# 机器学习02【机器学习】

# 主要内容

1. 朴素贝叶斯算法
2. 拉普拉斯估计
3. KNN最近邻算法
4. Kmeans聚类算法

# 学习目标

|  |  |  |
| --- | --- | --- |
| 节数 | 知识点 | 要求 |
| 第一节 （朴素贝叶斯算法） | 贝叶斯算法公式 | 掌握 |
| 贝叶斯算法分类原理 | 掌握 |
| 拉普拉斯估计思想 | 掌握 |
| 第二节（垃圾邮件分类案例） | 垃圾邮件分类案例 | 掌握 |
| Scala贝叶斯案例 | 掌握 |
| 第三节（KNN最近邻算法） | KNN最近邻算法分类原理 | 掌握 |
| KNN三要素 | 掌握 |
| KNN分类步骤 | 掌握 |
| KNN存在的问题 | 掌握 |
| 距离度量 | 掌握 |
| 第四节（KNN案例） | KNN相亲案例 | 掌握 |
| KNN文本分类案例 | 掌握 |
| 第五节（K-Means聚类算法） | K-Means聚类算法原理 | 掌握 |
| K-Means算法问题 | 掌握 |
| K-Means ++算法 | 掌握 |

# 朴素贝叶斯算法

朴素贝叶斯（Naive Bayes ,NB）算法是基于贝叶斯定理与特征条件独立假设的分类方法，该算法是有监督的学习算法，解决的是分类问题，是将一个未知样本分到几个预先已知类别的过程。

朴素贝叶斯的思想就是根据某些个先验概率计算Y变量属于某个类别的后验概率，也就是根据先前事件的有关数据估计未来某个事件发生的概率。

1. **举例：**

一个学校内有60%的学生是男生，40%的学生是女生。根据统计，男生总是穿长裤，女生则有一半穿长裤，一半穿裙子。

问题：

假设在校园中随机抽取一个穿长裤的学生，推断该学生是女生的概率？

已知：

P(男生) = 60%

P(女生) = 40%

P(长裤|女生) = 50%

P(裙子|女生) = 50%

求： P(女生|长裤)—穿长裤人数中是女生的概率？

要知道穿长裤的人是女生的概率，要知道穿长裤女生的人数，也要知道穿长裤的总人数，两者相除就是长裤中女生的概率。假设学校人数为U

穿长裤总人数 = 穿长裤的男生人数+穿长裤的女生人数

= 60% \* U + 40% \* U \*50%

穿长裤女生的人数 = 40% \* U \* 50%

随机抽取一个穿长裤的学生是女生的概率

= 穿长裤女生的人数/穿长裤总人数

= 40% \* U \* 50% / ( 60% \* U + 40% \* U \*50%)

= 0.25

假设学生穿长裤记作事件A，学生穿长裤的概率就是P(A)。学生是女生记作事件B，学生是女生的概率是P(B)，求:抽取的这个穿长裤学生是女生的概率（P(B|A)）？



=  =0.25

1. **朴素贝叶斯公式**



也可以简写为：

其中:

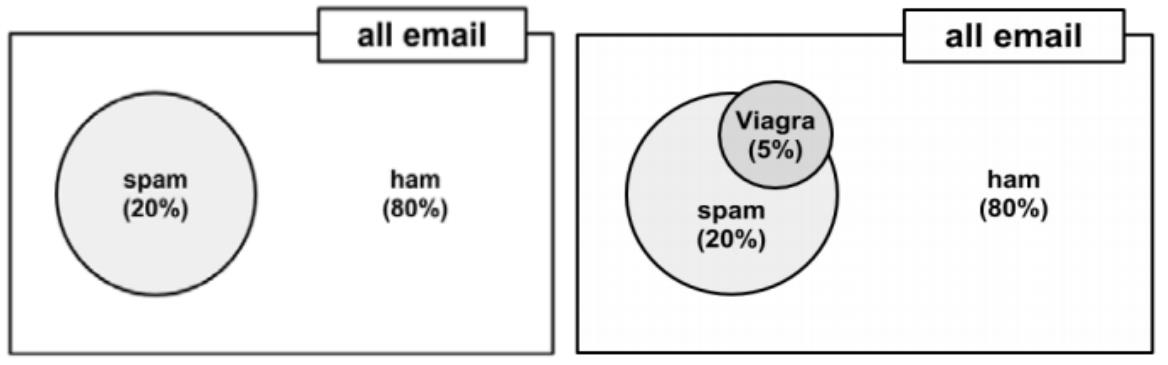
P(A)叫做A事件的先验概率，即一般情况下，认为A发生的概率。

P(B|A)叫做似然度，是A假设条件成立的情况下发生B的概率。

P(A|B)叫做后验概率，在B发生的情况下发生A的概率，也就是要求的概率。P(B)叫做标准化常量，即在一般情况下，认为B发生的概率。

1. **理解朴素贝叶斯**

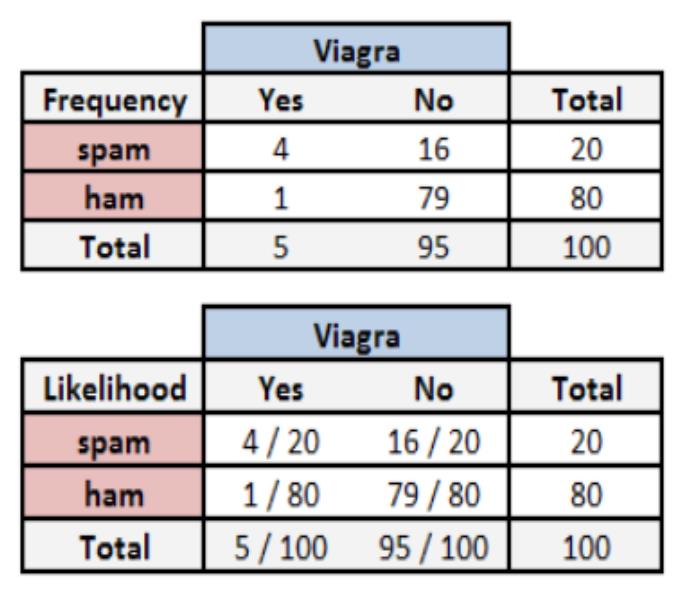
假设现在有一堆邮件，正常邮件的比例是80%，垃圾邮件的比例是20%，这堆邮件中，5%的邮件中出现Viagra单词，如果有一封邮件，这封邮件中包含Viagra单词，求这封邮件是垃圾邮件的概率。



显然不能使用5%\*20%=1%得到这封邮件是垃圾邮件的概率，因为垃圾邮件中有可能出现Viagra也有可能不会出现Viagra单词。那么根据贝叶斯公式可得包含Viagra单词的邮件是垃圾邮件的概率为：



是已知20%，也是已知 5%，那么如果求出的概率，结果就可以知道。我们可以根据邮件的数据集统计得到单词频率表：

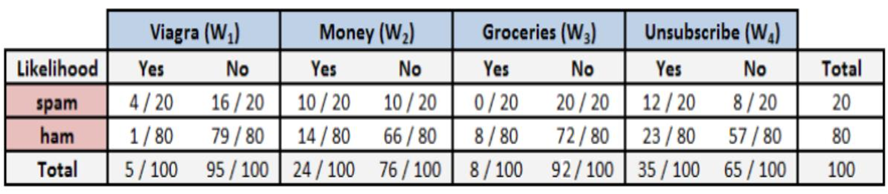


其中表示在垃圾邮件中出现Viagra单词的概率，通过统计得出为4/20。可以得出如果一封邮件中有Viagra单词，这封邮件是垃圾邮件的概率为：



  0.8

通过同样的计算可以得到，含有Viagra单词但不是垃圾邮件的概率为0.2。那么可以认为这封邮件是垃圾邮件的概率比较大。这里的Viagra可以理解为邮件中的一个特征。那么当一封邮件有额外更多的特征时，贝叶斯如何扩展？

假设所有历史邮件中只出现了4个单词，也就是4个特征，根据历史邮件统计的单词频率表如下：假设现在给定一封邮件中有Viagra和Unsubscribe(取消订阅)两个单词，求这封邮件是垃圾邮件的概率、不是垃圾邮件的概率？

利用贝叶斯公式，我们可以得到：

是垃圾邮件的概率：

=

=

=  98.07% 

不是垃圾邮件的概率：

= 

= 



17.4%

1. **拉普拉斯估计**

根据以上例子，假设有一封邮件这4个单词都出现了，求这封邮件是垃圾邮件的概率：



=

由于的概率为0/20，会导致整个结果是垃圾邮件的概率为0，那么就否定了其他单词出现的权重。

拉普拉斯估计本质上是给频率表中的每个单词的计数加上一个较小的数，这样就保证每一类中每个特征发生的概率非零。通常，拉普拉斯估计中加上的数值为1，这样就保证了每一个特征至少在数据中出现一次。

以上例子如果四个单词都出现情况下计算是否是垃圾邮件，出现的概率为0/20，可以增加4封垃圾邮件，使4封邮件中每个邮件中只有一个单词出现，这样就避免了垃圾邮件中有的单词出现概率为0的情况。同样在不是垃圾邮件中也增加4封，避免在正常邮件中出现有的单词出现概率为0的情况。

# 垃圾邮件分类案例

1. **Python 贝叶斯案例**

依次需要使用pip命令安装numpy、scipy、skikit\_learn、matplotlib模块。

python代码：

|  |
| --- |
| import os  import sysecs 编码转换模块  import codecs  from sklearn.naive\_bayes import MultinomialNB  from sklearn.feature\_extraction.text import CountVectorizer  if \_\_name\_\_ == *'\_\_main\_\_'*:  # 读取文本构建语料库  corpus = []  labels = []  corpus\_test = []  labels\_test = []  f = codecs.open(*"i:/ML/sms\_spam.txt"*, *"rb"*)  count = 0  while True:  line = f.readline()  if count == 0:  count = count + 1  continue  if line:  count = count + 1  line = line.split(*","*)  label = line[0]  sentence = line[1]  corpus.append(sentence)  if *"ham"*==label:  labels.append(0)  elif *"spam"*==label:  labels.append(1)  if count > 5550:  corpus\_test.append(sentence)  if *"ham"*==label:  labels\_test.append(0)  elif *"spam"*==label:  labels\_test.append(1)  else:  break  #CountVectorizer是将文本向量转换成稀疏表示数值向量（字符频率向量） vectorizer 将文档词块化  vectorizer=CountVectorizer()  fea\_train = vectorizer.fit\_transform(corpus)  print vectorizer.get\_feature\_names()  print fea\_train.toarray()  vectorizer2=CountVectorizer(vocabulary=vectorizer.vocabulary\_)  fea\_test = vectorizer2.fit\_transform(corpus\_test)  rint vectorizer2.get\_feature\_names()  print fea\_test.toarray()  #create the Multinomial Naive Bayesian Classifier  #alpha = 1 拉普拉斯估计给每个单词加1  clf = MultinomialNB(alpha = 1)  clf.fit(fea\_train,labels)  #  pred = clf.predict(fea\_test);  for p in pred:  if p == 0:  print *"正常邮件"*  else:  print *"垃圾邮件"* |

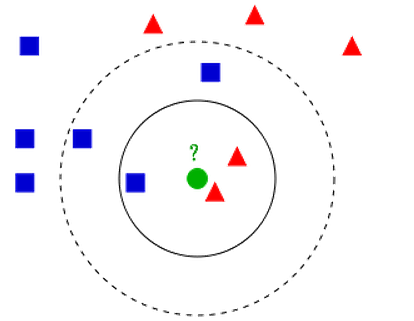
1. **Scala 贝叶斯案例**

|  |
| --- |
| **object** Naive\_bayes {  **def** main(args: Array[*String*]) {  //1 构建Spark对象  **val** conf = **new** SparkConf().setAppName("Naive\_bayes").setMaster("local")  **val** sc = **new** SparkContext(conf)  //读取样本数据1  **val** data = sc.textFile("sample\_naive\_bayes\_data.txt")  **val** parsedData = data.map { line =>  **val** parts = line.split(',')  **LabeledPoint**(parts(0).toDouble, Vectors.dense(parts(1).split(' 、').map(\_.toDouble)))  }  //样本数据划分训练样本与测试样本  **val** splits = parsedData.randomSplit(Array(0.9, 0.1), seed = 11L)  **val** training = splits(0)  **val** test = splits(1)  //新建贝叶斯分类模型模型，并训练 ,lambda 拉普拉斯估计  **val** model = NaiveBayes.train(training, lambda = 1.0)  //对测试样本进行测试  **val** predictionAndLabel = test.map(p => (model.predict(p.features), p.label))  **val** print\_predict = predictionAndLabel.take(10020)  println("prediction" + "\t" + "label")  **for** (i <- 0 to print\_predict.length - 1) {  println(print\_predict(i).\_1 + "\t" + print\_predict(i).\_2)  }    **val** accuracy = 1.0 \* predictionAndLabel  .filter(x => x.\_1 == x.\_2).count() / test.count()  println(accuracy)  //保存模型  **val** ModelPath = "naive\_bayes\_model"  model.save(sc, ModelPath)  **val** sameModel = NaiveBayesModel.load(sc, ModelPath)  }  } |

# KNN最近邻算法

1. **KNN最邻近算法**

K最近邻(k-Nearest Neighbor，KNN)分类算法，是一个理论上比较成熟的方法，也是最简单的机器学习算法之一，有监督算法。该方法的思路是：如果一个样本在特征空间中的k个最相似的样本中的大多数属于某一个类别，则该样本也属于这个类别。KNN算法由你的邻居来推断出你的类别，KNN算法就是用距离来衡量样本之间的相似度。



如果K = 3，绿色圆点的最近的3个邻居是2个红色小三角形和1个蓝色小正方形，少数从属于多数，基于统计的方法，判定绿色的这个待分类点属于红色的三角形一类。

如果K = 5，绿色圆点的最近的5个邻居是2个红色三角形和3个蓝色的正方形，还是少数从属于多数，基于统计的方法，判定绿色的这个待分类点属于蓝色的正方形一类。

*K 值的选择，距离度量和分类决策规则是该算法的三个基本要素。*K值的选择一般低于样本数据的平方根,一般是不大于20的整数。距离度量常用的有欧式距离，曼哈顿距离，余弦距离等，一般使用欧氏距离，对于文本分类，常用余弦距离。分类决策就是“少数服从多数”的策略。

1. **KNN算法步骤：**
2. 对于未知类别的数据（对象，点），计算已知类别数据集中的点到该点的距离。
3. 按照距离由小到大排序
4. 选取与当前点距离最小的K个点
5. 确定前K个点所在类别出现的概率
6. 返回当前K个点出现频率最高的类别作为当前点预测分类
7. **KNN算法复杂度：**

KNN 分类的计算复杂度和训练集中的文档数目成正比，也就是说，如果训练集中文档总数为 n，那么 KNN 的分类时间复杂度为O(n)

1. **KNN问题：**

该算法在分类时有个主要的不足是，当样本不平衡时，如一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的 K 个邻居中大容量类的样本占多数。解决：可以采用权值的方法，根据和该样本距离的远近，对近邻进行加权，距离越小的邻居权值大，权重一般为距离平方的倒数。

1. **KNN数据归一化：**

为了防止某一维度的数据的数值大小对距离计算产生影响，保证多个维度的特征是等权重的，最终结果不能被数值的大小影响，应该将各个维度进行数据的归一化，把数据归一化到[0,1]区间上。

归一化公式：

1. **距离度量**

**欧式距离**：也称欧几里得距离，在一个N维度的空间里，求两个点的距离，这个距离肯定是一个大于等于零的数字，那么这个距离需要用两个点在各自维度上的坐标相减，平方后加和再开方。一维，二维，三维的欧式距离计算方法：一维： 二维： 三维：

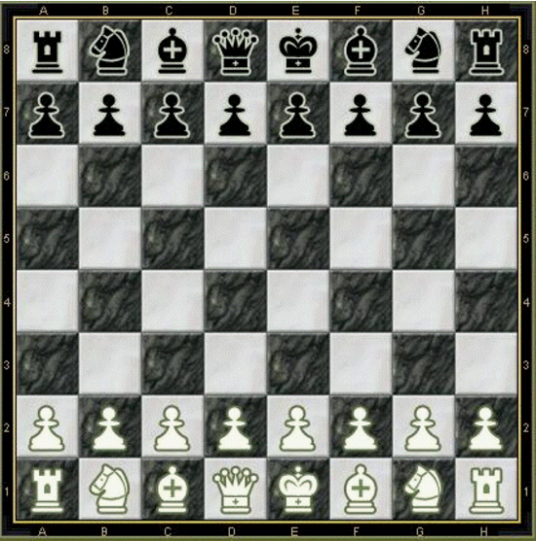
**平方欧式距离：**就是欧式距离的平方。

**曼哈顿距离**：相比欧式距离简单的多，曼哈顿距离只要把两个点坐标的x坐标相减取绝对值，y坐标相减取绝对值，再加和， 。三维，四维以此类推。

**余弦距离**：也叫余弦相似度，是用向量空间中两个向量夹角的余弦值作为衡量两个个体间差异的大小的度量。如果两个向量的方向一致，即夹角接近零，那么这两个向量就越相近。要确定两个向量方向是否一致，要用到余弦定理计算向量的夹角。

**闵可夫斯基距离**：闵式距离不是一种距离，而是一组距离的定义，是对多个距离度量公式的概括性表述。定义：两个n维变量（可以理解为n维数组，就是有n个元素）a()与b()间的闵可夫斯基距离定义为：**** 其中p是一个变参数，当p=1时，就是曼哈顿距离，当p=2时，就是欧式距离，当p 就是切比雪夫距离。

**切比雪夫距离**：国际象棋中，国王可以直行、横行、斜行。国王走一步，可以移动到相邻的8个方格的任意一个。国王从格子 到格子最少需要多少步？这个距离就是切比雪夫距离。



切比雪夫距离公式简单理解为就是各坐标数值差的最大值，在2维空间中的计算公式为： 。

**谷本距离：**同时考虑余弦距离和欧式距离的测度。

**加权距离测度**：可以指定某一维度的权重比例，从而使某个权重的影响力更大。

# KNN案例

1. **相亲案例**

代码实现：

|  |
| --- |
| import numpy as np  import operator  #matplotata 测试数据集的某行， dataSet 训练数据集 ，labels 训练数据集的类别，k k的值  def **classify**(normData,dataSet,labels,k):  dataSetSize = dataSet.shape[0]  # print 'dataSetSize 长度 =',dataSetSize  #当前点到所有点的坐标差值  diffMat = np.tile(normData, (dataSetSize,1)) - dataSet  #对每个坐标差值平方  sqDiffMat = diffMat \*\* 2  #对于二维数组 sqDiffMat.sum(axis=0)指定对数组b对每列求和，sqDiffMat.sum(axis=1)是对每行求和  sqDistances = sqDiffMat.sum(axis = 1)  #欧式距离 最后开方  distance = sqDistances \*\* 0.5  #argsort() 将x中的元素从小到大排序，提取其对应的index 索引，返回数组  sortedDistIndicies = distance.argsort()  # classCount保存的K是魅力类型 V:在K个近邻中某一个类型的次数  classCount = {}  for i in range(k):  #获取对应的下标的类别  voteLabel = labels[sortedDistIndicies[i]]  #给相同的类别次数计数  classCount[voteLabel] = classCount.get(voteLabel,0) + 1  sortedClassCount = sorted(classCount.iteritems(),key=operator.itemgetter(1),reverse=True)  return sortedClassCount[0][0]  def **file2matrix**(filename):  fr = open(filename)  # readlines:是一次性将这个文本的内容全部加载到内存中(列表)  arrayOflines = fr.readlines()  numOfLines = len(arrayOflines)  # print "numOfLines = " , numOfLines  #numpy.zeros 创建给定类型的矩阵 numOfLines 行 ，3列  returnMat = np.zeros((numOfLines,3))  classLabelVector = []  index = 0  for line in arrayOflines:  #去掉一行的头尾空格  line = line.strip()  listFromline = line.split(*'\t'*)  returnMat[index,:] = listFromline[0:3]  classLabelVector.append(int(listFromline[-1]))  index += 1  return returnMat,classLabelVector  *'''*  *将训练集中的数据进行归一化*  *归一化的目的：*  *训练集中飞行公里数这一维度中的值是非常大，那么这个纬度值对于最终的计算结果(两点的距离)影响是非常大，*  *远远超过其他的两个维度对于最终结果的影响*  *实际约会姑娘认为这三个特征是同等重要的*  *下面使用最大最小值归一化的方式将训练集中的数据进行归一化*  *'''*  #将数据归一化  def **autoNorm**(dataSet):  # dataSet.min(0) 代表的是统计这个矩阵中每一列的最小值 返回值是一个矩阵1\*3矩阵  minVals = dataSet.min(0)  maxVals = dataSet.max(0)  ranges = maxVals - minVals  #dataSet.shape[0] 计算行数， shape[1] 计算列数  m = dataSet.shape[0]1列  normDataSet = dataSet - np.tile(minVals,(m,1))  normDataSet = normDataSet / np.tile(ranges,(m,1))  return normDataSet,ranges,minVals  def **datingClassTest**():  hoRatio = 0.1  datingDataMat,datingLabels = file2matrix(*'datingTestSet2.txt'*)  #将数据归一化  normMat,ranges,minVals = autoNorm(datingDataMat)  # m 为行数 = 1000  m = normMat.shape[0]  # print 'm =%d 行'%m  #取出100行数据测试  numTestVecs = int(m\*hoRatio)  errorCount = 0.0  for i in range(numTestVecs):  #normMat[i,:] 取出数据的第i行,normMat[numTestVecs:m,:]取出数据中的100行到1000行 作为训练集，datingLabels[numTestVecs:m] 取出数据中100行到1000行的类别，4是K  classifierResult = classify(normMat[i,:],normMat[numTestVecs:m,:],datingLabels[numTestVecs:m],4)  print *'模型预测值: %d ,真实值 : %d'* %(classifierResult,datingLabels[i])  if (classifierResult != datingLabels[i]):  errorCount += 1.0  errorRate = errorCount / float(numTestVecs)  print *'正确率 : %f'* %(1-errorRate)  return 1-errorRate  def **classifyperson**():  resultList = [*'没感觉'*, *'看起来还行'*,*'极具魅力'*]  input\_man= [30000,3,0.1]  datingDataMat,datingLabels = file2matrix(*'datingTestSet2.txt'*)  normMat,ranges,minVals = autoNorm(datingDataMat)  result = classify((input\_man - minVals)/ranges,normMat,datingLabels,3)  print *'你即将约会的人是:'* , resultList[result-1]  if \_\_name\_\_ == *'\_\_main\_\_'*:  acc = datingClassTest()  if(acc > 0.9):  classifyperson() |

调用python 中 Scikit-learn 实现KNN算法：

|  |
| --- |
| from sklearn.neighbors import NearestNeighbors  import numpy as np  from KNNDateOnHand import \*  datingDataMat,datingLabels = file2matrix(*'datingTestSet2.txt'*)  normMat,ranges,minVals = autoNorm(datingDataMat)  t作为训练集拟合模型  nbrs = NearestNeighbors(n\_neighbors=3).fit(normMat)  input\_man= [30000,3,2]  #数据归一化  S = (input\_man - minVals)/ranges  #找到当前点的K个临近点，也就是找到临近的3个点  distances, indices = nbrs.kneighbors(S)  print indices  print distances  # classCount K：类别名 V：这个类别中的样本出现的次数  classCount = {}  for i in range(3):应的索引的类别号  voteLabel = datingLabels[indices[0][i]]  classCount[voteLabel] = classCount.get(voteLabel,0) + 1  sortedClassCount = sorted(classCount.iteritems(),key=operator.itemgetter(1),reverse=True)  resultList = [*'没感觉'*, *'看起来还行'*,*'极具魅力'*]  print resultList[sortedClassCount[0][0]-1] |

1. **KNN文本分类案例**

|  |
| --- |
| import os  import numpy as np  from KNNDateOnHand import classify  方法将每个文件中32\*32的矩阵数据，转换到1\*1024一行中  def **img2vector**(filename):  #创建一个1行1024列的矩阵  returnVect = np.zeros((1,1024))  #打开当前的文件  fr = open(filename)  #每个文件中有32行，每行有32列数据，遍历32个行，将32个列数据放入1024的列中  for i in range(32):  lineStr = fr.readline()  for j in range(32):  returnVect[0,32\*i+j] = int(lineStr[j])  return returnVect  def **IdentifImgClassTest**():  hwLabels = []  #读取训练街 TrainData目录下所有的文件和文件夹  trainingFileList = os.listdir(*'TrainData'*)  m = len(trainingFileList)  #zeros((m,1024)) 返回一个m行 ，1024列的矩阵，默认是浮点型的  trainingMat = np.zeros((m,1024))  for i in range(m):  #获取文件名称  fileNameStr = trainingFileList[i]  #获取文件除了后缀的名称  fileStr = fileNameStr.split(*'.'*)[0]  #获取文件"数字"的类别  classNumStr = int(fileStr.split(*'\_'*)[0])  hwLabels.append(classNumStr)  #构建训练集  trainingMat[i,:] = img2vector(*'TrainData/%s'* % fileNameStr)  #读取测试集数据  testFileList = os.listdir(*'TestData'*)  errorCount = 0.0  mTest = len(testFileList)  for i in range(mTest):  fileNameStr = testFileList[i]  fileStr = fileNameStr.split(*'.'*)[0]  classNumStr = int(fileStr.split(*'\_'*)[0])  vectorUnderTest = img2vector(*'TestData/%s'* % fileNameStr)  classifierResult = classify(vectorUnderTest, trainingMat, hwLabels, 3)  print *"识别出的数字是: %d, 真实数字是: %d"* % (classifierResult, classNumStr)  if (classifierResult != classNumStr):  errorCount += 1.0  print *"\n识别错误次数 %d"* % errorCount  errorRate = errorCount/float(mTest)  print *"\n正确率: %f"* % (1-errorRate)  if \_\_name\_\_ == *'\_\_main\_\_'*:  IdentifImgClassTest() |

# K-Means聚类算法

1. **K-means聚类算法**

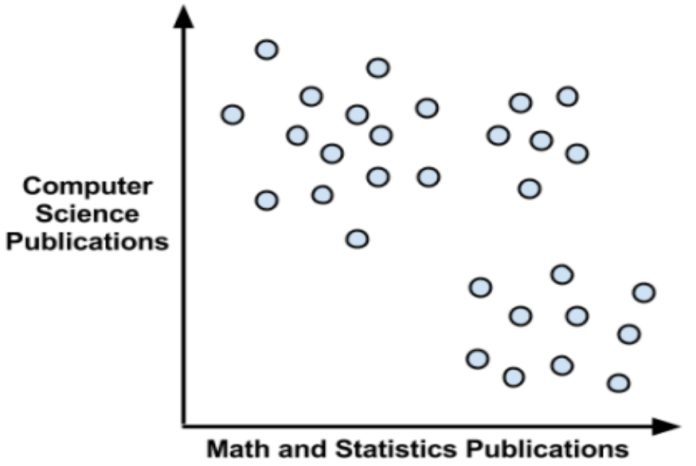
机器学习中有两类的大问题，一个是分类，一个是聚类。分类是监督学习，原始数据有标签，可以根据原始数据建立模型，确定新来的数据属于哪一类。聚类是一种无监督学习，聚类是指事先没有“标签”，在数据中发现数据对象之间的关系，将数据进行分组，一个分组也叫做“一个簇”， 组内的相似性越大，组间的差别越大，则聚类效果越好，也就是簇内对象有较高的相似度，簇之间的对象相似度比较低，则聚类效果越好。K-means就是一个聚类算法。

K-means聚类算法中K表示将数据聚类成K个簇，means表示每个聚类中数据的均值作为该簇的中心，也称为质心。K-means聚类试图将相似的对象归为同一个簇，将不相似的对象归为不同簇，这里需要一种对数据衡量相似度的计算方法，K-means算法是典型的基于距离的聚类算法，采用距离作为相似度的评价指标，默认以欧式距离作为相似度测度，即两个对象的距离越近，其相似度就越大。

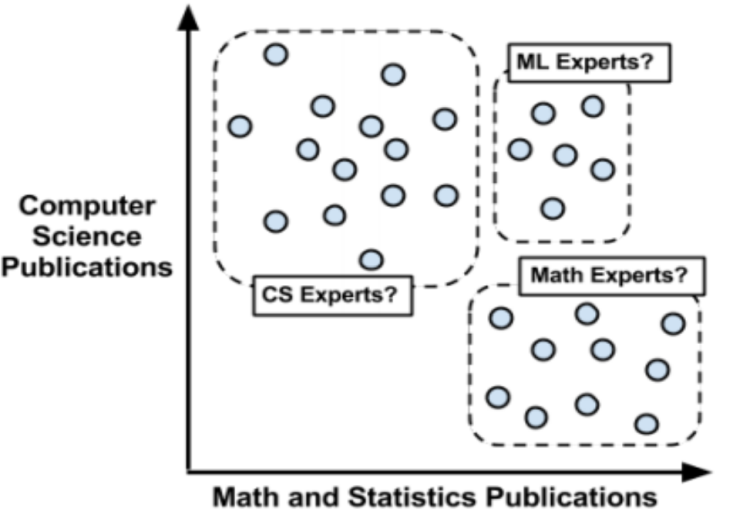
聚类和分类最大的不同在于，分类的目标是事先已知的，而聚类则不一样，聚类事先不知道目标变量是什么，类别没有像分类那样被预先定义出来，也就是聚类分组不需要提前被告知所划分的组应该是什么样的，因为我们甚至可能都不知道我们再寻找什么，所以聚类是用于知识发现而不是预测，所以，聚类有时也叫无监督学习。

1. **K-means过程原理**

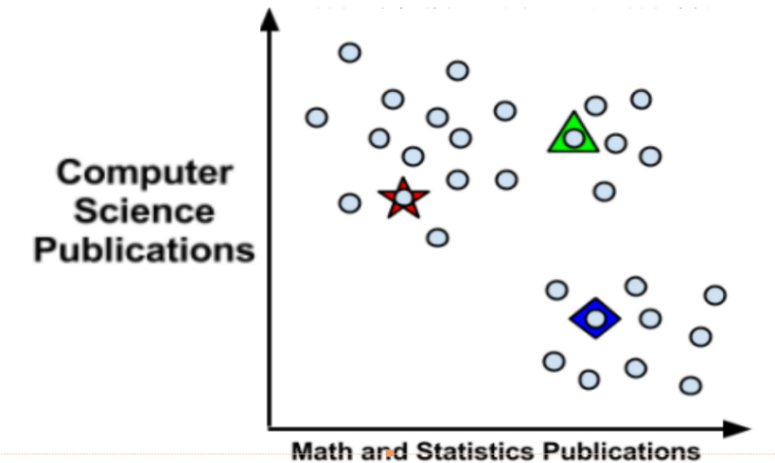
假设有一批关于计算机科学和数学统计相关的人才，这批人才中计算机人才、机器学习人才、数学人才三类，那么该如何将这批数据进行聚类？



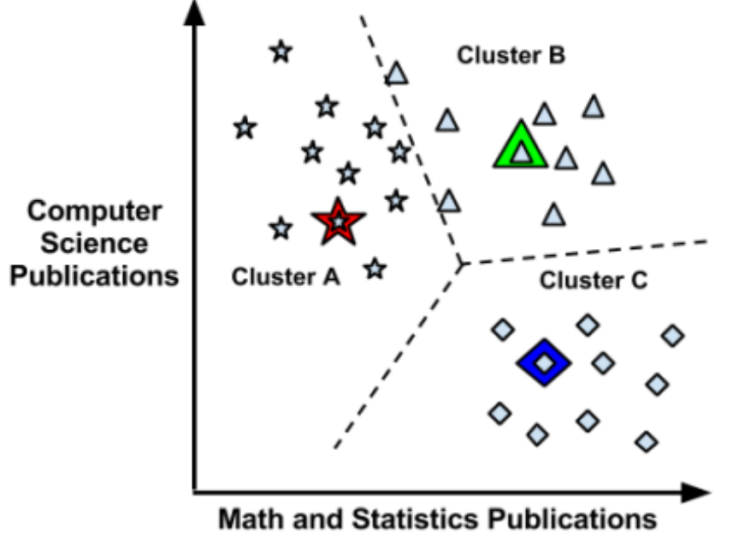
我们可以直观的感觉到应该如下分类：

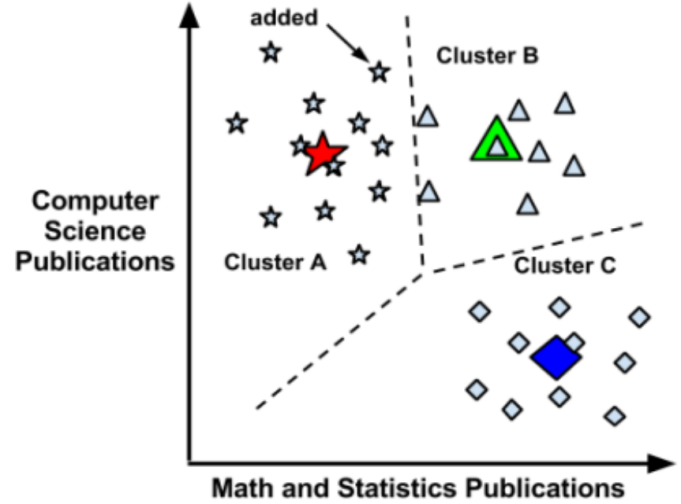


但问题是计算机不会直观的去观察数据，首先将这批数据向量化，K-means聚类会随机在这些点中找到三个点，然后计算所有的样本到当前三个点的距离大小，判断样本点与当前三个点哪个距离比较近，当前样本就属于那个类。

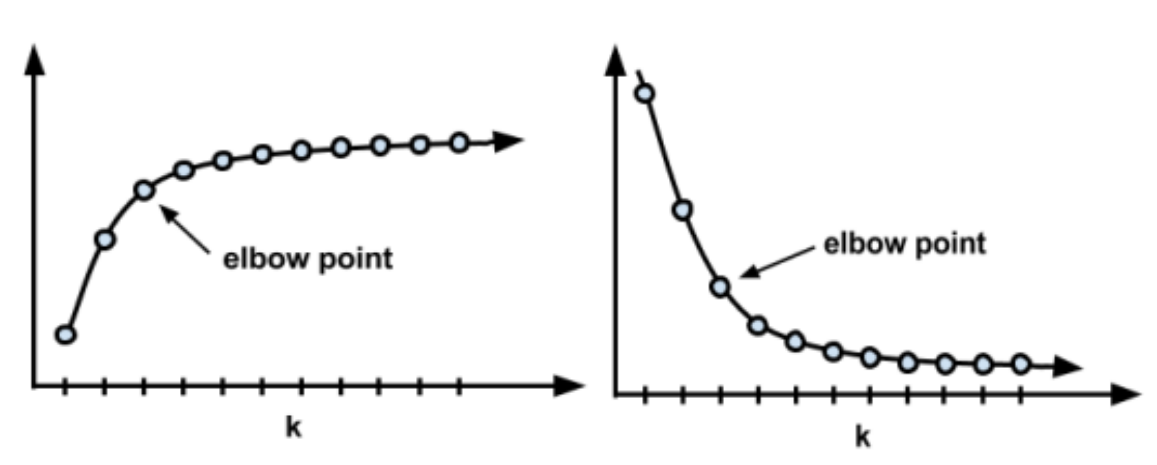


当经过一次计算之后，就是经过了一次迭代过程，当完成一次迭代后，就分出来是三个类别（簇），每个类别中都有质心。然后进行下一次迭代，继续计算所有点到三个类别中心点的距离，按照每个样本点与哪个质心距离最近就属于哪个簇，以此类推，继续迭代。当本次计算的中心点的距离较上次计算的中心点的位置不再变化，那么停止迭代。





K值的选择一般可以根据问题的内容来确定，也可以根据肘部法来确定。如图：横轴表示K值的选择，纵轴表示对应的K值下所有聚类的平均畸变程度。



每个类的畸变程度是每个类别下每个样本到质心的位置距离的平方和。类内部成员越是紧凑，那么类的畸变程度越低，这个类内部相似性越大，聚类也就越好。如图，在k=1请况下，相比k=2情况下，类的平均畸变程度变化大，说明，k=2的情况类的紧凑程度比k=1情况下要紧凑的多。同理，发现当k=3之后，随着k的增大，类的平均畸变程度变化不大，说明k=3是比较好的k值。k>3后类的平均畸变程度变化不大，聚类的个数越多，有可能类与类之间的相似度越大，类的内部反而没有相似度，这种聚类也是不好的。举个极端的例子，有1000个数据，分成1000个类，那么类的平均畸变程度是0，那么每个数据都是一类，类与类之间的相似度大，类内部没有相似性。

K-means算法的思想就是对空间K个点为中心进行聚类，对靠近他们的对象进行归类，通过迭代的方法，逐次更新聚类中心(质心)的值，直到得到最好的聚类结果。K-means过程：

1. 首先选择k个类别的中心点
2. 对任意一个样本，求其到各类中心的距离，将该样本归到距离最短的中心所在的类
3. 聚好类后，重新计算每个聚类的中心点位置
4. 重复2,3步骤迭代，直到k个类中心点的位置不变，或者达到一定的迭代次数，则迭代结束，否则继续迭代
5. **K-means++算法**

K-means算法假设聚类为3类，开始选取每个类的中心点的时候是随机选取，有可能三个点选取的位置非常近，导致后面每次聚类重新求各类中心的迭代次数增加。K-means++在选取第一个聚类中心点的时候也是随机选取，当选取第二个中心点的时候，距离当前已经选择的聚类中心点的距离越远的点会有更高的概率被选中，假设已经选取n个点，当选取第n+1个聚类中心时，距离当前n个聚类中心点越远的点越会被选中，这种思想是聚类中心的点离的越远越好，这样就大大降低的找到最终聚类各个中心点的迭代次数，提高了效率。

# 本节作业

1. 理解贝叶斯算法。
2. 理解KNN算法。
3. 理解Kmeans算法。