目录

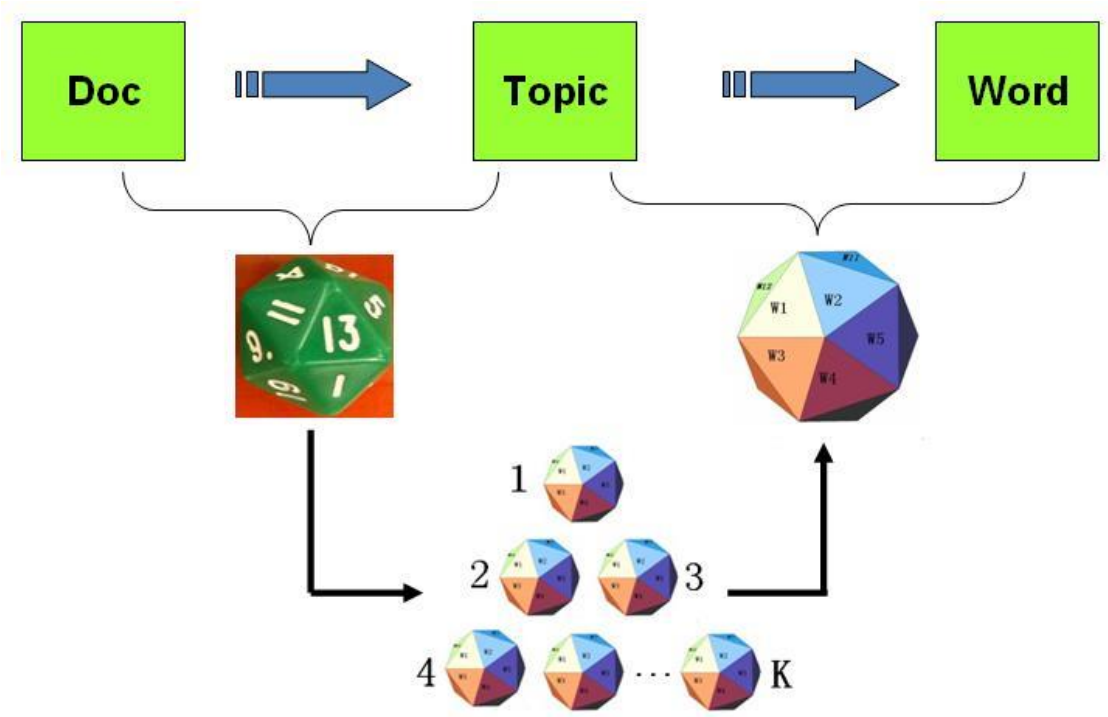
特征提取与数据预处理:

我使用的数据集是yelp 用户评论数据集. 原始数据是一堆用户评论的文本数据. 如何把文本数据转化为有意义的用户特征向量数据.是首先要解决的问题.

LDA:

我采取的做法是使用 LDA 进行文档主题建模.

LDA 是一种非监督机器学习技术，可以用来识别大规模文档集（document collection）或语料库（corpus）中潜藏的主题信息。它采用了词袋（bag of words）的方法，这种方法将每一篇文档视为一个词频向量，从而将文本信息转化为了易于建模的数字信息。但是词袋方法没有考虑词与词之间的顺序，这简化了问题的复杂性。每一篇文档代表了一些主题所构成的一个概率分布，而每一个主题又代表了很多单词所构成的一个概率分布。



LDA 认为一篇文档由一些主题按照一定概率组成，一个主题又由一些词语按照一定概率组成。早期人们用词袋模型对一篇文章进行建模，把一篇文档表示为若干单词的计数。无论是中文还是英文，都由大量单词组成，这就造成词袋向量的维数巨大，少则几千多则上万，在使用分类模型进行训练时，非常容易造成训练缓慢以及过拟合。LDA 本质上把词袋模型进行了降维，把一篇文档以主题的形式进行了表示。
主题的个数通常为几百，这就把文档使用了维数为几百的向量进行了表示，大大加快了训练速度，并且相对不容易造成过拟合。从某种程度上来说，主题是对若干词语的抽象表示。所以使用 LDA 提取主题的过程也就是对高维的词袋向量降维的过程,而这正是我需要的.

这样的话 LDA 的结果可以作为进一步文档分类、文档相似度计算以及文档聚类的依据，所以可以把 LDA 当做一种特征提取方法。

TF-IDF:

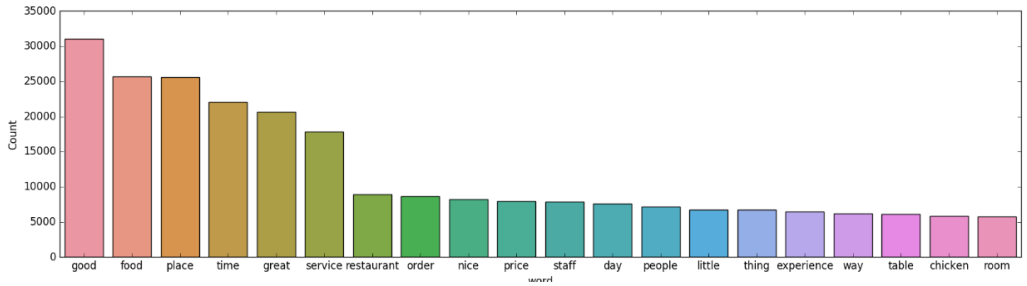
tf-idf 方法实际是对简单的词频矩阵的扩展.它的基本思想是: 如果某个词比较少见，但是它在这篇文章中多次出现, 那么它很可能就反映了这篇文章的特性, 正是我们所需要的关键词。

用统计学语言表达，就是在词频的基础上，要对每个词分配一个"重要性"权重。最常见的词（"的"、"是"、"在"）给予最小的权重，较常见的词（"中国"）给予较小的权重，较少见的词（"蜜蜂"、"养殖"）给予较大的权重。这个权重叫做"逆文档频率"（Inverse Document Frequency，缩写为 IDF），它的大小与一个词的常见程度成反比。

知道了"词频"（TF）和"逆文档频率"（IDF）以后，将这两个值相乘，就得到了一个词的 TF-IDF 值。某个词对文章的重要性越高，它的 TF-IDF 值就越大。所以，排在最前面的几个词，就是这篇文章的关键词。

特征提取过程:

1. 首先需要从评论数据处理,构造每条评论的词袋.
2. 对此袋进行过滤,过滤掉停用词,不规范的词,只保留有实际意义的动词,名词,形容词,副词
3. 词性还原,词语还原回词根



这是经过处理后的词频排序的词语

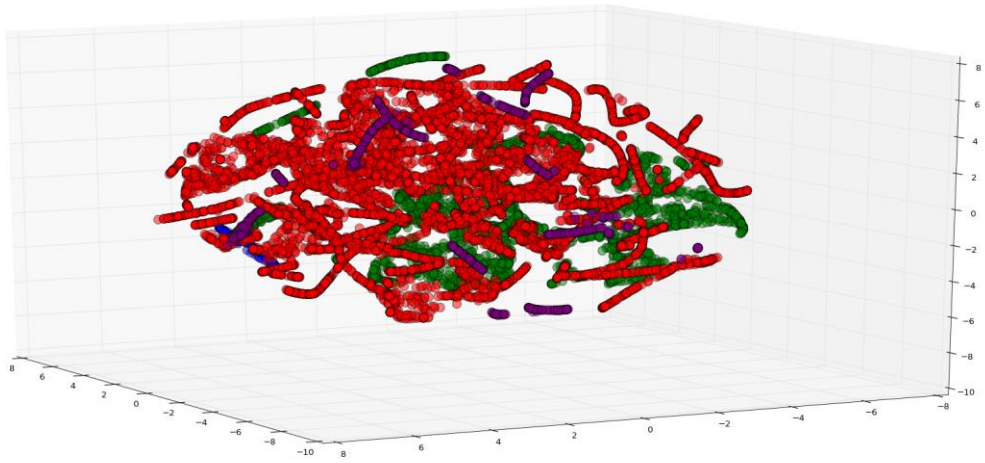
4. 得到文档-单词矩阵 （直接利用统计词频得到特征）
5. 将文档-单词矩阵转化为一个词袋，得到文档-单词矩阵
6. 使用 TF-IDF 处理文档-单词矩阵
7. 使用 LDA 模型训练文档-单词矩阵与词袋
   这是 LDA 处理得到的 8 个主题

```
0.002*"da" + 0.002*"die" + 0.002*"der" + 0.002*"ich" + 0.002*"ist" + 0.002*"und" + 0.001*"sehr" + 0.001*"nicht" + 0.001*"mit" + 0.001*"ein"
0.001*"anna" + 0.001*"necklace" + 0.001*"lauren" + 0.000*"ken" + 0.000*"threading" + 0.000*"kia" + 0.000*"dispatcher" + 0.000*"gabriel" + 0.000*"brow" + 0.000*"atv"
0.003*"food" + 0.003*"great" + 0.003*"good" + 0.003*"place" + 0.003*"service" + 0.003*"time" + 0.002*"get" + 0.002*"would" + 0.002*"back" + 0.002*"staff"
0.003*"good" + 0.002*"food" + 0.002*"place" + 0.002*"chicken" + 0.002*"great" + 0.002*"really" + 0.002*"delicious" + 0.002*"amazing" + 0.002*"restaurant" + 0.002*"ordered"
0.001*"attorney" + 0.001*"loan" + 0.001*"sarah" + 0.001*"edinburgh" + 0.001*"ohio" + 0.001*"healing" + 0.001*"reschedule" + 0.001*"blamed" + 0.001*"sitter" + 0.001*"paula"
0.001*"rick" + 0.001*"sunglass" + 0.001*"instrument" + 0.000*"dentistry" + 0.000*"jenny" + 0.000*"good" + 0.000*"ruben" + 0.000*"great" + 0.000*"food" + 0.000*"prince"
0.001*"audi" + 0.000*"taylor" + 0.000*"awsome" + 0.000*"greg" + 0.000*"mgr" + 0.000*"jake" + 0.000*"denver" + 0.000*"shannon" + 0.000*"brenda" + 0.000*"great"
0.001*"andrea" + 0.000*"pam" + 0.000*"firestone" + 0.000*"stuart" + 0.000*"lucy" + 0.000*"grandson" + 0.000*"kevin" + 0.000*"mandy" + 0.000*"flay" + 0.000*"edc"
```

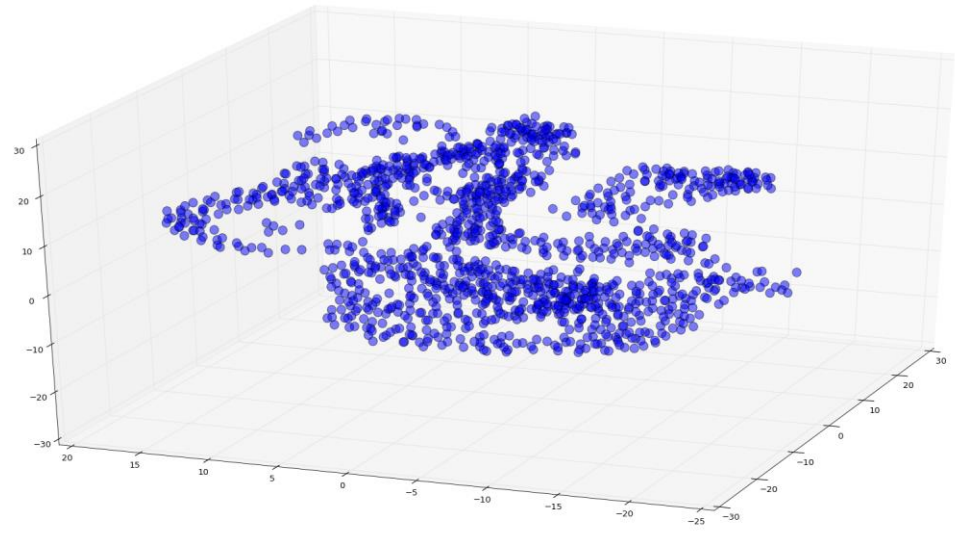
8. 通过 LDA 模型计算得到特征矩阵(即该评论属于某个主题的概率)

数据预处理:

得到评论特征后,我又对得到的数据进行了二次处理. 其实这次处理是被迫进行的.因为评论数据的主题分布不均.(大部分评论都只属于第三个主题)所以聚类效果并不明显.而且把数据还原为 3 维后,因为数据量太大,也看不出来数据分布特征.



这是预处理前的数据,基本可视化没有任何效果.聚类效果也看不出来.
所以我对评论数据做了如下处理.对 8 维特征,每一维选取特征最明显的 200 个数据(可能有重复),得到处理后的数据.



可以看到分布特征现在就可以很明显的展示出来.

参考链接:

https://www.analyticsvidhya.com/blog/2018/10/mining-online-reviews-topic-modeling-lda/

https://github.com/duoergun0729/nlp/blob/master/%E4%BD%BF%E7%94%A8LDA%E8%BF%9B%E8%A1%8C%E6%96%87%E6%A1%A3%E4%B8%BB%E9%A2%98%E5%BB%BA%E6%A8%A1.md
https://cosx.org/2010/10/lda_topic_model/
http://www.ruanyifeng.com/blog/2013/03/cosine_similarity.html

关于可视化与降维：

根据之前的讨论，我对评论数据处理后得到的是 8 维的用户特征矩阵。之后的聚类将会在 8 维上进行。但是我们只能可视化 3 维一下的数据，而如果没有可视化的结果，那我很难对自己的数据分布以及聚类结果有全面的认识。这就要求我需要寻找一个降维工具用户数据可视化。

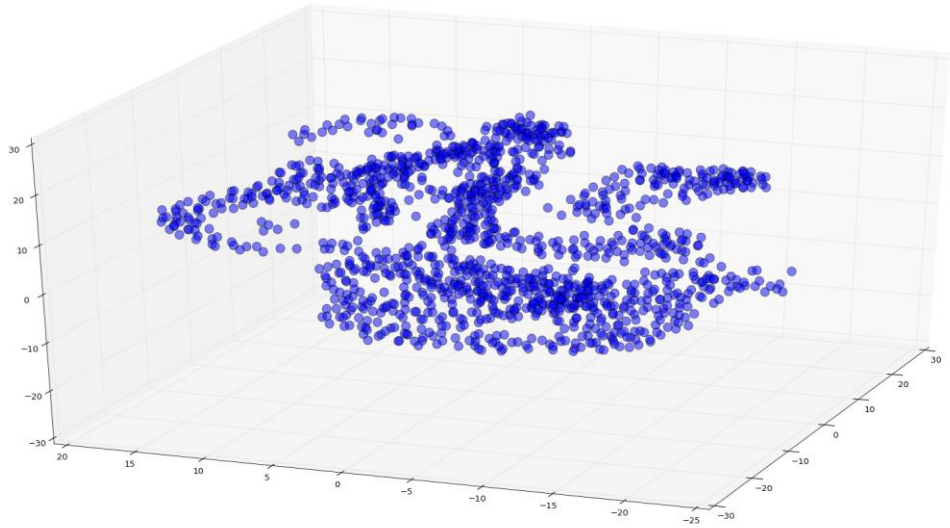
经过调研，t-SNE 非常适合可视化的降维场景。
t-SNE 可将样本点间的相似度关系转化为概率：在原空间（高维空间）中转化为基于高斯分布的概率；在嵌入空间（二维空间）中转化为基于 t 分布的概率。这使得 t-SNE 不仅可以关注局部（SNE 只关注相邻点之间的相似度映射而忽略了全局之间的相似度映射，使得可视化后的边界不明显），还关注全局，使可视化效果更好（簇内不会过于集中，簇间边界明显）

总结：
1. t-SNE 是一种非线性降维方法，可以同时考虑数据全局与局部关系。
2. 它的核心思想是保证在低维上的数据分布与原始特征空间中的分布相似度高。
3. t-SNE 的运算开销比较大 o(n^2)经过优化可以达到 o(nlogn)。

因为我们的降维操作不需要频繁去做（只要一次降维保存降维的结果，之后每次加入标签数据作图即可）所以 t-SNE 的运算开销劣势并不明显。但是相对于其他降维算法，它不仅关注局部而且兼顾全局，使可视化效果更好，优势明显。

考虑到数据原始维度较高，而且降维一定不可避免损失一些信息；所以我选择将原始数据降到 3 维，尽可能减少这方面的损失。

这是数据降维后的可视化效果。



参考链接

https://www.zhihu.com/question/52022955
https://chenrudan.github.io/blog/2016/04/01/dimensionalityreduction.html
https://blog.csdn.net/hustqb/article/details/80628721

聚类结果评价标准：

- 仅讨论作业中使用的几种评价标准
- 仅讨论没有真实标签的评价标准

轮廓系数:

轮廓系数 (Silhouette Coefficient) 结合了聚类的凝聚度 (Cohesion) 和分离度 (Separation)，用于评估聚类的效果。该值处于-1~1 之间，值越大，表示聚类效果越好。

具体计算方法如下:

1. 对于每个样本点 i，计算点 i 与其同一个簇内的所有其他元素距离的平均值，记作 a(i)，用于量化簇内的凝聚度。
2. 选取 i 外的一个簇 b，计算 i 与 b 中所有点的平均距离，遍历所有其他簇，找到最近的这个平均距离,记作 b(i)，即为 i 的邻居类，用于量化簇之间分离度。
3. 对于样本点 i，轮廓系数 s(i) = (b(i) – a(i))/max{a(i),b(i)}
4. 计算所有 i 的轮廓系数，求出平均值即为当前聚类的整体轮廓系数，度量数据聚类的紧密程度

使用注意：

1. 不正确的聚类得分为-1，而高度密集的聚类得分为+1。0 分左右表示重叠的集群。
2. 凸簇的得分一般比其他簇（例如基于密度方法得到的簇）得分要高
3. 计算复杂度较高 o(n^2)

方差比标准

CH 指标（Calinski-Harabaz Index）通过计算类中各点与类中心的距离平方和来度量类内的紧密度，通过计算各类中心点与数据集中心点距离平方和来度量数据集的分离度，CH 指标由分离度与紧密度的比值得到。从而，CH 越大代表着类自身越紧密，类与类之间越分散，即更优的聚类结果。

For k clusters, the Calinski-Harabaz score s is given as the ratio of the between-clusters dispersion mean and the within-cluster dispersion:

$$s(k) = \frac{\mathrm{Tr}(B_k)}{\mathrm{Tr}(W_k)} \times \frac{N-k}{k-1}$$

where B_K is the between group dispersion matrix and W_K is the within-cluster dispersion matrix defined by:

$$W_k = \sum_{q=1}^{k} \sum_{x \in C_q} (x - c_q)(x - c_q)^T$$

$$B_k = \sum_{q} n_q (c_q - c)(c_q - c)^T$$

使用注意：

1. 计算复杂度更低
2. 得分越高代表簇的内聚性越高，分离度越低，聚类结果更优
3. 同轮廓系数一样，凸簇的得分一般比其他簇高。

DB 指数

DB 指数（Davies-Bouldin Index）被定义为各个聚类的平均最小相似度，更小的 DB 指数表示聚类间相似度越低，模型聚类结果更优。

The index is defined as the average similarity between each cluster C_i for $i = 1, \ldots, k$ and its most similar one C_j. In the context of this index, similarity is defined as a measure R_{ij} that trades off:

- s_i, the average distance between each point of cluster i and the centroid of that cluster – also know as cluster diameter.
- d_{ij}, the distance between cluster centroids i and j.

A simple choice to construct $R_i j$ so that it is nonnegative and symmetric is:

$$R_{ij} = \frac{s_i + s_j}{d_{ij}}$$

Then the Davies-Bouldin index is defined as:

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{i \neq j} R_{ij}$$

使用注意：

1. 计算复杂度更低
2. 得分越高效果不一定最好。
3. 只适用于欧式距离聚类算法
4. 同轮廓系数一样，凸簇的得分一般比其他簇高。

肘部法则（簇内误差平方和）

肘部法则的计算原理是成本函数，成本函数是类别畸变程度之和，每个类的畸变程度等于每个变量点到其类别中心的位置距离平方和，若类内部的成员彼此间越紧凑则类的畸变程度越小，反之，若类内部的成员彼此间越分散则类的畸变程度越大。

在选择类别数量上，肘部法则会把不同值的成本函数值画出来。随着值的增大，平均畸变程度会减小；每个类包含的样本数会减少，于是样本离其重心会更近。但是，随着值继续增大，平均畸变程度的改善效果会不断减低。值增大过程中，畸变程度的改善效果下降幅度最大的位置对应的值就是肘部。

肘部法则考虑的仅仅是单个聚类的紧凑程度，没有考虑聚类间的关系。

参考链接

https://blog.csdn.net/sinat_33363493/article/details/52496011
https://blog.csdn.net/u010159842/article/details/78624135
https://scikit-learn.org/0.17/modules/classes.html#clustering-metrics

http://www.icode9.com/content-4-120567.html

K-means 算法

算法简述

K-Means 算法的思想很简单，对于给定的样本集，按照样本之间的距离大小，将样本集划分为 K 个簇。让簇内的点尽量紧密的连在一起，而让簇间的距离尽量的大。

首先算法选择 k 个点作为中心点；进行聚类输出 k 个簇；重新计算簇的中心点。多次迭代直到中心点偏移量很小。

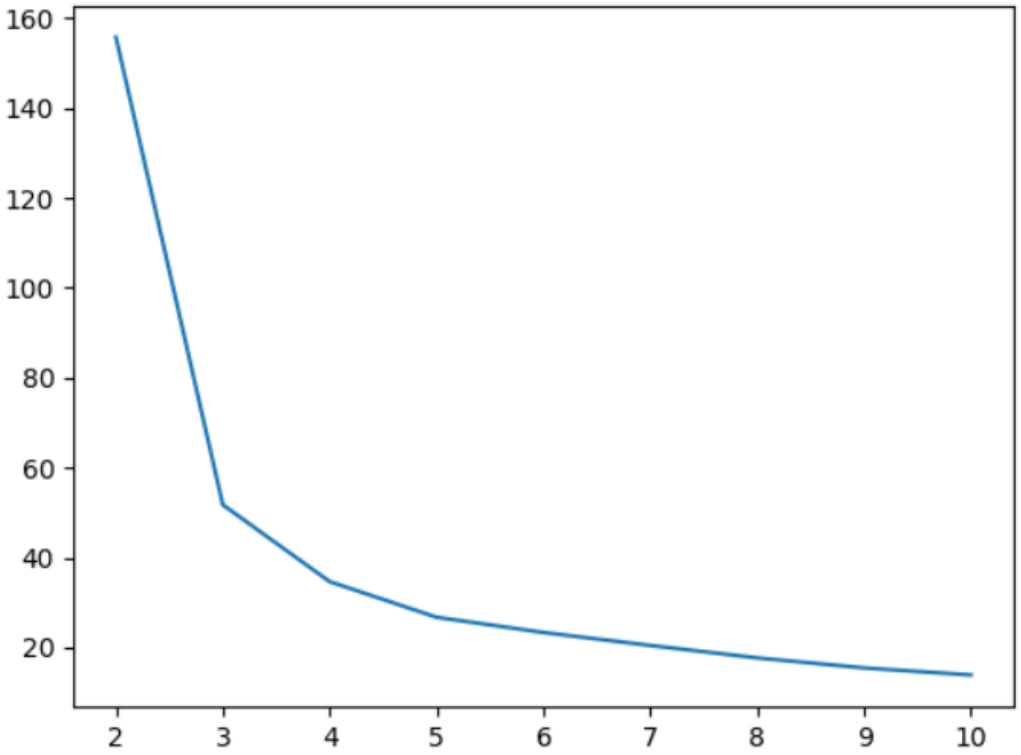
参数确定

K-Means 算法最重要的参数只有一个 k，聚类个数的选取。

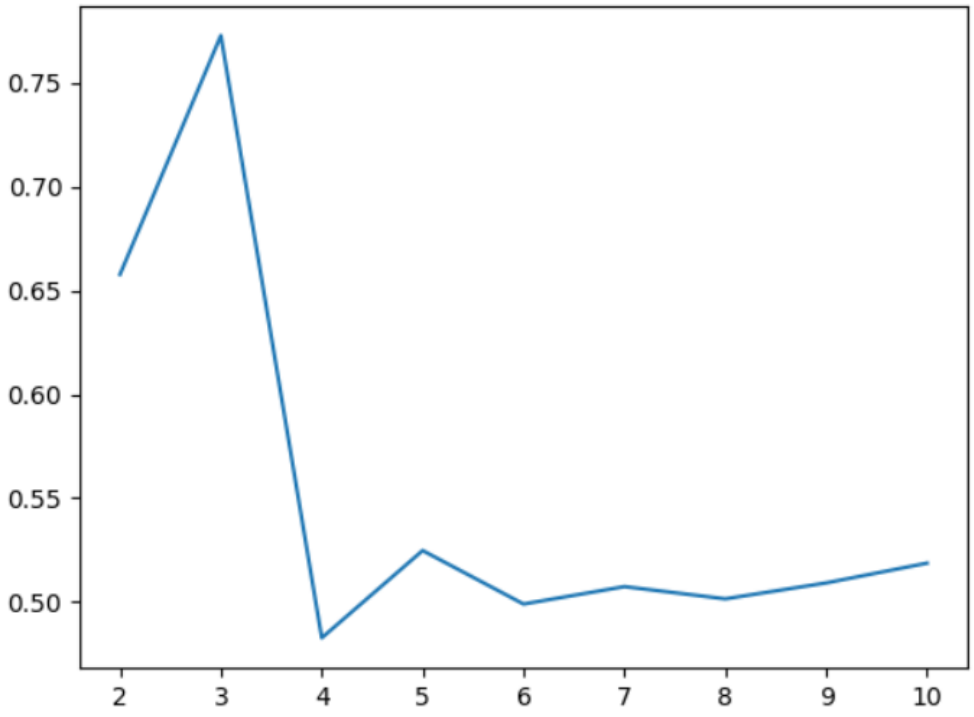
因为我对数据没有足够的先验知识，只能通过一些聚类效果评价标准，结合可视化分析确定聚类个数。

具体流程是，我遍历 k 从 2 到 10，每个聚类多次尝试取评价指标（轮廓系数、肘部法则、方差比标准）的平均值，画出指标变化曲线，结果如下。

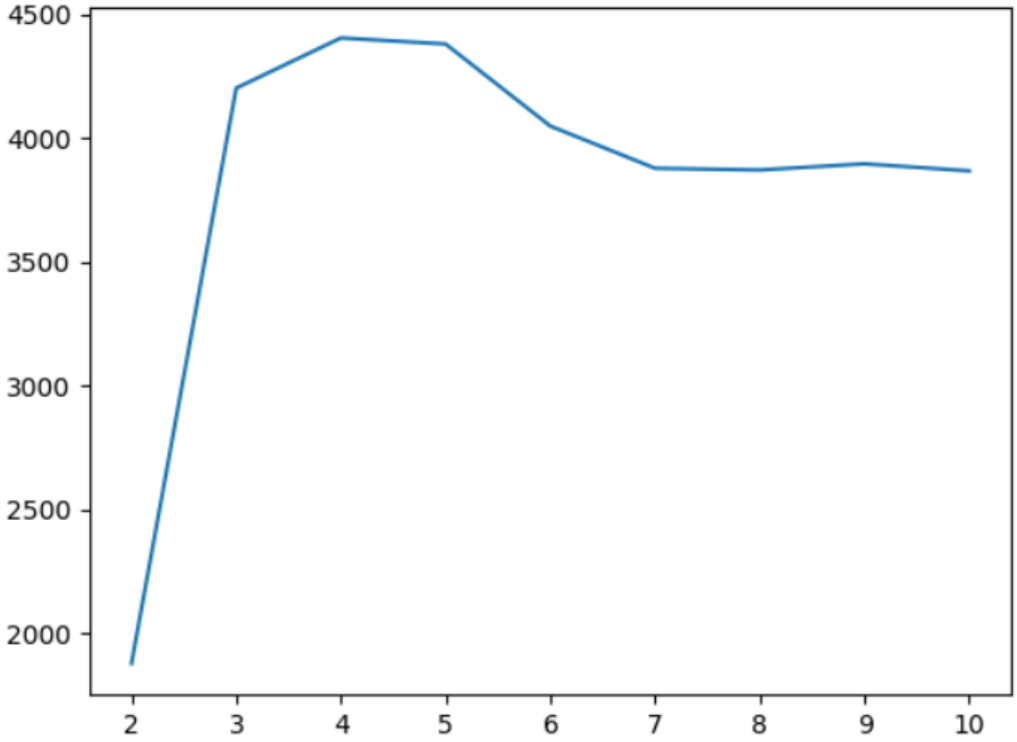
肘部法则：


160
140
120
100
80
60
40
20
2
3
4
5
6
7
8
9
10

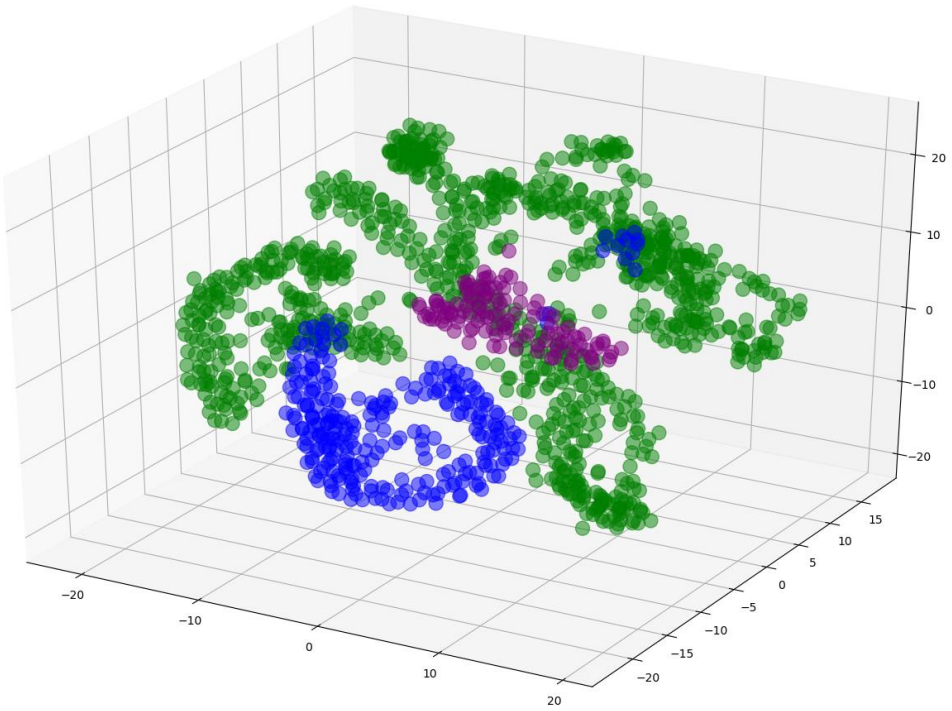
轮廓系数:

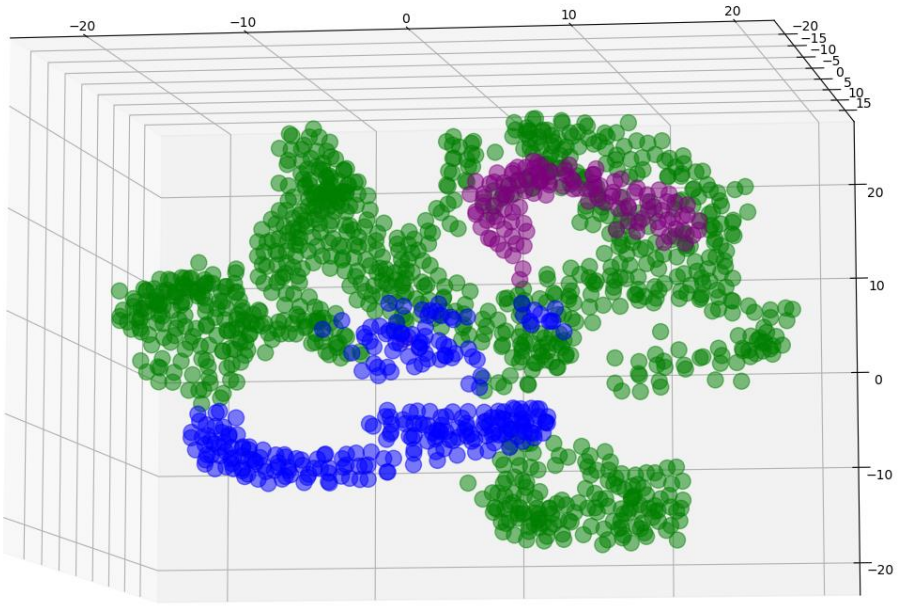

0.75
0.70
0.65
0.60
0.55
0.50
2
3
4
5
6
7
8
9
10

方差比标准:



根据图像显示，这几种评价指标的结果指向还是很统一的（除了方差比标准不太明显），所以我选择的 k 值为 3。接下来是聚类的可视化效果。





从降维后的可视化结果来看，结果还算不错。

DBSCAN 算法

算法简述

DBSCAN 是一种基于密度的聚类算法，这类密度聚类算法一般假定类别可以通过样本分布的紧密程度决定。同一类别的样本，他们之间的紧密相连的，也就是说，在该类别任意样本周围不远处一定有同类别的样本存在。

通过将紧密相连的样本划为一类，这样就得到了一个聚类类别。通过将所有各组紧密相连的样本划为各个不同的类别，则我们就得到了最终的所有聚类类别结果。

参数确定

与 k-means 不同 dbscan 不能直接指定聚类结果的个数。它的参数主要有两个：

1）eps： DBSCAN 算法参数，即我们的 ϵ-邻域的距离阈值，和样本距离超过 ϵ 的样本点不在 ϵ-邻域内。默认值是 0.5.一般需要通过在多组值里面选择一个合适的阈值。eps 过大，则

更多的点会落在核心对象的 ϵ-邻域，此时我们的类别数可能会减少， 本来不应该是一类的样本也会被划为一类。反之则类别数可能会增大，本来是一类的样本却被划分开。

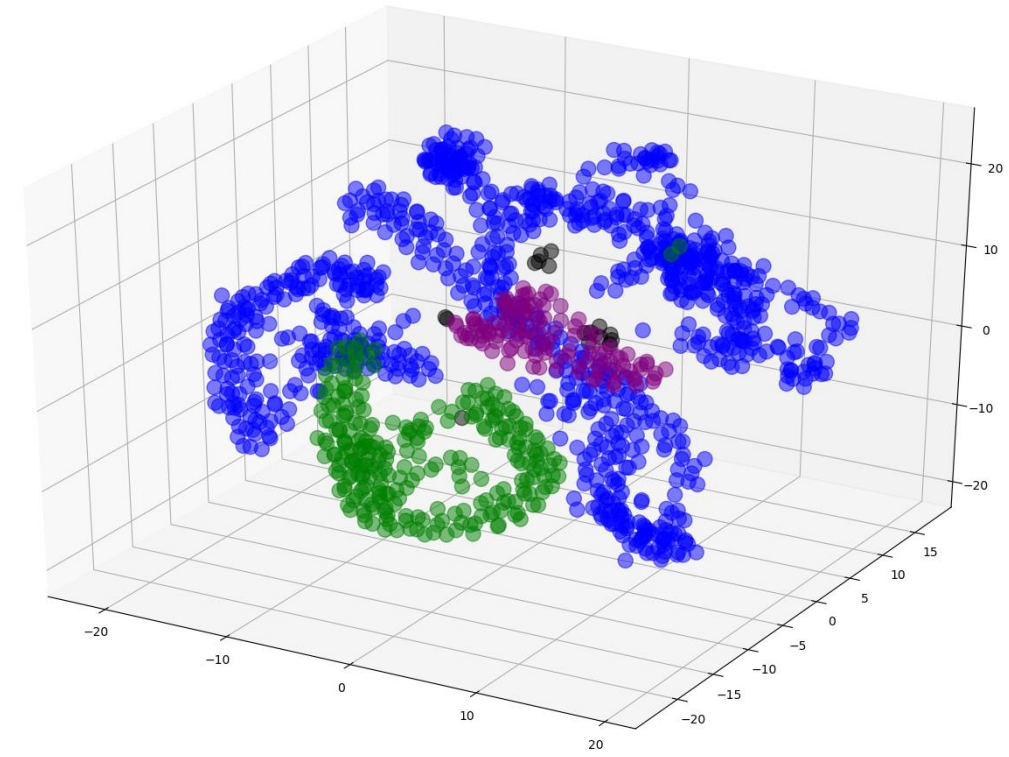
2）min_samples： DBSCAN 算法参数，即样本点要成为核心对象所需要的 ϵ-邻域的样本数阈值。默认值是 5. 一般需要通过在多组值里面选择一个合适的阈值。通常和 eps 一起调参。在 eps 一定的情况下，min_samples 过大，则核心对象会过少，此时簇内部分本来是一类的样本可能会被标为噪音点，类别数也会变多。反之 min_samples 过小的话，则会产生大量的核心对象，可能会导致类别数过少

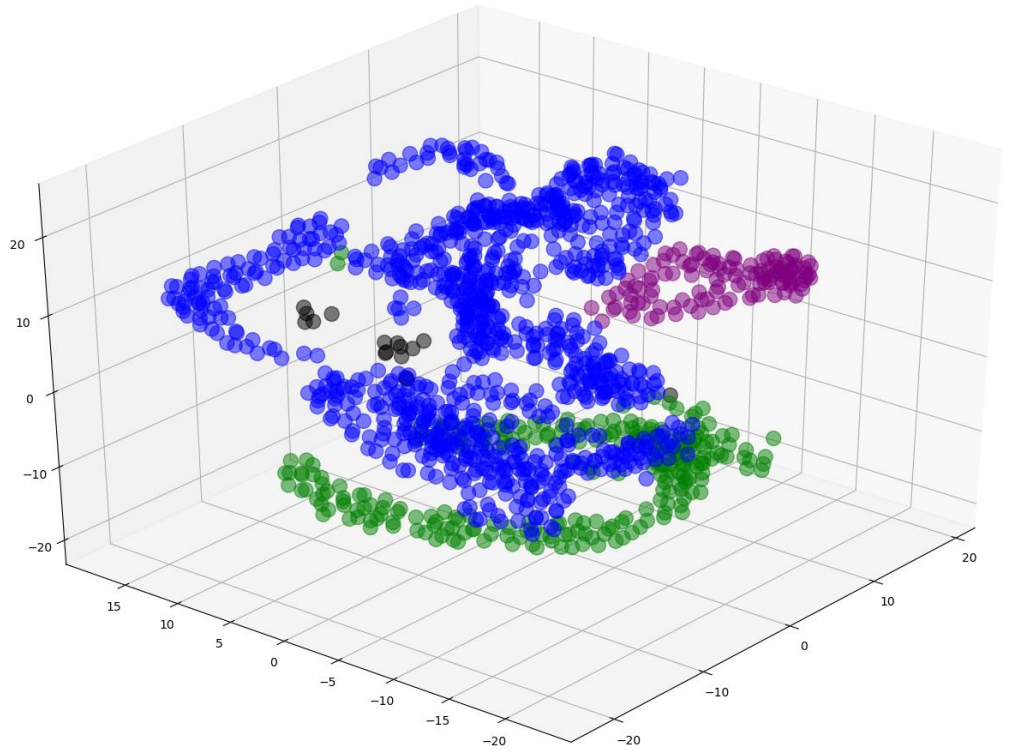
根据我多次实验的结果（详见 result1.txt）：

```
eps = 0.2
min_samples = 15
0 : 1243
1 : 118
s_score: 0.42339346
c_score: 289.5906061228337

eps = 0.2
min_samples = 20
0 : 982
1 : 260
2 : 118
s_score: 0.7203446
c_score: 3197.564700893969

eps = 0.2
min_samples = 25
0 : 981
1 : 260
2 : 118
s_score: 0.7164269
c_score: 3196.6908955635176

eps = 0.2
min_samples = 30
0 : 980
1 : 260
2 : 118
s_score: 0.7105
c_score: 3190.578267906501
```

eps = 0.2 min_sample=20 时效果最好

可视化：





可以看出与 kmeans 结果相差不大

spectral clustering 算法

算法简述

Spectral Clustering（谱聚类，有时也简称 SC），其实是一类算法的统称。它是一种基于图论的聚类方法（而 K-Means 是基于点与点的距离计算），它能够识别任意形状的样本空间且收敛于全局最有解，其基本思想是利用样本数据的相似矩阵进行特征分解后得到的特征向量进行聚类。

我对于谱聚类的理解是，原本相似度矩阵就是对样本点的一种特征表达（特征维数等于样本数），现在进行了谱聚类求得的特征值矩阵，实际上是对原始特征矩阵的一种降维（也可能是升维），总之就是将样本从原始空间变换（可能是线性的也可能是非线性的）到另一个空间，在这个空间中具有良好的全局欧式性。

参数确定

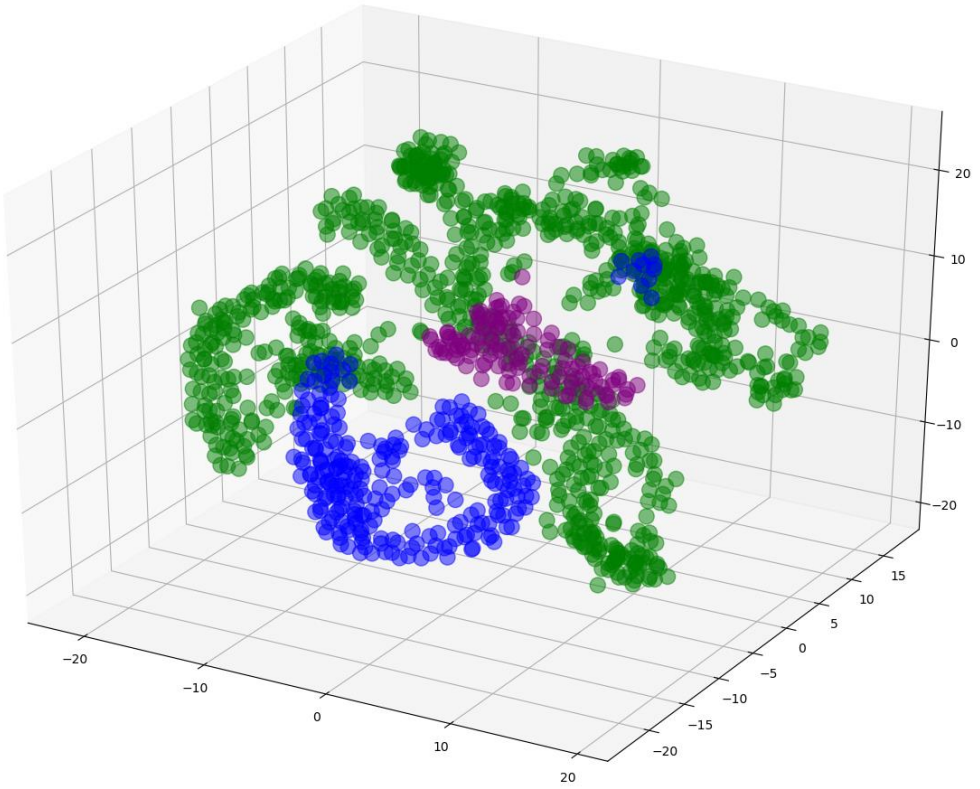
1） n_clusters：代表我们在对谱聚类切图时降维到的维数，同时也是最后一步聚类算法聚类到的维数。也就是说 scikit-learn 中的谱聚类对这两个参数统一到了一起。简化了调参的参数个数。
2） 核函数参数 gamma:：这个参数会在谱聚类确定相似矩阵时发挥作用。一般使用的高斯核函数。高斯核函数中这个参数对应 $K(x,z)=\exp(-\gamma||x-z||2)$中的 γ。

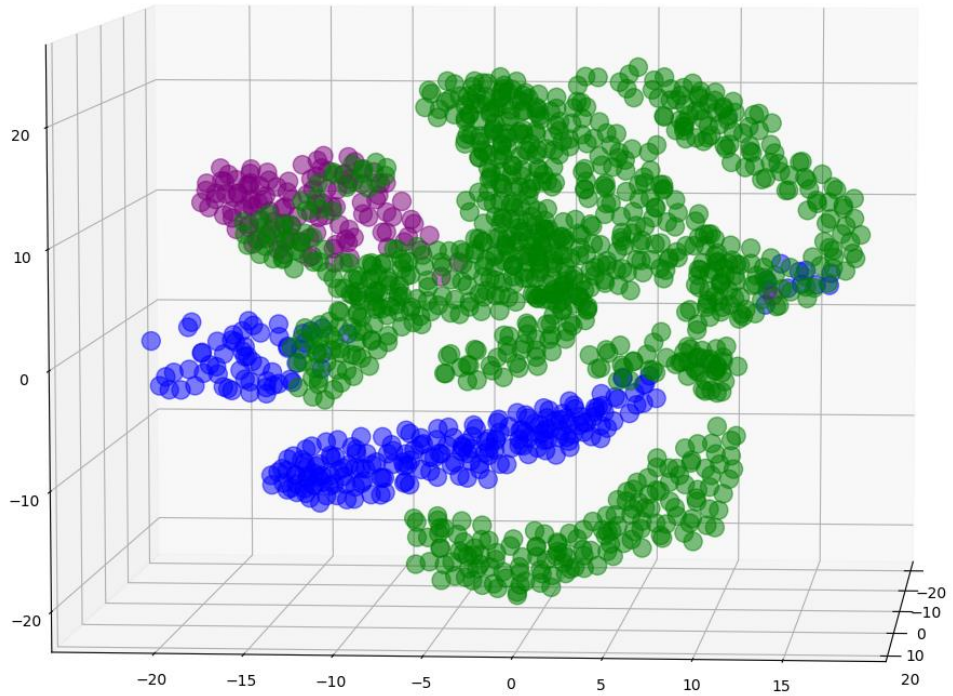
根据实验结果（详见 result2.txt）：

```
gamma = 0.12
_clusters = 3
0 : 987
1 : 270
2 : 119
s_score: 0.77325916
c_score: 4194.621523859814

gamma = 0.12
_clusters = 4
0 : 370
1 : 628
2 : 118
3 : 260
s_score: 0.45074615
c_score: 3791.3132659729044

gamma = 0.15
_clusters = 3
0 : 270
1 : 987
2 : 119
s_score: 0.77325916
c_score: 4194.621523859814

gamma = 0.15
_clusters = 4
0 : 258
1 : 118
2 : 364
```

gamma = 0.15
n_cluster = 3

可视化结果:


20
10
0
−10
−20
15
10
5
0
−5
−10
−15
−20
−20
−10
0
10
20


20
10
0
−10
−20
−20
−15
−10
−5
0
5
10
15
−20
−10
0
10
20

Hierarchical clustering 算法

算法简述

层次聚类(Hierarchical Clustering)通过计算不同类别数据点间的相似度来创建一棵有层次的嵌套聚类树。在聚类树中，不同类别的原始数据点是树的最低层，树的顶层是一个聚类的根节点。创建聚类树有自下而上合并和自上而下分裂两种方法。

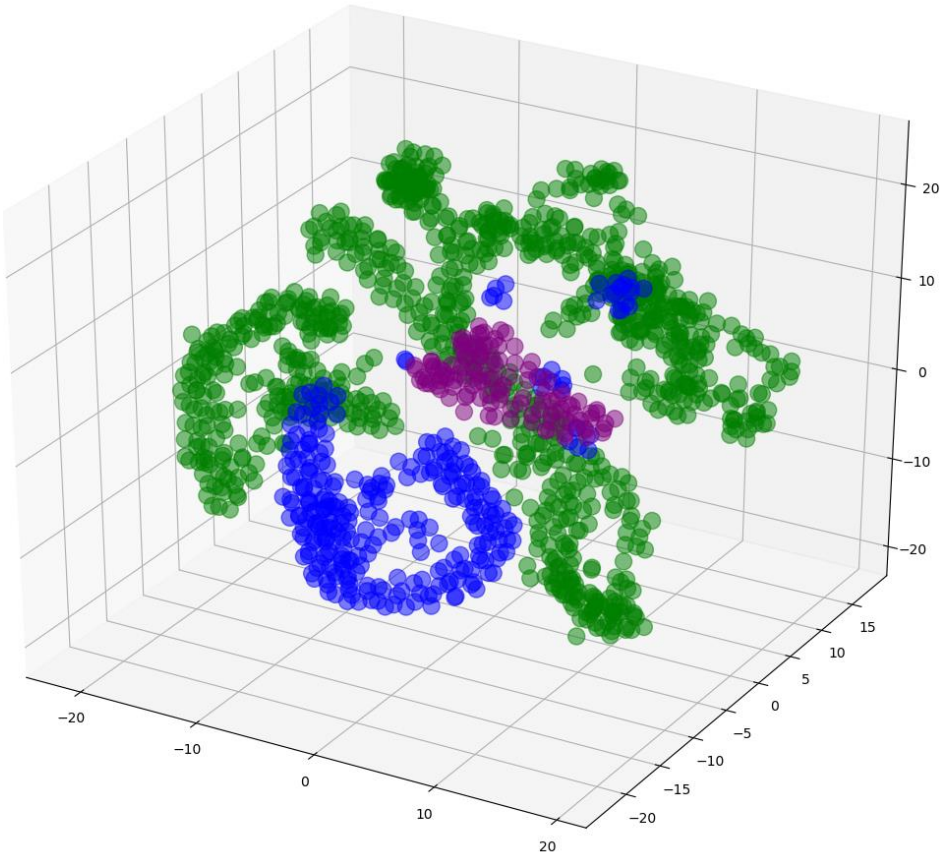
参数确定

1） n_clusters：聚类个数;

2） linkage='ward'： {“ward”, “complete”, “average”}，计算类簇间距离的方法，“ward”：所有类簇的方差和，“complete”：取两个集合中距离最远的两个点的距离作为两个集合的距离，“average”：把两个集合中的点两两的距离全部放在一起求一个平均值，Agglomerative cluster 算法中存在“rich get richer”的现象，导致聚类大小不均匀，对此，“complete”是最坏策略，“ward”给出了最规则的大小，但是 linkage 是“ward”，affinity 只能是“euclidean”，所以对于 affinity 不是“euclidean”的情况，“average”是一个好的选择

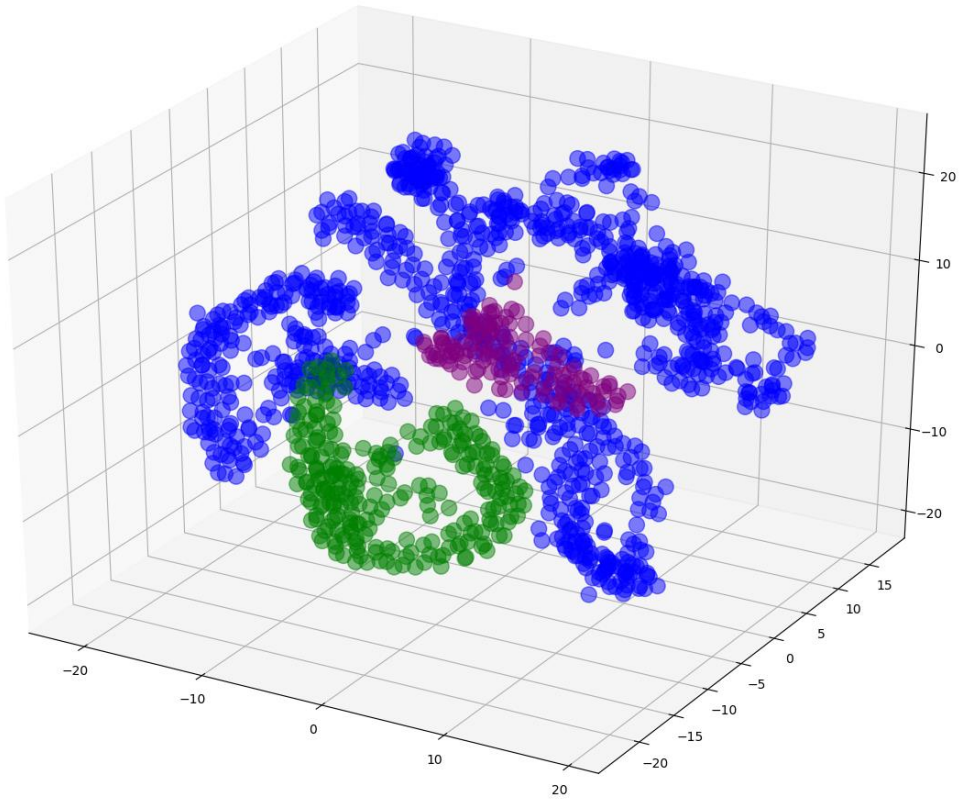
因为之前的实验证明最优聚类个数是 3.所以这里我没有再在聚类个数上做文章。而是转而研究 linkage 这个参数。

linkage = ‘complete’



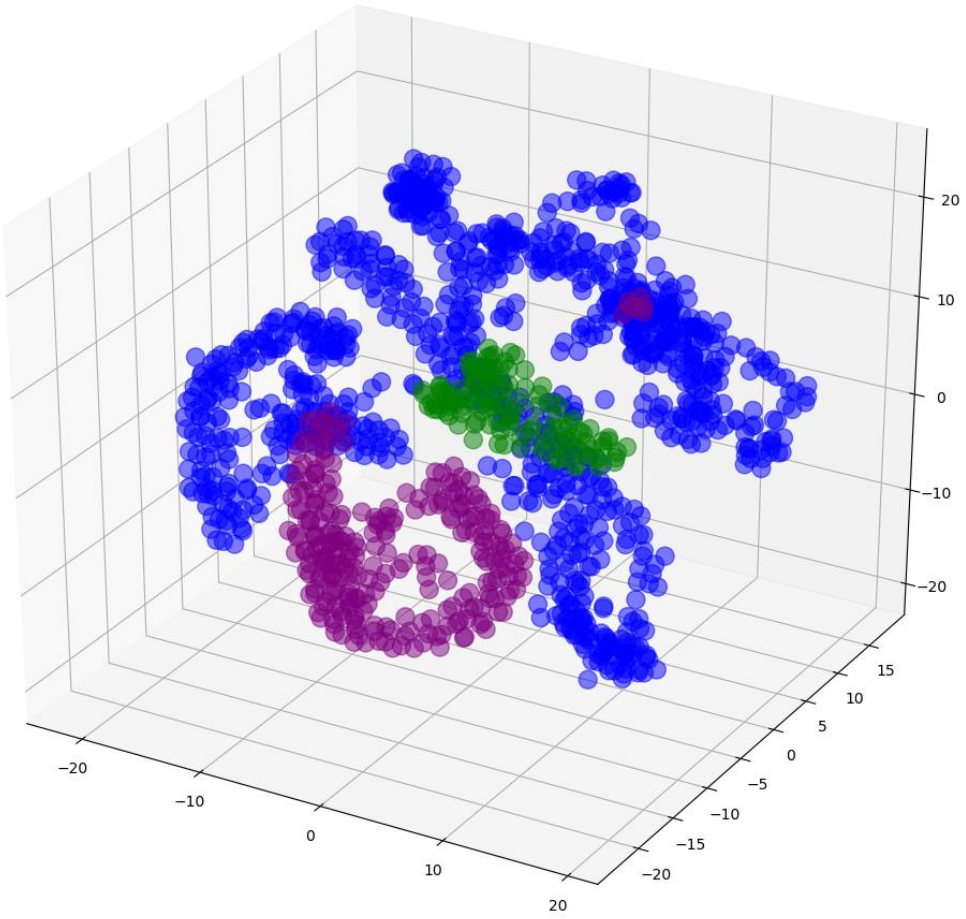

```
0 : 295
1 : 963
2 : 118
s_score: 0.7561547
c_score: 3656.045139898589
```

linkage = ‘ward




```
0 : 999
1 : 258
2 : 119
s_score: 0.7707766
c_score: 3987.3791291379353
```

linkage='average':




```
0 : 993
1 : 118
2 : 265
s_score: 0.7739067
c_score: 4158.754775623942
```

从可视化结果来看，三种参数选取基本没有太多差别。
从评价标准来看，显然是‘complete’的效果最差，但反而是 linkage=’average’时效果最好，当然好的并不明显。
不过观察三种聚类的数目分布倒是有些有趣的地方，可以看出来三种聚类中，有一个聚类的个数基本不变一直是 119 或 118。即图片右上角的那小部分。影响变化的只有另两大类（数目最大和次大的两类）的结果。

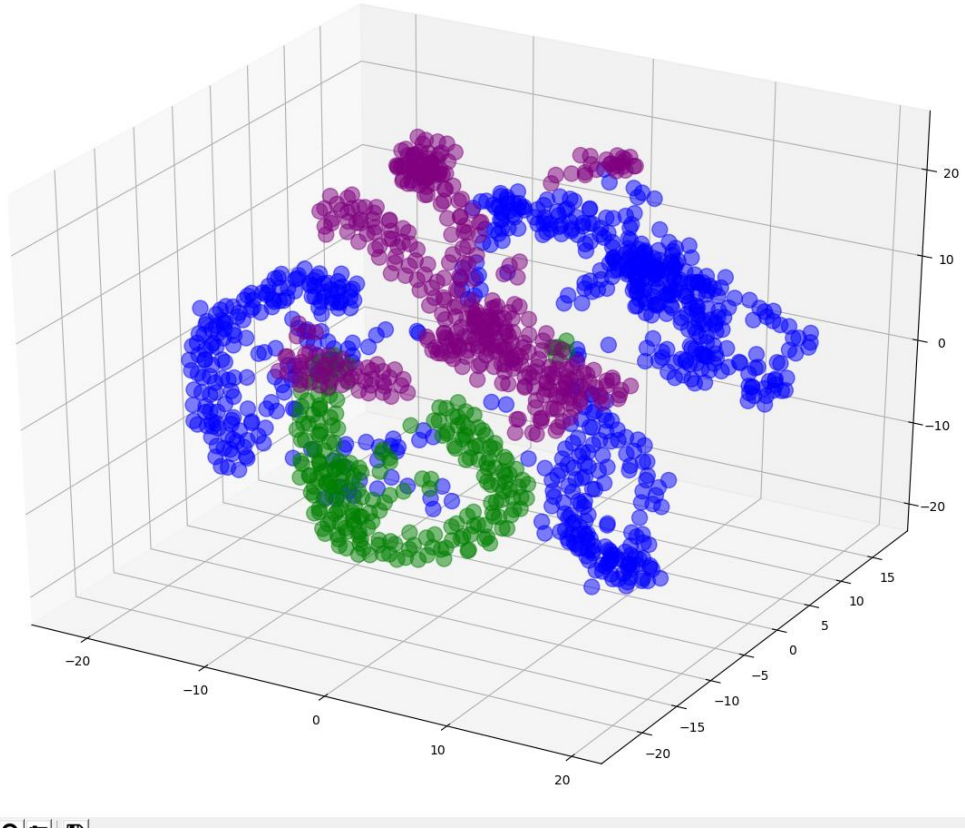
EM-GMM 算法

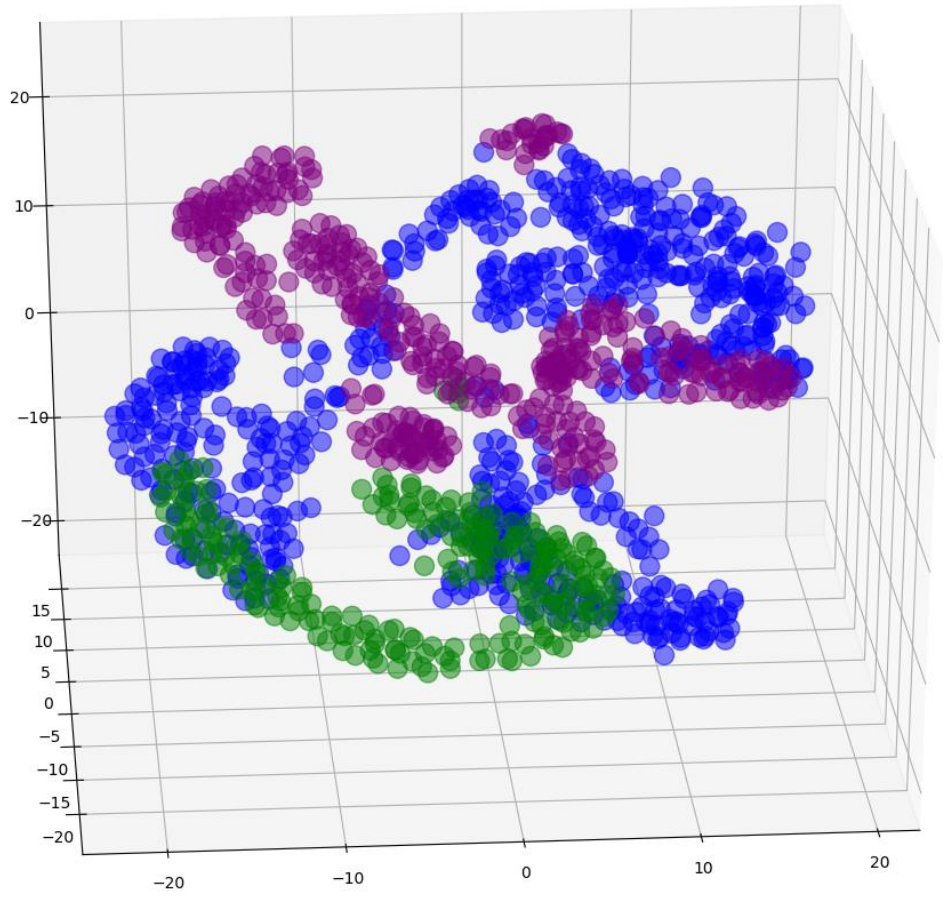
算法简述

GMM 利用高斯概率密度函数来量化事物，将事物分解为若干高斯分布的模型。将观测点数据及的分布，看做多个单一的高斯分布模型进行混合，每一个 component 就是一个聚类的中心。

参数确定

1） n_components: 混合高斯模型个数

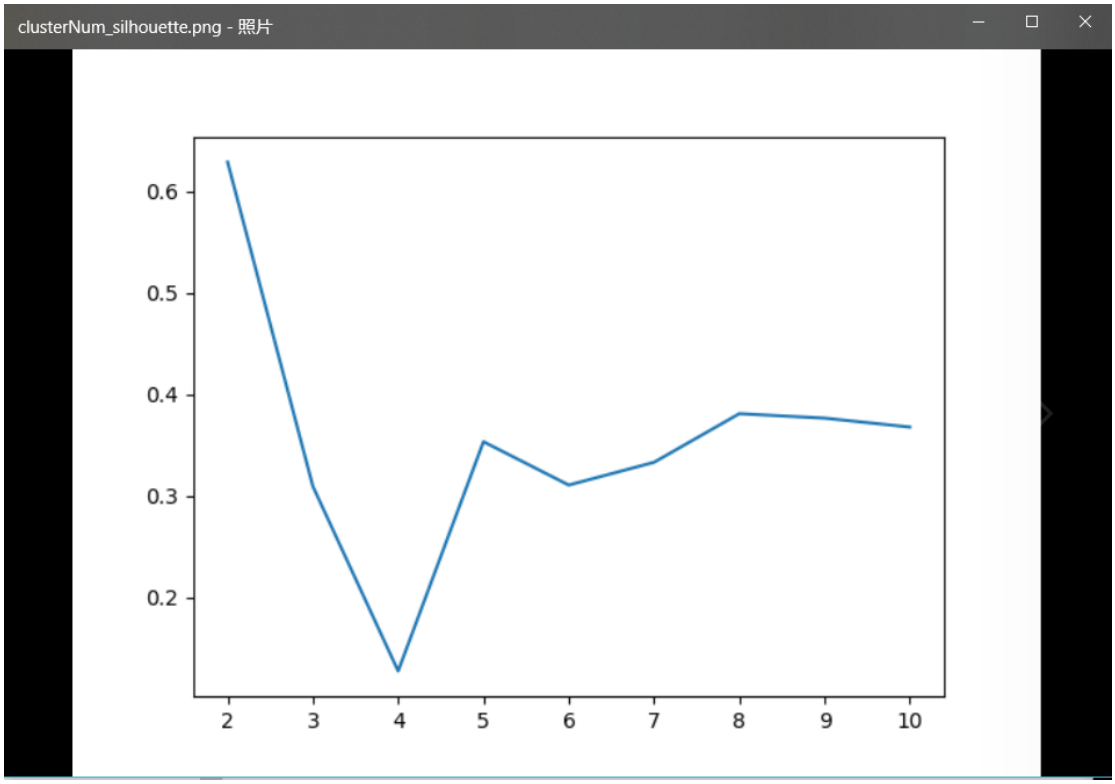


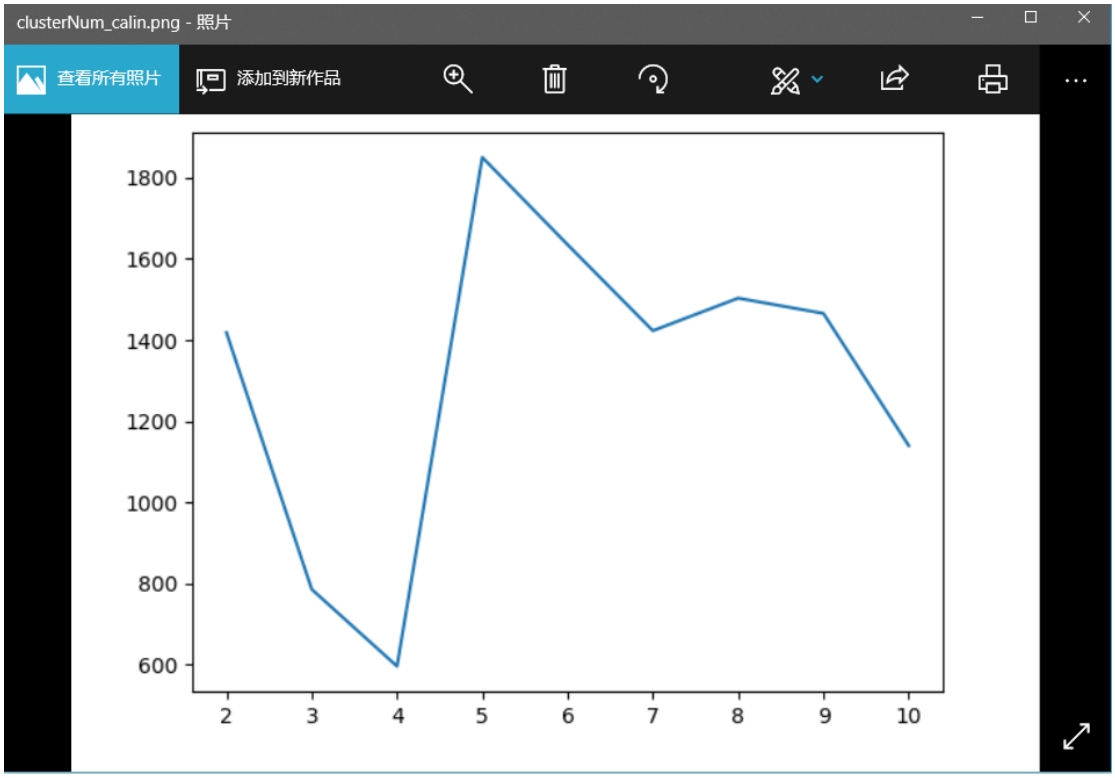



```
0 : 722
1 : 419
2 : 235
s_score: 0.3090956
c_score: 785.823999874373
```

em-gmm 算法的聚类结果与其他聚类方法结果有很大不同，而且从评价指标上看却非常让人意外.

于是,我特意去做了一张与 k-means 确定 K 值一样的图研究.





可以看到,如果把这两张图与 k-means 两张图做一个纵向对比,那么 em-gmm 的聚类结果无论是哪个结果都完全落于下风.

究其原因,我认为这恰恰是 em-gmm 算法的独特之处以及评价指标的局限.相对于其他聚类方法. em-gmm 算法从统计学角度考虑数据,比如它假设数据是多个高斯分布的叠加,它的迭代过程是在不断优化一个极大似然函数.而其他算法相比下,更像是在刻意迎合我选择的那两个评价指标,迭代的目标就是使簇内内聚性更高,簇间相似性更低.

当然结果孰优孰劣,还要从实际情况出发进行分析.

DIY 算法

算法简述

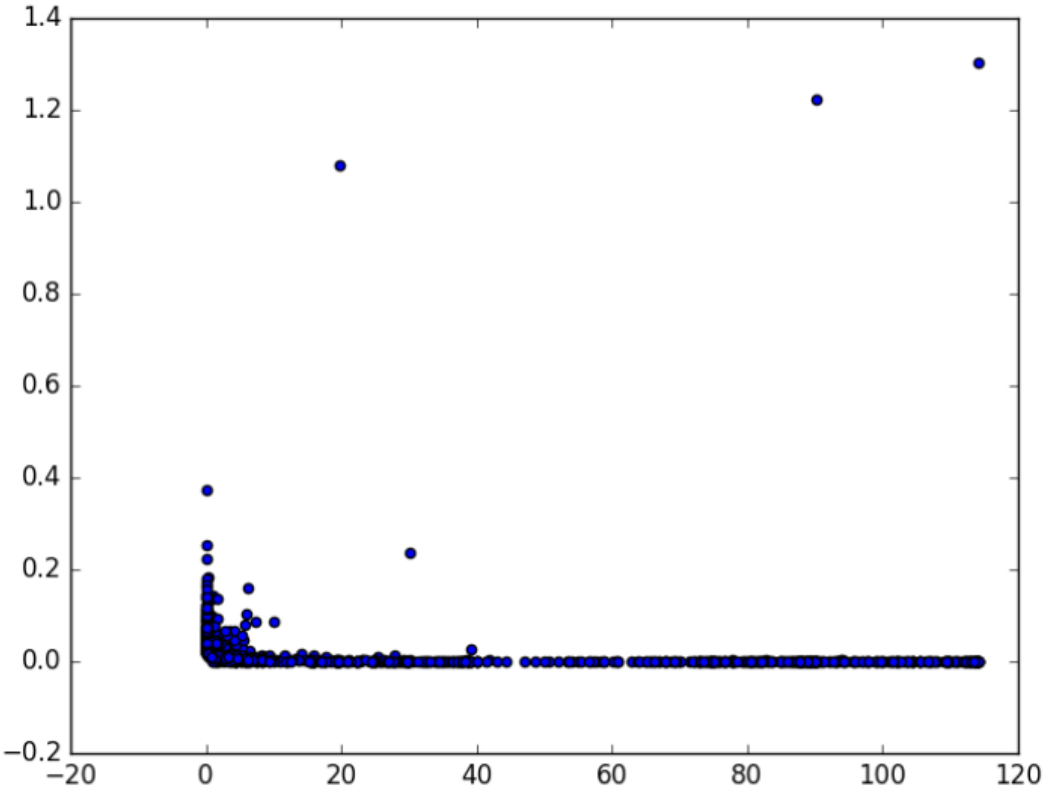
参考文档 1

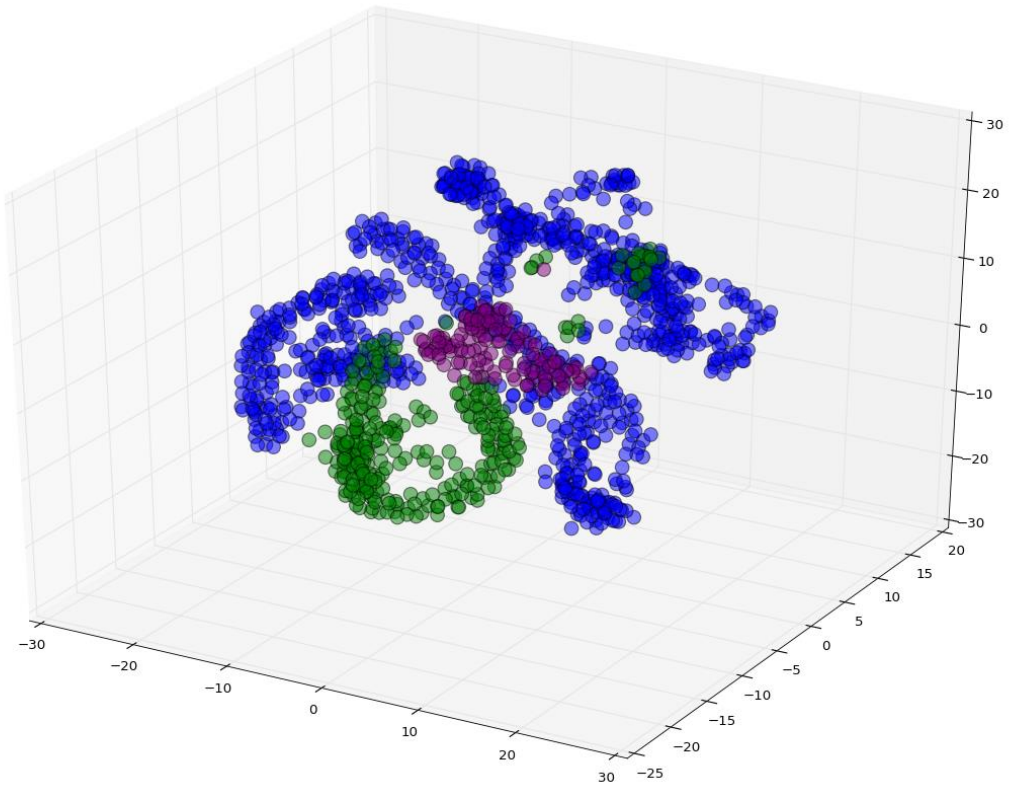
参数确定

使用自己的算法给我最深刻的印象就是一个字,慢. 真实的体会到自己的算法与别人家的算法之间的差距

dc 使用 0.02
n_cluster 沿用之前的 3 个

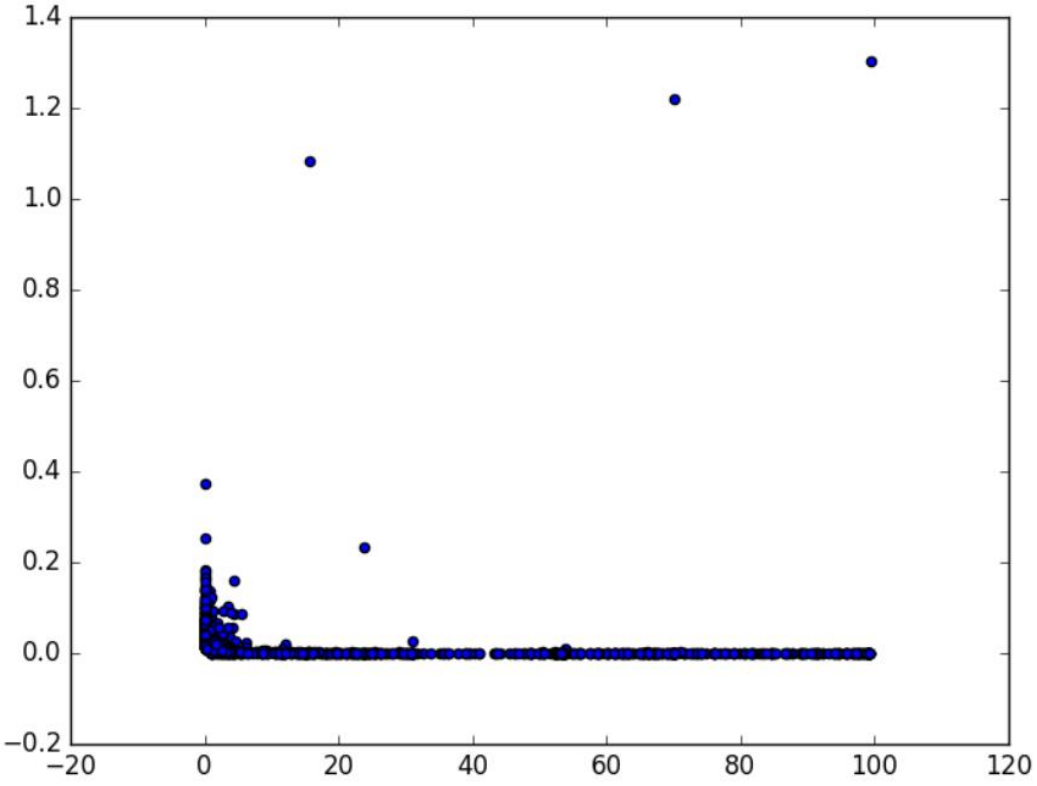
decision tree:

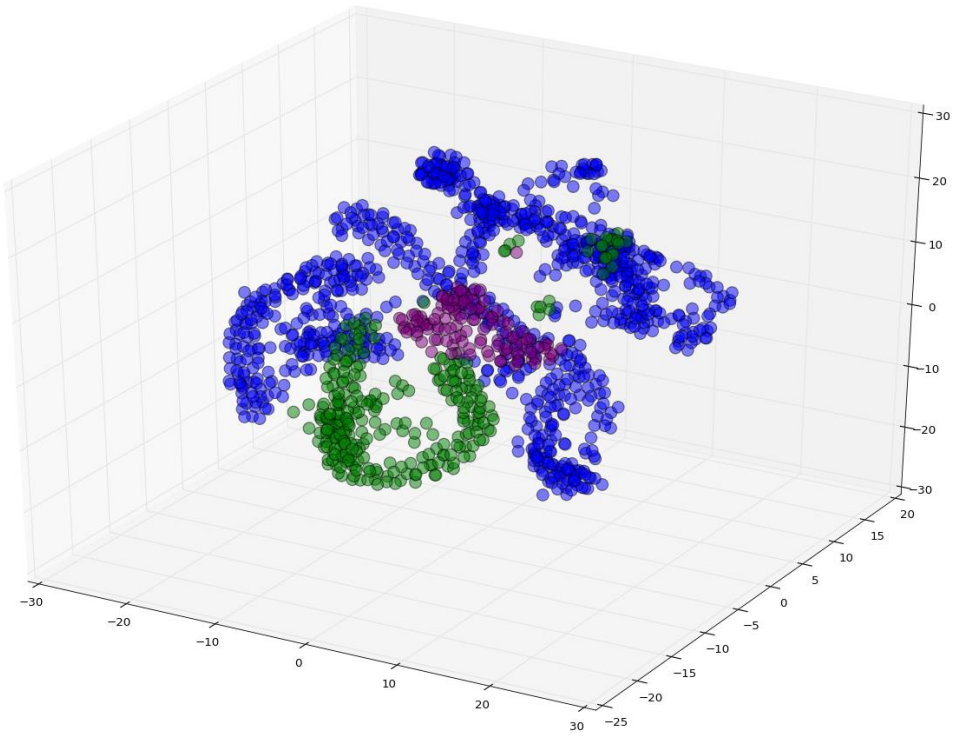

1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−20
0
20
40
60
80
100
120


30
20
10
0
−10
−20
−30
20
15
10
5
0
−5
−10
−15
−20
−25
−30
−20
−10
0
10
20
30

```
C:\Users\lenovo\Desktop\DataMining\p2>python35 cluster.py
dc found: 0.013453916597439443
averageNeighbors: 23.746626109042403
0 : 971
1 : 286
2 : 119
s_score: 0.84203774
```

dc = 0.15






```
0 : 971
1 : 286
2 : 119
s_score: 0.84203774
```

取了几个不同的 dc 对结果没有太大影响,而且根据轮廓系数评价指标,反而 DIY 算法是效果最好的.也算是拿时间换效率了…