P(y\_i|X) = \frac{P(X|y\_i)P(y\_i)}{P(X)}  
 \end{equation}

### 特点

* 优点：数据较少的情况下仍然有效，可以处理多类别问题
* 缺点：对于输入数据的准备方式较为敏感
* 适用数据类型：标称型数据

### 举例说明

#### 说明

考虑一个医疗诊断问题，有两种可能假设： 1). 病人有癌症 2). 病人没癌症。样本数据来自某化验测试，测试结果有两种 1). 阳性 2). 阴性。 假如我们已经知道了普通人中只有 0.008 的人会患病。此外，化验结果对有病的患者有 98% 的可 能返回阳性结果，对无病患者有 97% 概率返回阴性结果。此时，有一个病人化验测试结果时阳性，是否可以将病人诊断为得了 癌症。

#### 考虑

\begin{alignat}{2}  
 \P(canser) &= 0.008 &\quad P(no canser) &= 0.992 \\   
 \P(positive|canser) &= 0.98 &\quad P(negative|canser) &= 0.02 \\   
 \P(positive|no canser) &= 0.03 &\quad P(negative|no canser) &= 0.97   
 \end{alignat}

这样，通过贝叶斯公式，我们可以求得该病人测试结果为阳性时，其得癌与不得癌的概率：

\begin{eqnarray}  
 P(canser|positive) &=& \frac{P(positive|canser)P(canser)}{P(positive)}\\\nolinenumber  
 {} & = & \frac{0.98\times{}0.008}{P(positive)}\\\nolinenumber  
 {} & = & \frac{0.0078}{P(positive)}\\  
 P(no canser|positive) & = & \frac{P(positive|no canser)P(no canser)}{P(positive)}\\\nolinenumber  
 {} & = & \frac{0.03\times{}0.992}{P(positive)}\\\nolinenumber  
 {} & = & \frac{0.0298}{P(positive)}  
 \end{eqnarray}

可以看出，该患者不得癌的概率更大，按照朴素贝叶斯的方法，我们认为其没有癌症。除此之外，我们还可以看出，我们不需要关系贝叶斯 公式的分母部分，只需要考察分子部分大小即可。

## 示例

### 场景一 利用朴素贝叶斯方法过滤垃圾邮件

数据集包含50封邮件，其中25封为垃圾邮件，随机将数据集切分为训练集(40封)和测试集(10封)。邮件目录为 'email'，下面又分为两个目录，分别为 'ham' 和 'spam'，存放普通邮件与垃圾邮件。

#### 考虑

1. 实例与类别的考虑：实例是邮件，类别是 'spam' 和 'ham'
   * 一个邮件相当于一个实例，其由一个个单词组成，因此，可以考虑用单词作为其属性的表述
   * 很多邮件的单词会有重叠，用所有英文单词作为属性去描述邮件，工作量太大，因此，可以考虑将用于训练的邮件中 所有不重复的单词作成一个词汇表，词汇表中的单词作为不同邮件的描述
   * 一个词汇表相当于 $(x1, x2, …, xm)$，一个邮件相当于一个实例，实例对应的属性(词汇表中的单词)值 就是词汇表中不同单词出现的次数，譬如：
   * vocabList = ['hello', 'world', 'sun', 'moon', 'good'] # 5 个属性  
      docWordList1 = ['ni', 'hao', 'sun', 'is', 'big', 'world', 'good', 'world']  
      docWordList2 = ['hello', 'beautiful', 'girl']  
      docWordList1 -- vectorize --> [0, 2, 1, 0, 1] # 用 5 个属性表述结果  
      docWordList2 -- vectorize --> [1, 0, 0, 0, 0] # 用 5 个属性表述结果
2. 词汇表的创建分为两步：
   1. 文档的分割，变成 '词汇'(words) 的组合
   2. 所有文档的词汇组合，创建词汇表
3. 实例的表述
   1. 初始化一个长度等于 'vocabList' 的 'vector'
   2. 对应每个属性，输入实例的属性值
4. 算法实现
   1. 输入参数为训练集(所有文档表述组合成的matrix)和labels
   2. 计算每个属性对应的先验概率

#### 实现

1. 词汇表创立与文档描述
   1. 文档分割
   * import re  
       
      def loadEmailFile(filename):  
      fr = open(filename)  
       
      # 分割邮件为单词的组合，去除空格与标点  
      regEx = re.compile('\W\*')  
      wordList = regEx.split(fr.read())  
       
      # 考虑到可能出现的网址，有可能出现类似 py 等单词，需要这种情况排除  
      return [tok.lower() for tok in wordList if len(tok) > 2]
   1. 原始文档集
      * 将各文档列出
      * 分割单词
      * 每个文档作为一个 vector, 填入分割出来的单词
      * from numpy import \*  
         from os import listdir  
         from os import path  
         import re  
          
         def loadDoc():  
         spamDir = "./email/spam"  
         hamDir = "./email/ham"  
         spamMailList = []  
         hamMailList = []  
         spamMailList = [path.join(spamDir, spamEmail) for spamEmail in listdir(spamDir)]  
         hamMailList = [path.join(hamDir, hamEmail) for hamEmail in listdir(hamDir)]  
          
         # 定义 docList 存放原始 email 内容  
         docList = []  
         # 同时按照 doc 顺序存入 label， 1 表示 'spam'， '0' 表示 'ham'  
         classList = []  
         for mail in spamMailList:  
         wordList = loadEmailFile(mail)  
         docList.append(wordList)  
         classList.append(1)  
         for mail in hamMailList:  
         wordList = loadEmailFile(mail)  
         docList.append(wordList)  
         classList.append(0)  
         return docList, classList
   2. 词汇表创建
   * # 这里 docList 是不同文档原始单词表示的 vector  
      def createVocabList(docList):  
      vocabSet = set([])  
      for document in docList:  
      vocabSet = vocabSet | set(document)  
      return list(vocabSet)
   1. 原始文档集用词汇表表示
   * def bagOfWords2VecMN(vocabList, inputSet):  
      # 初始化词向量，每个元素对应词汇表中的一个单词，初始值为 0  
      returnVec = [0] \* len(vocabList)  
       
      # 遍历输入的邮件，每遇到一个词， 词向量对应值加 1  
      for word in inputSet:  
      if word in vocabList:  
      returnVec[vocabList.index(word)] += 1  
      return returnVec
2. 算法实现

* def trainNB0(trainMatrix, trainCategory):  
   # 文档数量  
   numTrainDocs = len(trainMatrix)  
   # 属性数量  
   numWords = len(trainMatrix[0])  
    
   # 初始化  
   # 'spam' 对应的先验概率  
   # 需要注意的是， 'ham' 对应的 'traincategory' 为 0，因此 sum(trainCategory) 是 'spam' 数目  
   pAbusive = sum(trainCategory)/float(numTrainDocs)  
   # 原本是定义一个初始 list， 用来存放 'ham' 和 'spam' 对应属性值的和  
   # 正常是 p0Num = zeros(numWords); p1Num = zeros(numWords)  
   # 考虑到可能出现某属性值的概率为 0，因此用 1 作为初始向量  
   p0Num = ones(numWords); p1Num = ones(numWords)  
   # 对应的 'ham' 与 'spam' 类别对应的总数用 p0Denom 与 p1Denom 表示  
   # 初始值也应该设置为 0.0，考虑到 p0Num 与 p1Num 已经设置为 (1,...) vector  
   # 将 p0Denom 与 p1Denom 设置为一个不为 1 的数  
   p0Denom = 2.0; p1Denom = 2.0  
    
   # 遍历文档，计算每个属性值的概率  
   for i in list(range(numTrainDocs)):  
   # 判断 'spam'  
   if trainCategory[i] == 1:  
   # 对应的存放 'spam' vector，其加上标志为 'spam' 邮件的实例  
   # 标志为 'spam' 邮件的实例表示类似 [1, 0, 1, 0, ...]  
   # 按文档相加，最后得到的是每个属性出现次数  
   p1Num += trainMatrix[i]  
   # 相应的，将标志为 'spam' 的文档所有属性相加  
   # 遍历文档后，这个值是对应 'spam' 种类中所有属性值之和  
   p1Denom += sum(trainMatrix[i])  
   else:  
   p0Num += trainMatrix[i]  
   p0Denom += sum(trainMatrix[i])  
    
   # 概率表示用 log 型  
   p1Vec = log(p1Num/p1Denom)  
   p0Vec = log(p0Num/p0Denom)  
    
   return p0Vec, p1Vec, pAbusive

1. 给定单词向量，进行分类

* def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):  
   # 概率用 log 值表示  
   # 一个实例 vec2Classify，其表示为各个属性(单词表向量)的数量值  
   # 将实例 vec2Classify 乘以 我们利用训练集得到的每个属性的概率值  
   # 然后乘以对应 class 的先验概率就可以得到我们需要求的概率值  
   # log 型概率求和对应原始概率乘积  
   p1 = sum(vec2Classify\*p1Vec) + log(pClass1)  
   p0 = sum(vec2Classify\*p0Vec) + log(1.0-pClass1)  
   if p1 > p0:  
   return 1  
   else:  
   return 0

1. 测试算法

* def spamTest():  
   # 原始文档，对应标签向量  
   docList, classList = loadDoc()  
   # 词汇表  
   vocabList = createVocabList(docList)  
    
   # 随机从 docList 中抽取 10 个 vector 作为测试  
   trainingList = range(50); testList = []  
   for i in list(range(10)):  
   randomIdx = int(random.uniform(0, len(trainingList)))  
   testList.append(trainingList[randomIdx])  
   # 将测试邮件从训练集中删除  
   del(trainingList[randomIdx])  
    
   # 构建训练算法所需要的参数  
   trainMat = []; trainClasses = []  
   for docIdx in trainingList:  
   trainMat.append(bagOfWords2VecMN(vocabList, docList[docIdx]))  
   trainClasses.append(classList[docIdx])  
    
   # 执行训练算法，获得概率向量  
   # 注意将 list 转换为 mat  
   p0V, p1V, pSpam = trainNB0(array(trainMat), array(trainClasses))  
    
   errorCount = 0.0  
   for docIdx in testList:  
   wordVector = bagOfWords2VecMN(vocabList, docList[docIdx])  
   if classifyNB(array(wordVector), p0V, p1V, pSpam) != classList[docIdx]:  
   errorCount += 1  
   print("classification error: ", docList[docIdx])  
   # 打印错误率  
   print('the error rate is: ', float(errorCount/len(testList)))

### 场景二 朴素贝叶斯分类器从个人广告中获取区域倾向

* 不同城市的人发布的征婚广告，通过分析网站 'Cragslist' 上的RSS信息，我们可以比较不同城市对于 征婚关心的内容差异。分析RSS信息的工具可以采用 Python 的 feedparser 库

#### 考虑

* 可用的 Rss 源
* 不同城市对应的 Rss 信息描述
  + 属性值一样用一个 'vocabList' 来描述
  + 'vocabList' 创建方式与场景一雷同
  + 分割 xml 网页的工具，使用 feedparser , 介绍可见网页 [python解析RSS(feedparser)](http://www.cnblogs.com/MrLJC/p/3731213.html)
* 朴素贝叶斯算法实现
* 需要去除词频最高的词语，取排行前 30 的单词

#### 实现

* 分词器
* import re  
    
   def loadFromRSS(feed):  
   data = feed['summary']  
   regEx = re.compile('\W\*')  
   wordList = regEx.split(data)  
   return [tok.lower() for tok in wordList if len(tok) > 2]
* 创建词汇表
* def createVocabList(docList):  
   vocabSet = set([])  
   for document in docList:  
   vocabSet = vocabSet | set(document)  
   return list(vocabSet)
* 文档描述
  + 与场景一处理方式相同，但是考虑到要挑选出词频最高单词，多加一个 'fulltexts' 的向量 用来存放所有的词汇
  + from numpy import \*  
     from os import listdir  
     from os import path  
     import re  
      
     def loadDoc():  
     ny = feedparser.parser("http://newyork.craiglist.org/search/stp?format=rss")  
     sf = feedparser.parser("http://sfbay.craiglist.org/search/stp?format=rss")  
      
     docList = []; classList = []; fullText = []  
     minLen = min(len(ny['entries']), len(sf['entries']))  
     # 注意 docList 与 fullText 区别  
     # 前者以 vector 为一个 entry， 后者以一个单词为一个 entry  
     for i in range(minLen):  
     wordList = loadFromRSS(ny['entries'][i])  
     docList.append(wordList)  
     fullText.extend(wordList)  
     classList.append(1)  
     wordList = loadFromRSS(sf['entries'][i])  
     docList.append(wordList)  
     fullText.extend(wordList)  
     classList.append(0)  
     return docList, classList, fullText, minLen
* 贝叶斯算法训练函数，与场景一相同
* 分类器，与场景一相同
* 测试
* def loacalWords():  
   # 原始文档，对应标签向量  
   docList, classList, fullText, minLen = localDoc()  
   # 词汇表  
   vocabList = createVocabList(docList)  
    
   # 去除高频词汇  
   top30Words = calcMostFreq(vocabList, fullText)  
   for pairW in top30Words:  
   if pairW[0] in vocabList: vocabList.remove(pairW[0])  
    
   # 初始化训练数据集和测试数据集  
   trainingList = range(2\*minLen); testList = []  
   for i in range(10):  
   randomIdx = int(random.unifor(0, len(trainingList)))  
   testList.append(trainingList[randomIdx])  
   # 将测试广告从训练集中删除  
   del(trainingList[randomIdx])  
    
   # 构建训练算法所需要的参数  
   trainMat = []; trainClasses = []  
   for docIdx in trainingList:  
   trainMat.append(bagOfWords2VecMN(vocabList, docList[docIdx]))  
   trainClasses.append(classList[docIdx])  
    
   # 执行训练算法，获得概率向量  
   # 注意将 list 转换为 mat  
   p0V, p1V, pSpam = trainNB0(array(trainMat), array(trainClasses))  
    
   errorCount = 0.0  
   for docIdx in testList:  
   wordVector = bagOfWords2VecMN(vocabList, docList[docIdx])  
   if classifyNB(array(wordVector), p0V, p1V, pSpam) != classList[docIdx]:  
   errorCount += 1  
   print("classification error: ", docList[docIdx])  
   # 打印错误率  
   print('the error rate is: ', float(errorCount/len(testList)))