

**《数据挖掘》报告**

课题名称：kMeans与kNN的实现与比较

院系： 电子信息与通信学院

专业： 电子信息工程

班级： 电信1204班

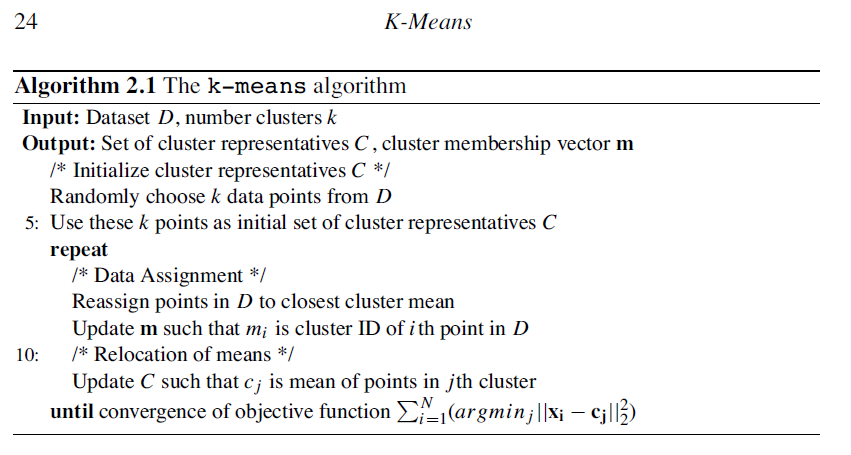
姓名： 黄衎

学号： U201213468

# kMeans

## 原理

kMeans算法流程如下：



kMeans首先随机选取k个点作为聚类中心，余下的点进行择近选择，然后以聚类的均值作为新的聚类中心，作为初始迭代。此后的迭代则对每个点，对k个样本中心点（聚类中心）择近选择，再计算样本中心点。直到样本中心点不再变化，算法结束。

注意在实际计算时，由于样本坐标采用浮点表示，样本中心点不再变化的判断方法是判断新的样本中心与旧的样本中心相差是否小于浮点数的精度，在Matlab中表达如下：

if abs(sum(kmeansNew - kmeans)) <eps(sum(kmeansNew))

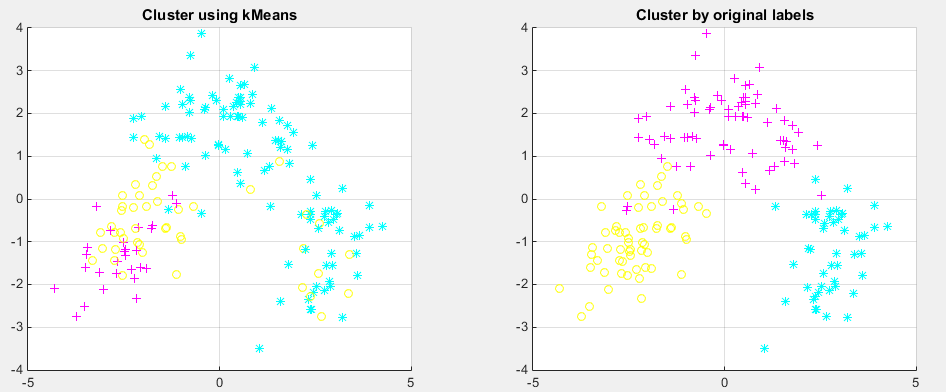
kMeans算法由于其初始化方式，带有随机性，聚类具有随机性。

## 原始结果

以下是处理UCI的Wine数据的两次结果：

注意wine数据是13维的，在数据可视化之前做了PCA降维处理。





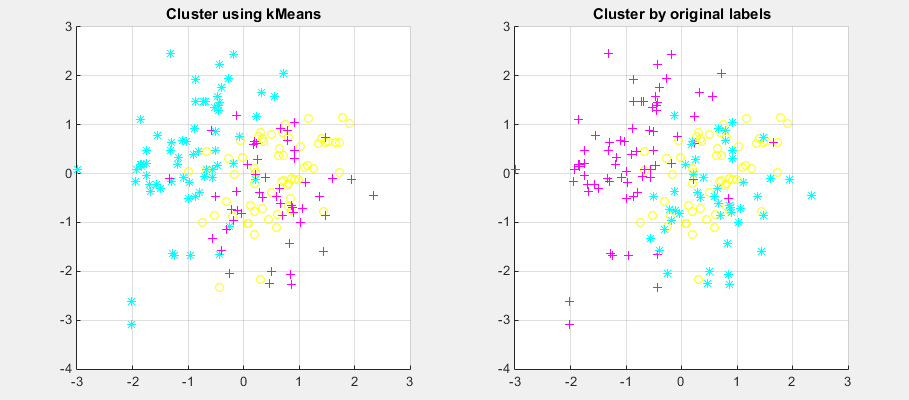
由于数据量小，聚类结果差异不明显。但是比较明显的是，由于kMeans是非监督学习，它对各个聚类的编号（Label）是随机的，如果不手动干预初始化的方式，则聚类编号极大可能与设想的编号不一致。这样，对聚类结果的解释则需要人工进行。

Wine数据中各个维度的数据范围、方差都不同，以上结果是直接对原始数据进行kMeans得到，在实际的机器学期和数据挖掘实践中，进行诸如PCA降维、数据标准化等数据预处理有时候是很有必要的，因此，对Wine数据进行处理后再进行试验。

## 数据标准化

数据标准化的意义在于，把不同范围各个属性归一化到同一个范围，使得各个属性在计算样本间距的时候具有相同的重要性（原始数据由于属性的范围不同，大范围的数据影响程度明显要高），也方便我们定义各个属性的加权重要程度，对不同的feature采取不同的权重，改进聚类结果。

下图是对Wine数据进行数据标准化并选取5个主成分的聚类结果：

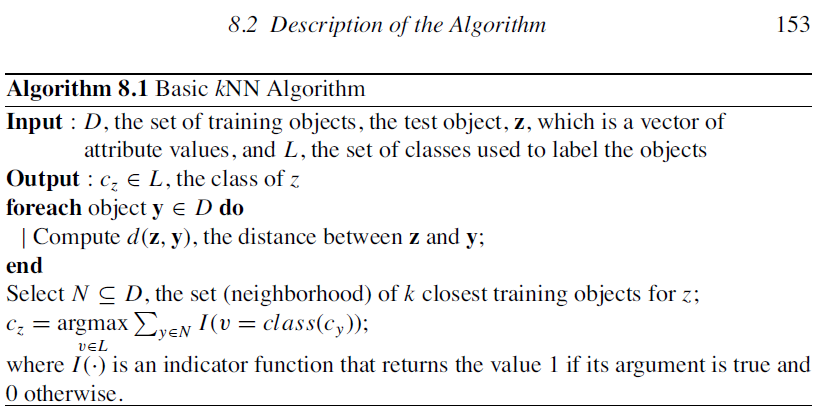


由此可见，虽然聚类的标签编号仍然随机，但是聚类结果有了明显的改善，与原始数据聚类Label更加一致。同时可以看出，Wine数据的主维度在3维以上，因为2维空间（绘图平面）不足以把各个聚类分开。

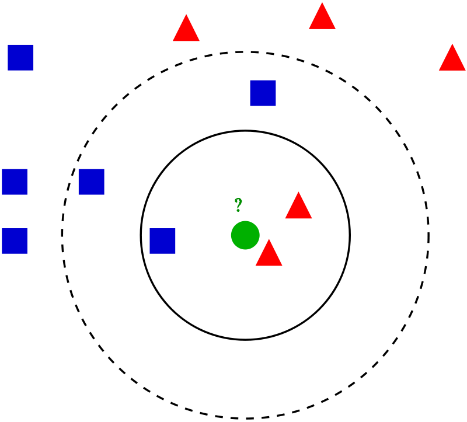
# kNN

## 原理

同为聚类算法，kNN实质上是有监督的学习，必须将数据集分为两部分——Train（训练集）和Test（测试集），用Train来监督Test的聚类。kNN的算法流程如下：



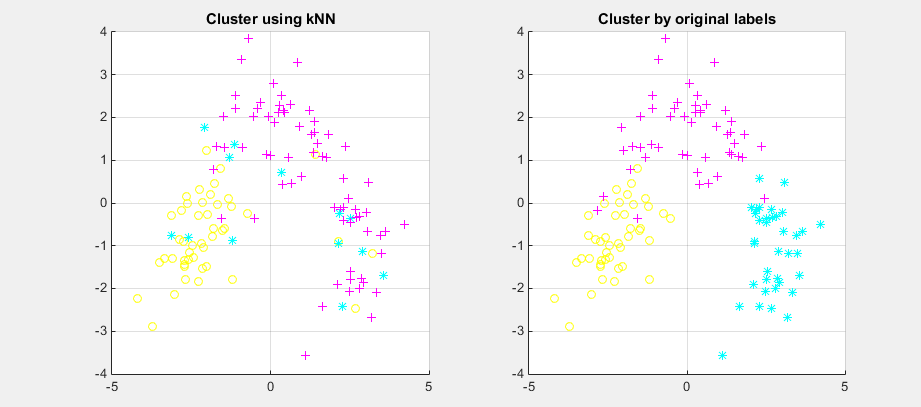
kNN的实质是，根据给定Label的Train的数据，对Test中的每个数据，寻找最近的kN（kN，最近邻数，一般大于10且取奇数，和聚类数k不是一个变量）个Train邻居。在这kN个最近邻居中，哪个聚类的邻居最多，该数据就属于哪一类。kNN不需要迭代，一次计算即可完成。

摘自维基百科[4]的一张解释kNN算法的经典图例。显然，kN的设置对kNN的聚类效果影响很大。

一般来说，kN越小，聚类边界越呈现齿状分布，kN越大，聚类边界越平滑[5]。

## 原始结果

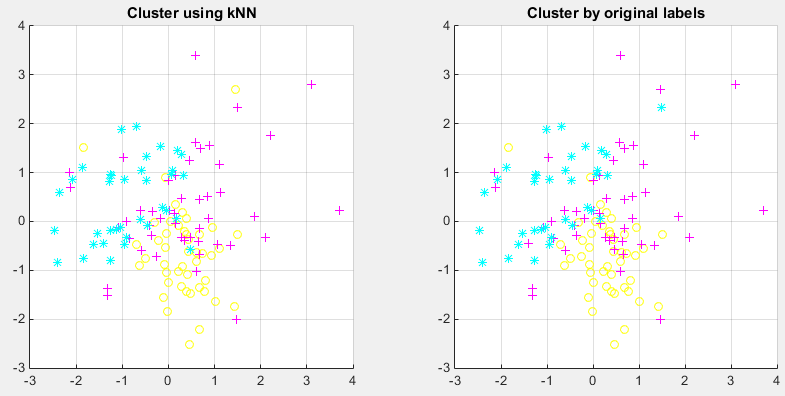
同样对Wine数据进行处理，它共有178个样本，初始化kNN聚类器时随机选取其中50个样本作为Train（保证每个Label都有）。



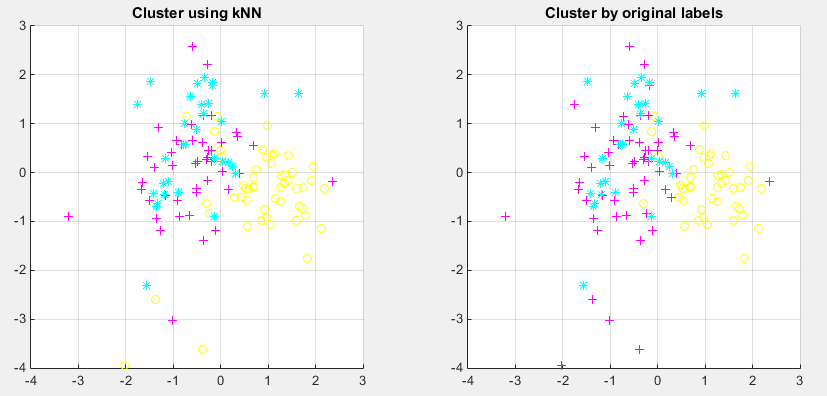
和kMeans一样，不进行数据预处理，结果很不理想，但是和kMeans比，同样不进行预处理，效果却比kMeans好一些。

## 数据标准化和降维

同kMeans，把Wine数据降到5维，同时取kN=9，结果如下：

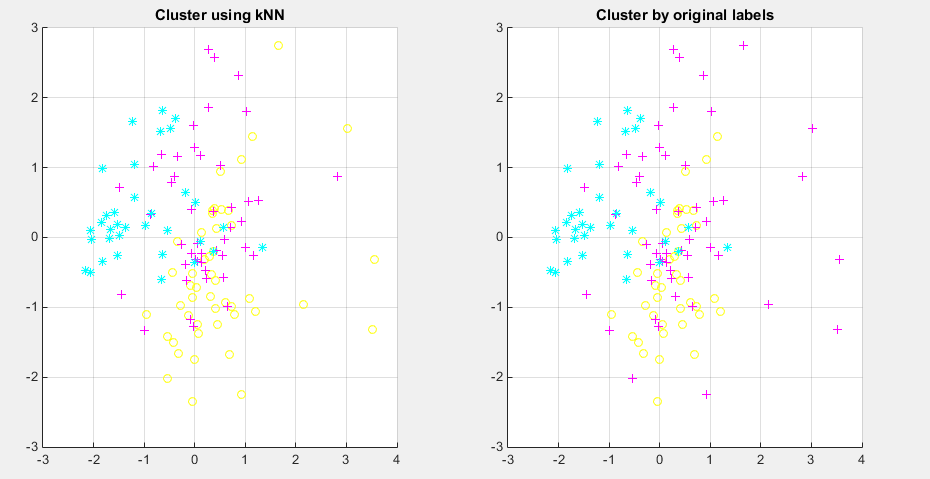


在调试过程中发现有趣的现象，kN=10的结果：

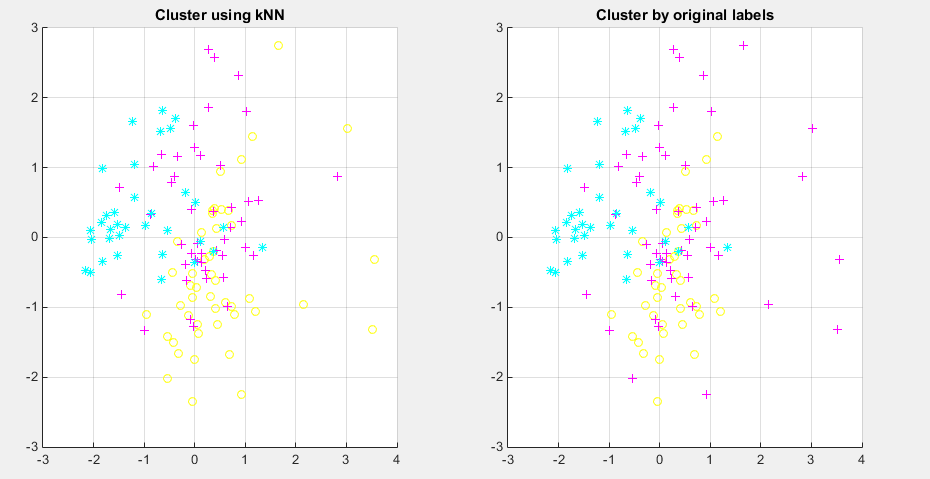


明显没有kN=9好，错误点数比较明显。

对比kN=19：



kN=20：



总之kN取奇数可能比较好，因为能以较大概率避免“投票数”相等的情况，不过实验结果是在取不同的Train集合得到的，只能参考，有待深入研究。

# 聚类算法比较——kMeans与kNN

严格意义上讲，kNN属于分类算法，kMeans属于聚类算法，不过这两个概念有些时候界限比较模糊，因为用聚类的方法也可以实现分类。两者最主要的区别是，kNN是有监督的，即要有Train集，kNN通过已有的Label（完全正确的），通过最近的kN（kNN中k的含义）个点的Label确定Test的Label[6]。而kMeans是无监督的，不需要Train集，给定数据就能跑。同时kMeans需要反复迭代，kNN一次完成。

从方法上说，kNN是基于统计的（kN的设定也有一套统计学方法），属于lazy-learning，又属于memory-based-learning，本质上是分类算法。kMeans则是Centroid-based，并且有明显的学习过程（迭代），本质上是聚类算法。

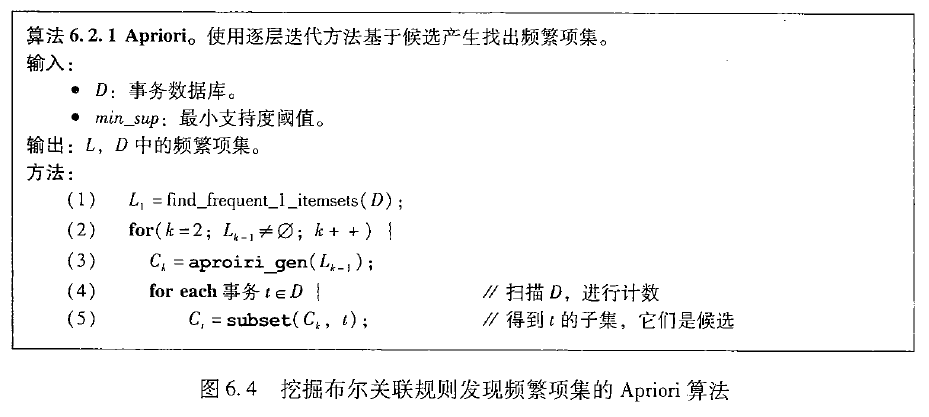
应用上，kNN适合已知Train集来对Test集做预测，聚类数k隐式给定。kMeans则适合对未知数据做预分析，同时聚类数k需要人工显式给出。它们各有优缺点，kNN需要Train集，并且kN需要调整，但是抗噪声能力强（基于统计和距离）。kMeans不需要Train集，但是也要设定k，并且抗异常数据能力弱（基于样本中心的）。

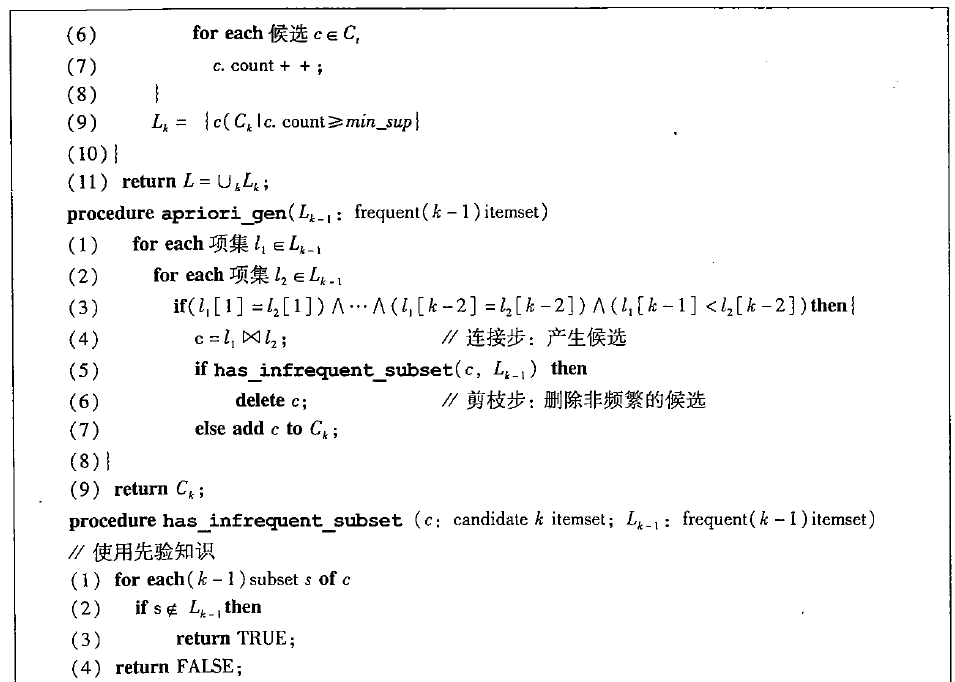
但是它们都有改进的方法，kNN的kN有一套基于统计学的自适应设定方法，kMeans则可以改进聚类中心的初始化方法（kMeans++算法），同时k值也可以利用基于聚类质量评估的方法来自动设定（不吝计算资源的话）。

算法效率上，取决于kMeans的收敛速度。从实现方法上来看，kMeans主要时间花在迭代上，距离计算很快，每个样本点只要计算k次距离（k一般为个位数）。kNN不需要迭代，但是每个样本点对每个Train集样本都要计算距离。kMeans的时间复杂度为O(n\*k\*t)，而kNN为O(n\*n)。如果对kMeans的迭代次数t做限制，则kMeans明显更快一些。不过如果kNN做了基于K-D树的改进实现的话，时间复杂度可以降到O(logN)[7]。

# 基础Apriori算法

## 原理





Apriori是基于频繁项集来确定关联规则的算法。而搜寻频繁项集则利用其反单调性的特点（频繁项集的子集一定也是频繁项集，反之不一定成立；非频繁项集的超集一定也是非频繁项集）。“频繁”的定义也是设定支持度来决定的。从C1候选1项集开始，逐步选择出满足最小支持度的频繁k项集Lk。其中两步非常重要，连接和剪枝。连接是通过取频繁k-1项集来产生可能的频繁k项集。剪枝则在候选k项集Ck中剔除不可能的数据，利用了频繁项集的先验知识——如果Ck有一个子集不是频繁项集，则Ck一定不是频繁项集，由此也看出Apriori（先验的）算法名称的由来。

获得最大频繁项集后，再由大频繁项集开始，产生关联规则，由于大频繁项集是由小频繁项集做并集运算产生的，而算法运行时，对每种可能的频繁项集在事务集合Tractions中的出现次数进行了计数——即绝对支持度，故置信度很容易求得。



其中L为大频繁项集，A为从L中任意选取的单元素组成的子集，B是补集。则置信度：

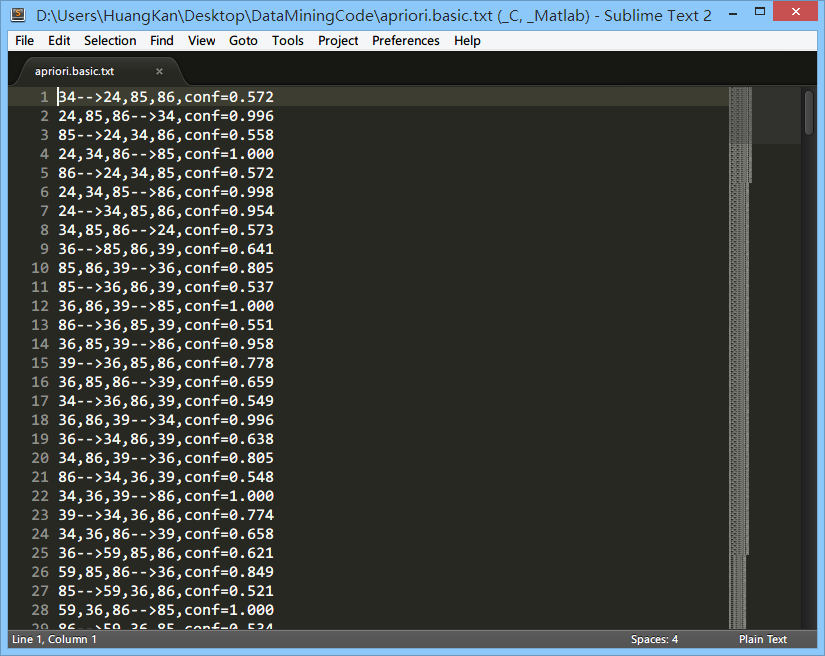


反之：



产生了单元素->多元素和多元素->单元素的关联规则，实际上也可以做多元素->多元素的，但是会产生更多的关联规则，时间复杂度也会更大。

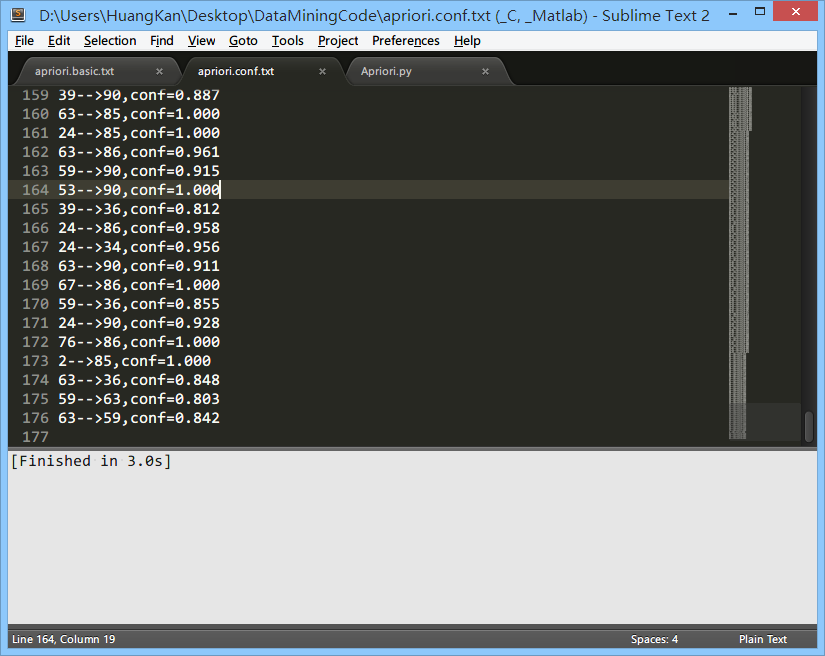
以下是使用Mushroom数据，实现Apriori产生的结果，最小相对支持度设为0.5（至少在T中出现一半以上）。



Mushroom数据共8k多条，算法在3.2s内完成（包括读写文件）。得到了298个关联规则。

# 带约束Apriori算法

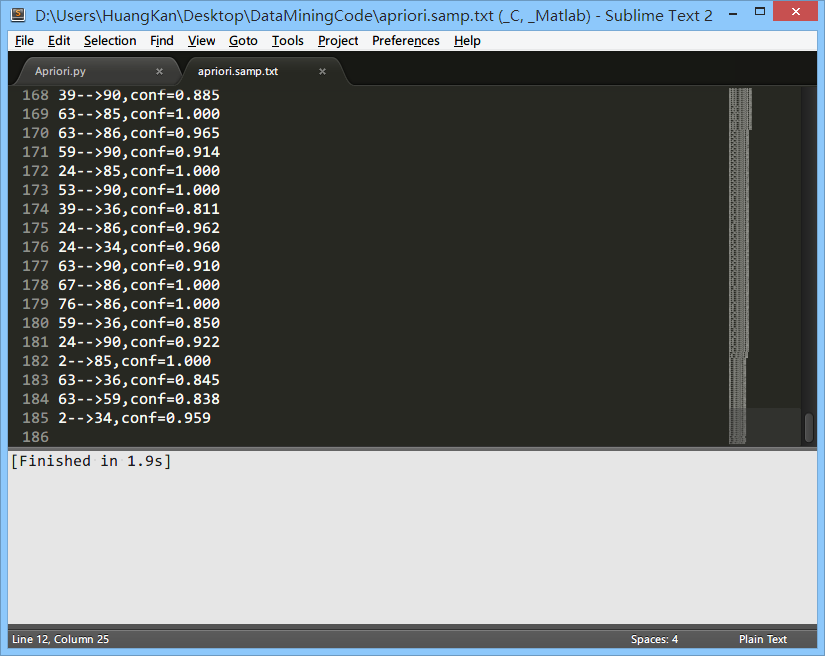
带约束的Apriori算法即除了限制最小支持度，还限制了最小置信度，即confidence小于min\_conf，则该关联规则不予收录。测试程序将min\_conf设置为0.8。在3s内完成，得到了176个规则。



# 改进Apriori算法

改进Apriori算法的方法有很多，基于hash、事务压缩、划分、抽样等，还可以用比较复杂的数据结构要优化算法流程，如FP-Tree算法。我做了比较简单的，对原始数据进行抽样。

只选取原始事务集T中一半的元素（等距离采样），min\_conf和带约束的Apriori一样，设为0.8。结果是1.9s就能完成，并且产生的关联规则更多了，有185个。应该不是程序的问题，可能与数据的分布有关系。



# Apriori算法复杂度

Apriori算法的复杂度相对比较大，主要因为每次从Ck中选择Lk，都要对事务集整个遍历一遍。此外，如果L1数量增大，C2将成倍增长，用基于散列的方法可以减小L1到C2的时间。由于多次扫描数据，Apriori算法受事务集大小影响很大，时间复杂度接近O(T^2)，T为事务集大小，所以采样的方法也是非常可行的，能够很可观地减小时间，从实验结果也能看出来。

FP-growth算法则在时间和空间上都比Apriori快一个量级，但是实现起来比较复杂。

# 感想与总结

这几天都呆在自习室写程序，还是感觉自己效率太低了。代码写得不熟练，不过也和语言的选取有关。记得莫老师说过，语言只是工具，算法的思想是最重要的。

我觉得有一定道理，毕竟语言都是相通的。但是又不那么对。

因为做数模的时候用过聚类，所以算是比较熟悉。但是作业要求面向对象编程，就去查了Mathworks的官方资料，发现Matlab真的可以面向对象编程，而且资料非常详细。但是使用Matlab用户体验十分不好，启动慢，调试也慢，还有各种诸如矩阵维度不匹配的问题，不过可能也与我的编程习惯不太好有关。

各种语言有各自的优缺点，我认为Matlab不适合做算法编程，只适合建模仿真，矩阵运算和数学分析才是Matlab的长处，而不是OO编程。Matlab的语法属于对科学家友好，但是对程序员不友好的。我甚至感觉写Matlab代码久了，自己编程习惯都差了。但是毕竟尝试过，也学了点东西，并且以后还要用过这东西搞科研，也不算坏。

做第二个算法的时候果断放弃Matlab，去学Python的面向对象。同为脚本语言，和Matlab一样，Python省去了C++、Java编译的麻烦，并且十分轻巧，一个文本编辑器和控制台就能写程序了。Python的文本字符串处理能力比较强，自带的数据类型也适合做各种算法。比如Apriori中要做集合合并，一Google发现Python竟然有set类，愈发觉得Python的强大和自己水平还不够。可能别的语言也有，比如C#和Java的Collections类，但是用起来肯定没有Python这么方便了。Python作为开源语言，有各路大牛开发各种库，刚看到一个图形库，效果堪比Matlab，本打算学一学做C4.5的可视化的，但是时间不够也只好放弃了。配合R语言，Python做数据挖掘会十分强大并且方便。

所以语言固然不是最重要的，但是选好工具还是有点必要。但是另一方面，不管用什么语言，只是实现算法是不够的。我感觉自己学得太盲目，忘记了思考。如果想当算法科学家，这种状态肯定不行。上莫老师的课的时候，有一些想法的交流，我才发现自己想法少得可怜，还是学识不够吧，做得不够好。实现算法是一回事，创新和钻研则是另外一回事儿。

最后说说对自己编程的看法，这个学期经常上知乎等程序猿网站，也耳濡目染了一些。已经大三了，到头来发现自己编程水平还远远不够。无论是习惯上、效率上、算法知识上，感觉自己都做得不够好。虽然做数模期间学了很多算法，但是毕竟走马观花，很不熟练，加上数模基地浮躁的环境（很多人直接用网上的代码），并没有锻炼自己真正需要的能力。所以自己大三真的做的不够好。但是这学期也有点进步吧，逼着自己看编程规范文档，改掉坏习惯，也没事看看别人写的代码，挑挑毛病。

这学期因为各种原因没有好好上课，都没有will-to-win的斗志了，记得去年翘课也来蹭数据挖掘来着，可能是“大三病”吧，也很后悔，有时候都找不到学习的意义。最近几天又忙了起来，不管什么事情，不管有没有用，先一件件做好吧，活得充实、学得充实是很重要的。

最后，我马上大四了，特优生也毕业了，以后和莫老师交流的机会不多了，十分感谢莫老师的培养。虽然莫老师上课很多人不听，但是我喜欢这种方式，我觉得比起大部分照本宣科的课强太多了，希望老师继续保持个性，继续开拓学生的思维。

此致。

# 附件清单

|  |  |
| --- | --- |
| FlieName | Description |
| apriori.basic.txt等 | Apriori算法关联规则结果文件 |
| Apriori.py | Apriori各个类 |
| Clusterer.m | Matlab类，集成kMeans和kNN算法 |
| iris.data等，wine.data等 | UCI原始数据，用Python预处理后供Matlab读入的数据 |
| PCA.m | MatlabPCA类，集成降维方法 |
| wineDataAdapter.py等 | Python的数据预处理脚本 |
| wineTestClusterer.m 等 | Matlab用数据测试聚类算法的主脚本 |

# 参考文献

[1]The.Top.Ten.Algorithmsin.DataMining

[2]UCI数据Iris：<http://archive.ics.uci.edu/ml/datasets/Iris>

[3]UCI数据Wine：<http://archive.ics.uci.edu/ml/datasets/Wine>

[4]维基百科-最近邻居法：<https://zh.wikipedia.org/wiki/%E6%9C%80%E8%BF%91%E9%84%B0%E5%B1%85%E6%B3%95>

[5] KNN（K Nearest Neighbor）算法的MatLab实现：<http://blog.csdn.net/rk2900/article/details/9080821>

[6]Kmeans、Kmeans++和KNN算法比较：<http://blog.csdn.net/chlele0105/article/details/12997391>

[7] 从K近邻算法、距离度量谈到KD树、SIFT+BBF算法：<http://www.cnblogs.com/v-July-v/archive/2012/11/20/3125419.html>

[8] PYTHON风格规范：<http://zh-google-styleguide.readthedocs.org/en/latest/google-python-styleguide/python_style_rules/>

[9]mushroom数据集：<http://fimi.ua.ac.be/data/>