Task 2 实验报告文档

姓名：孙浩然

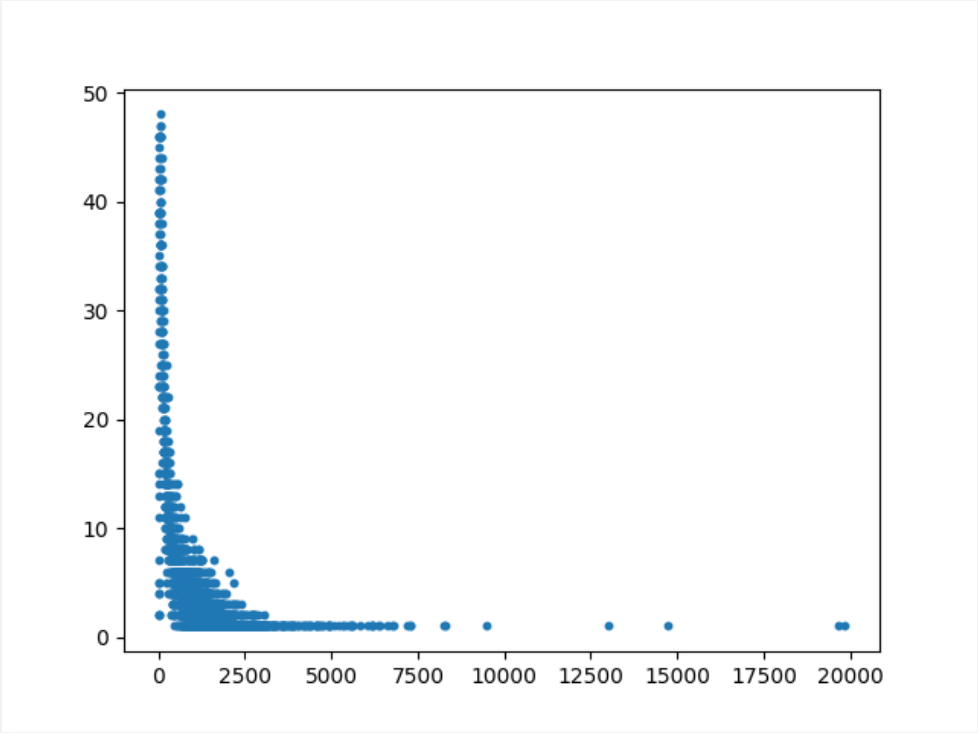
学号：1652714

数据预处理：

相比第一问来说，第二问的数据复杂了多，一万多个用户，每个用户近千词。如果把所有的数据全部用上的话，每个程序都要运行好久好久，在使用 sci2014 的算法对所有数据进行处理时，笔记本发烫了都还没算完 local density。所以我决定在数据预过程中，应该先对所有的数据进行预览，用肉眼观察数据的特点，然后根据预览的结果对数据进行帅选。

预览数据：

首先通过 `preview_data.py` 脚本对数据进行预览。我们首先对数据总体进行预览，下图展示了每个用户使用词汇数量的分布（横坐标：词汇数量；纵坐标：用户数量）：



我们可以从图中看出，大部分用户使用的词汇数量集中在500词左右，只有少数用户使用的词汇数量达到了上千个。

为了进一步了解用户、词汇数量的分布情况，让脚本程序输出以下内容：

```
### 每人词汇使用量情况 ###
平均每人使用词汇 550.2005732878835
每人使用词汇中位数 285.0
每人使用词汇最大值 19852
每人使用词汇最小值 1
```

现在我们对数据有了一个总体的概念，每个人使用的词汇数量差别可以非常大，但是分布总体上还是较为集中的，为下面数据筛选的过程提供了参考。接下来，我们在仔细看一下具体的词汇分布，脚本输出了所有用户使用频率最爱高的五十个词，以下是截取的一小部分：

```
('味道', 223176)
('人', 175304)
('店', 112172)

….··

('牛蛙', 17427)
('结果', 17265)
('服务态度', 16956)
```

我们可以看到，虽然只输出了50个词，但是有的词汇出现的次数已经差了10倍。

筛选数据：

通过了以上的预览，对数据有了大概的了解，接下来我筛选出所有数据中，我认为比较有价值的数据。整个过程分为两个部分，分别为筛选用户和词汇。

筛选用户：

在上文的图中可以看出，用户使用的词汇量虽然差别较大，但是分布总体上还是比较集中的。我认为，如果一个用户使用词汇数量过大，大概率说明该用户进行了刷评论的行为；而如果一个用户使用的词汇数量过小，则可能说明该用户根本没有认真的评价，很有可能是随便输入了一些简单的内容来进行评论的。

所以，在数据预处理过程中，我仅仅保留下了用户使用词汇数量在280 - 800之间的用户，也就是上图中最高峰附近的用户。我认为这些用户更可能认真的进行了评论，而不是随便输入了一些东西。经过这一次筛选过后，还剩下3324个用户。

筛选词汇：

每个用户使用的词汇不仅数目不同而且频率也差别很大。我们从预览脚本输出的内容可以看到，类似 `人` 和 `店` 这样一个字的词，根本看不出用户的任何特征，所以我认为，至少两个字的词才能提供用户的有效信息。同时，由于一共有一万多个用户，我认为出现次数太少的词汇也不会存在实际的意义，而且出现次数太少的词也无法很好地表示两个用户之间的距离，因为每个用户几乎都没用过这个词。

所以在这个过程中，我筛除了一个字的词和出现频率小于1000的词。进行完这个过程后，我输出了剩下的词汇数量，发现还是大的难以接受，如果使用scic2014 算法，基于它的时间和空间复杂度还是很难接受的。

最后，我决定人工从用户使用频率最爱高的五十个词中选取有特点的六个词，作为每个用户的特征词汇，我最终选择的词汇为：

```
selected_words = ['味道', '口感', '服务态度', '服务员', '性价比', '价格']
```

其中 `味道` 和 `口感` 属于用户对商家口味的评论，`服务态度` 和 `服务员` 属于用户对商家服务的评论，最后 `性价比` 和 `价格` 属于用户对商家价格的评论。所以这也为下文的分类数目提供了借鉴。

结果：

经过以上的数据预处理后，处理过的的数据可以在 `./data/parsed_data.json` 中看到，数据的格式如下：

```
# ./data/parsed_data.json
{
  "100157364": { # 每个用户
    "味道": 13, # 用户评论中，该词汇出现的次数
    "口感": 2,
    "服务态度": 1,
    "服务员": 5,
    "性价比": 3,
    "价格": 6
  },

  ......

}
```

用户数据定义：

对原数据进行预处理过后，留下的数据量已经到了可以接受的范围，接下来就需要把这些数据转换成程序中容易进行操作和处理的数据结构了。

用户向量定义：

我们把每个用户的数据定义成一个六维向量，每个维度就是我们选取的六个词汇中的一个出现的次数，如下：

用户ID	味道	口感	服务态度	服务员	性价比	价格
100157364	13	2	1	5	3	6

这样，我们就可以成功把所有的数据转化为Python程序中的一个字典，字典的形式如下：

```
{
  "100157364": [13, 2, 1, 5, 3, 6],

  ......

}
```

用户间距离计算：

在task1中，我就为将 sci2014 算法应用到task2的数据上做好了准备，让距离的计算公式可以计算多维的两点之间的距离。两个用户之间的距离被转化为了两个六维坐标点之间的距离。

```
def calcu_distance(p1, p2):
    """
        计算两个点之间的距离，勾股定理。适应多维坐标点
    """
    temp_sum = 0
    for i in range(len(POINTS[p1])):
        temp_sum += (POINTS[p1][i] - POINTS[p2][i])**2
    return math.sqrt(temp_sum)
```

用户分类数目确定：

和task1不同，task2的数据我确定为了6维向量，显然无法再画出到二维平面上再通过肉眼观看估计了。所幸，我在用户向量定义的时候就考虑到了这个问题，我选取的六个词汇如下：

```
selected_words = ['味道', '口感', '服务态度', '服务员', '性价比', '价格']
```

根据选择的6个词汇，我们把用户分为三类：

1. 更加关注食物口味的
2. 更加关注服务态度的
3. 更加关注商家价格的

实验结果与分析：

在运行完 `task2.py` 程序代码后，我得出了以下表格中的结果：

算法	轮廓系数	分类数目
Sci2014	0.07	3
K-Means	0.29	3
DBSCAN	-0.24	3
Hierarchical	0.44	2
Special Clustering	0.23	3
EM-GMM	0.11	3

所有算法取得的结果都不是很好，取得最好的轮廓系数的算法：Hierarchical，还是将实验结果分为了两类，与期待的三类不相符。其中 sci2014 算法效果很差可以理解，因为是完全是自己手写实现的，肯定和现有的包效果有很大差距。但是为什么其他算法取得的结果也这么差呢？经过思考，我觉得问题可能出在以下几点：

1. 数据预处理不当
2. 用户数据定义不当
3. 算法选择不当
4. 参数设置不合理

其中，前两点可能导致了所有算法的轮廓系数较低，而第三点原因则可能导致了个别算法的轮廓系数非常低。

数据预处理不当：

首先，我们应该确认一个事实：很少有人愿意去认认真真的写评论。大部分的评论可能是无效的，甚至是没有意义的，因为经常有的软件评论送积分。所以我们获得一万两千多的评论数据，有很多可能是没有完全不能反应用户的任何性质。

实际上，有的用户可能看到评论送积分后，顺手就把一个他看到的评论复制粘贴了上去，那么，这条评论不仅不能反映这个用户的特征，还可能反映了其他用户的特征，也就是错误的特征。那么如果这个用户在每次进行评论时都顺手粘贴一条评论，这个数据就是混乱的。

在我前文提到的预处理中，我没有考虑上面的情况，我只是考虑了用户使用的词汇的数量，不能找出那些真正有意义的评论。如果这样的情况是少数的话，这些数据就仅仅是一些噪点，有的聚类算法考虑到了这个情况，会把这些数据点剔除出去，但是如果这类数据很多的话（实际上，我认为这样的数据很多，因为大部分人都懒于去评论），那么就不是噪点那么简单了，这些无效数据占了主要部分，就会让数据的聚类效果非常差。

用户数据定义不当：

在预处理后，我们为了缩小数据的规模，为了让数据可以在可接受的时间内运行出来，我选取了六个词汇：

```
selected_words = ['味道', '口感', '服务态度', '服务员', '性价比', '价格']
```

之后依据六个词汇的出现次数，把每个用户的数据转化为了一个六维向量。

这个做法虽然有效的缩减了数据的规模，但是也损失了大量的潜在有价值的数据，有可能有更加优秀的分类特征就在这些数据中，但是我没能看出来。

算法选择不当：

有些聚类算法的轮廓系数非常低的原因可能是，这个算法就不适用于这种数据集，比如轮廓系数最低的 DBSCAN 算法：

DBSCAN 算法适合于比较集中的数据集，对数据较为分散的数据的聚类效果就比较差，而我们的数据就是较为分散的，因为里面可能有着大量的没有价值的数据，所以导致的 DBSCAN 算法的结果比较差。不过也可以说，正是 DBSCAN 算法的较低的轮廓系数，进一步确定了数据集中存在大量无效点的情况。

有一个轮廓系数很低的算法是 EM-GMM 算法，高斯混合模型聚类算法有一项前提假设：特征独立。但是我们的数据显然无法保证这个假设，一个用户可能具有多个特点，所以导致了该聚类算法的效果不是很好。

参数设置不合理：

最后一个导致算法轮廓系数很低的可能是：参数选择不当。这里我们以 DBSCAN 算法为例。

由于我们并没有完全了解算法的实现，所以对参数的变化进一步对算法整体的影响了解的不够深刻，所以无法合理的调整参数，我调试参数的方法非常暴力，直接两层循环，不断地查看输出的聚类结果，然后选择其中较好的：

```
for i in range(n):
  for j in range(m):
    result = DBSCAN(eps=i, min_samples=m).fit_predict(X)
```

这样调试参数的方法显然不够好，修改的过程一点数学依据都没有，但是由于没有时间去详细的了解算法的具体实现，只能采用如此下策。这种方法有很多明显的缺点：

1. 如果最合理的参数值不在我给定的范围中怎么办？
2. 如果我给的步长太大了怎么办？

由于程序运行时间很长，我不可能把所有可能的值，所以很可能错过了最合理的参数值，从而导致了最终的聚类效果不够好。