背景

这次项目是此前刚进公司的一个练手项目。做完项目之后，通过文档整理、部门内分享等形式的总结，除了发现自己技术上不成熟的地方以外，还看到了一些做事方法上的问题。本文的重点不在于“让我重做一遍的话技术上我会怎么改进”，而是“让我重做一遍的话项目方法上我会怎么改进”。因此下面将按项目的实际历程来复盘、总结，而不是按项目常规流程来写。希望通过这样一个写作的过程，让自己也再反思一遍，避免以后重蹈覆辙吧。此外，由于这段时间有每天写日报的要求，正好能看下自己每次写的“明日计划”到底执行得怎样。

整个项目描述起来并不复杂：从百家号的网页数据中，提取出词汇，并对词汇进行聚类。

项目中还给指定了一种特定的特征向量计算方式（我们一直没想太清楚具体为什么这样做）：先构建词汇共现矩阵C，然后对每个词的特征向量$\overrightarrow{c_i}$，除去与该词汇共现过的词汇的数量，即

$$v_{ij} = \frac{c_{ij}}{count\{j; c_{ij} > 0\}}$$

历程

下面按时间轴来进行回顾，其中“当日完成”与“次日计划”基本都是直接从我的日报中摘下来的，部分做了些精简。另外项目期间有假期等其他情况，也都给跳过了。

Day - 01

> **当日完成**
>
> 完成了初步代码开发，在小批量数据中跑通了；具体分为文章内容提取、tokenize、词汇特征向量计算、词汇聚类这四块；
>
> **次日计划**
>
> 细调代码（更细致的数据清洗、用哪些符号分行、词汇特征向量的计算是否有误、尝试其他特征建模方式、KMeans调参等），并在全量数据集中跑通；

拿到任务的当天，我要到原始网页数据集后就直接开始敲代码了。看起来思路清晰明了：

1. 从网页中（用BeautifulSoup）提取出文章的内容；
2. 把这些文章（拿PaddleNLP中预处理所用的tokenizer）进行tokenize：分短句、分词；
3. 按照任务指定的方式来计算词共现矩阵，进而得到特征向量；
4. 用sklearn中的KMeans对词汇进行聚类。

沿着这个思路，一顿操作猛如虎，一天就开发了一版，在小量数据中跑通了以后感觉信心满满。后面我们会看到，这个方案中其实存在很多bug。

Day - 02

> **当日完成**
>
> 1. 从全量百家号网页数据（125万）中提取出文章内容（56万）作为全量语料；

> 2. 尝试对全量语料进行分句、分词。存在代码执行效率问题，未全部跑出，暂时只用跑出来的约120万短句进行后续流程；
> 3. 基于120万短句，分词后取Top-1w的词汇，基于两词共现次数计算特征向量；发现构建稀疏特征矩阵（1wx1w）时效率低下，且特征维度如果想升至2w时，内存不够用；
> 4. 在前一步的结果中尝试基于sklearn跑KMeans聚类，报MemoryError；
>
> **次日计划**
>
> 1. 考虑优化词汇特征向量（词共现矩阵）的构建效率；
> 2. 考虑如何提升特征维度，需要由1w升至6w；
> 3. Debug看为什么KMeans跑出错；

第二天就翻车了：把小量数据一换成全量数据，整个就垮掉。

容易看出，问题主要在于此前**没有考虑时间、空间复杂度**。

对文章进行tokenize的效率不行。主要的原因在于，当时所采用的方式是**单进程**的：从sys.stdin中**读入一行**数据，然后用tokenizer**加载模型**进行inference。所以至少有几个优化方向：

1. 改为多进程；
2. 调用模型进行inference时，应该增加batch的大小；
3. 应该考虑能否只加载一次模型；

特征工程方面，很快就遇到了内存瓶颈。这其实在接任务时就隐约感觉到了——N维特征向量是不是太大了些?

“存储一个1w*1w的矩阵，需要多大的空间？”这个问题如果是在面试的过程中遇到，我能直接答出应该怎么算；而在实际项目进行中，我很少进行这样的思考与计算，基本都是“现在的内存感觉够不够用？先试试吧？”

仔细想来，**提前对内存开支与执行效率做预估，是一件性价比很高的事情。**因为粗略的估算并花不了太多的时间，而如果存在致命的资源（时间或空间）瓶颈而提前不知道，后面改动整个方案将会耗费大量的精力。

对比当天实际工作跟前一天安排的计划，发现前一天**安排计划时过于自信**，认为在全量数据上跑通并不难。而回过头来想，如果前一天下班前就顺手用全量数据跑一下，问题马上也就暴露出来了。

Day - 03

> **当日完成**
>
> 1. 发现用scipy的稀疏矩阵来构建共现矩阵的做法存在效率瓶颈，转为使用MapReduce之后效率大幅度提升，由数小时降为10分钟内；
>
> **次日计划**
>
> 1. 共现矩阵目前是<token_pair, count>的形式，接下来需要考虑能否用MapReduce直接做KMeans聚类，或者转成稀疏矩阵后再调用sklearn进行处理

发现内存不够以后，想到第一个优化方向是改numpy数组为scipy中的稀疏数组。为此还搜集资料盘点了三种稀疏矩阵的异同。这样一来确实内存是能hold住了，但是构建稀疏矩阵的过程太慢了。然后决定转用MapReduce来计算词共现矩阵——效率大幅度提升。

回过头来看，为什么一开始我没想到用MapReduce呢？主要还是**认知水平上的局限**：

1. 不习惯：我习惯了用sklearn做所有事情；
2. 不熟悉：虽然知道MapReduce的原理以及大概的编写方式，但此前几乎就没有自己写过MR脚本，**潜意识中给自己设限了**。

对比实际工作跟前一天安排的计划，整体上是沿着计划在走的：优化词共现矩阵的构建效率。只是光这一点就花去了一天时间，而没来得及做提高特征维度以及debug聚类模型的事情。

Day - 04

> **当日完成**
>
> 1. 将昨天的MapReduce改了一下，改成按<token, count_vector>的形式，并跑通；
> 2. 跑出了一个更大的短句语料（共3555万行，未用全部文章语料），并得到107w*107w的词汇共现矩阵；
> 3. 取Top 6w的词汇，分别输出了利用不同类别数的KMeans算法结果（10，50，100，300，500，700，1000）；
>
> **次日计划**
>
> 1. 人工评估不同K值输出的聚类效果，并思考优化方向；（目前程序取Top 10w的词汇，K取1000，2000，3000，4000，5000在执行中）

注意到，当天还换了MR程序的输出格式，这说明我此前并**没有想清楚该模块跟后续模块的数据对接方式**。更进一步的，**为什么不提前做好系统设计再开发呢?**

另外，一切似乎过于顺利？107万*107万的词共现矩阵，因为MR是神器就那么轻松的跑出来了？6万*6万的数据量，那么快就跑出了7种结果，真的只是因为KMeans的高效?

Day - 05

> **当日完成**
>
> 1. 今天发现了一个重要的代码bug（编码问题导致之前MapReduce计算词汇共现是错误的），已处理；
> 2. 改了bug之后，数据量变正常了，也大了许多（取Top 6w的话，共现词对有5千万），因此针对执行效率做了些代码优化；目前已经得到Top 6w的词汇特征矩阵；但用sklearn的KMeans做聚类时，发现执行效率特别低；目前安排了K=500的正在跑；
> 3. 尝试开发了一版基于MapReduce计算KMeans，但未调试；
>
> **次日计划**
>
> 1. 了解有没有办法优化sklearn的执行效率
> 2. 调通MapReduce版的KMeans；

进展过于顺利的时候似乎需要更多的注意。

代码有bug是不可避免的，而这里暴露的主要问题是，此前虽然跑出了一个共现矩阵的结果，但**没有做数据校验**。只要看下6万*6万的矩阵中有多少非空元素，不需要太多的常识就能发现矩阵过于稀疏了。所以**在跑数之前就应该对怎么验数有一定的思考**。

说来惭愧，数据校验这个意识，最早在两年前就在一个项目中被领导提醒过。

另外，由于此前用MR计算词共现矩阵发现“真香”，于是看到sklearn的KMeans跑得慢的时候，也想自己动手来个MR版的KMeans。

这一天由于代码的bug，事情的进展自然跟计划的完全不一样了。

Day - 06

> **当日完成**
>
> 1. 多次尝试优化sklearn的KMeans聚类算法的执行效率，一直没有提升；收集资料后决定换为sklearn的MiniBatchKMeans，效率上来了，但效果（仅从轮廓系数 silhouette score来看）可能略差一些；目前已有N=2w，K=1000的结果，正在跑K=1500和K=2000的结果；
>
> 【避坑】
>
> 1. 使用sklearn的模型进行训练时经常设置n_jobs=-1，这在python=2.7.3的时候可能会报错，改用python>=2.7.17能避免；
> 2. 对python脚本输出重定向时，需要及时用sys.stdout.flush() 清缓存，否则日志可能要等很久才看的到；
> 3. 多查看博客，参考别人的做法；会更快发现MiniBatchKMeans可能更好用；
> 4. 要及时评估当前算法执行效率、节省时间；而不是丢在一边干等待；
>
> **次日计划**
>
> 1. 人为评估聚类效果
> 2. 数据清洗（统一大小写、标点符号过滤）
> 3. 增大数据量（N从2w往上增加）
> 4. 仔细阅读MiniBatchKMeans文档，进行调参（K值、迭代次数等其他参数）；

现在想来，还好当时发现了sklearn中还有个MiniBatchKMeans的“神器”，不然如果朝着MR版的KMeans折腾下去不知道还要走多少弯路。KMeans算法虽然简单易懂，但是要造个MR版的轮子，可一点都不简单——我在看了KMeans的源代码后才发现其中有那么多的门道。

正如日报中所说的，需要吸取的经验是：多看博客、收集资料，参考别人的做法；以及**执行脚本前都应该预估执行效率**。

改用MiniBatchKMeans后，效率不再是主要问题，也才终于可以开始做优化了。

Day - 07

> **当日完成**
>
> 1. 主要花时间在数据清洗上，通过修改PaddleNLP中给出的tokenizer.py，统一成小写、统一标点符号、去除标点符号。并修改成pipe的方式，进而实现多CPU并行inference的效果；
> 2. 通读了sklearn中MiniBatchKMeans的源代码；
>
> **次日计划**
>
> 1. 进行聚类效果评估；

通读sklearn中MiniBatchKMeans的源代码的过程中，我才发现自己此前对KMeans类算法的理解有多肤浅。虽说官方文档写得已经很详细了，但如果不读源代码，还是不会懂：同样一个n_init参数在KMeans和MiniBatchKMeans中有什么区别？tol和reassignment_ratio到底是怎么影响最终收敛判断的？对每个batch迭代计算时又是怎么更新类标签的？

诸如此类的关键细节，在教科书上基本是不会看到的，只藏在代码中。**学算法还是得多看源码**。

Day - 08

> **当日完成**
>
> 1. 利用昨天清洗后的数据，重新出了聚类结果；
>
> **次日计划**
>
> 1. 进行聚类效果评估；
> 2. 抽空整理代码、实验数据等，准备分享；

Day - 09

> **当日完成**
>
> 1. 抽样观察聚类结果；
> 2. 整理代码，拟了一份初步的总结文档；
>
> **次日计划**
>
> 1. 整理文档；

Day - 10

> **当日完成**
>
> 今天主要是挑了一个具体的结果进行了较为细致的分析与评估，初步的结论是：当前的聚类算法方案，聚出来的类中，词汇量2~5的类别效果可接受，词汇量为1以及词汇量>=200的类效果不如预期；
>
> **次日计划**
>
> 讨论如何做优化等进一步问题；

接着三天主要是抽样了一些聚类的结果，进行人工评估，并开始写文档。

Day - 11

> **当日完成**
>
> 上午讨论发现聚类效果随K值的影响波动特别大，开始时怀疑是n_init=1导致，改为n_init=10后问题依旧存在；仔细研读MiniBatchKMeans库的源代码后，发现原因可能在于init_size=6000太小，改为init_size=20000后问题并未解决；目前怀疑效果波动是由于初始化时采用随机分配的方式所导致的，已经改init=random为init=k-means++正在跑，明天看效果；
>
> **次日计划**
>
> 基于现有结果做数据分析，寻找效果波动大的原因；

跟指导人讨论了以后，发现聚类效果随着K值的波动特别大。

这一点为什么此前三天的人工评估里面没有发现呢？回想起来，主要是因为当时**没有明确实验目标以及实验计划**。我只是想着先出一份结果看看，而一直没有去想，做出来以后该**如何评估聚类模型的效果**。只有对评估方案有较为清晰的定义，才能谈迭代优化。所以我才一直没有想到怎样去评估不同K值下带来的效果差异，以及如何挑选K值才是最为合适的。

Day - 12

> **当日完成**
>
> 主要是整理了此前做过的多次实验（共计68次），进行了汇总，尝试从中分析K值的选择；初步看来K值取1000较为合适；除了 Calinski-Harabasz Index 这个指标，目前正在用Silhouette Score对之前的结果跑数，看了下其中几个结果，出现了负值，需要进一步分析；
>
> **次日计划**
>
> 1. 梳理清楚两个指标的差异，明确为什么Silhouette Score会出现一些负值的现象；
> 2. 挑选Calinski-Harabasz Index较高以及较低的两份结果，人为对比分析，进而判断该指标是否确实适用；

既然发现了自己的问题，也就找到了方向——K值取多少较为合适。

聚类算法的效果评估本身就是个领域里的大问题。在搜集了一些资料后，摆在面前的主要有三种量化指标：inertia、Silhouette Score、Calinski-Harabasz Index。这里不展开谈三种指标的区别。而且坦白讲，当时我并没有仔细的去看这三种指标的区别！**在没有完全理解指标的含义时就直接使用，是风险很大的事情**。一旦画出来的曲线“长得奇怪一些”，也就懵圈了。

Day - 13

> **当日完成**
>
> 1. 梳理清楚inertia、Calinski-Harabasz Index、Silhouette Score的区别；
> 2. 在回查日志时发现，MiniBatchKMeans在每次迭代时内部inertia波动很大，搜集资料、并思考后发现，对数据进行normalize之后（即除去特征向量的模），整体效果更为稳定，可以看到内部迭代时inertia是稳定下降的；且随着K值的增大，整体inertia稳定下降，符合预期；
>
> **次日计划**
>
> 1. 对于新的结果，再次看了下Silhouette Score，仍旧都是负值，只是由原来的-0.5、-0.3调整到了-0.07、-0.08的量级，整体有效果提升；但仍需要定位原因、排查问题；

在盘点清楚三种指标的区别之后，看曲线的时候也才能看出规律及异常来。同时，inertia曲线还启发了我在特征工程上的优化灵感。到这个时候，也就对“如何挑选K值”这个问题有了进一步的理解，后面的分析自然水到渠成了。

Day - 14

> **当日完成**
>
> 1. 今天重新整理了实验数据，基于正则化前和正则化后做数据对比分析，观察簇内误差平方和（inertia）随K值的波动情况，挑选拐点k=900、k=2000、k=5000进行case分析；
> 2. 分析后整理了至今的一些结论：
>
> 聚类算法调优方面：
>
> 1. MiniBatchKMeans 的执行效率显著高于 KMeans；
> 2. 选择 k-means++ 的初始化方式，一般会优于 random 的，但是效率会慢很多；
> 3. 提前对特征进行正则化很重要，能让聚类效果更稳定、良好，同时效率也会提升；

4. MiniBatchKMeans 在使用时，可以尽量调高 init_size 与n_init，使效果更稳定；
5. 挑选效果指标时应首选 inertia，因为该指标在 KMeans 训练完之后就能得到，不需要额外计算，其次再考虑其他指标，其中轮廓系数计算起来特别慢；
6. 挑选K值时，可以绘制 inertia 随着K值的变动情况，再寻找关键的拐点，挑选合适的K值；

聚类效果方面：

1. K值如果太大，会孤立出许多词汇作为单独的簇，例如K=5000时，会有40%的词汇被孤立，这显然不符合业务预期；
2. 聚类出来的小簇（含有词汇量10个以内）通常效果较好，而K值越大，出来的簇普遍就小一些；相对于k=900而言，k=2000输出的小簇质量相当而数量更多，所以表现更好一些；
3. 对于中等大小的簇（含有10~100个词汇），关键是需要有个更明确的业务评估标准，才能进行评判；
4. 超大的簇应该尽量少，因为通常而言，这种簇难以在业务中应用；

次日计划

1. 整理文档；

到这里项目基本就算结束了，出来的结论也算是靠谱。

总结

总结来看，经过这次的项目，主要有这么一些体会（非技术层面）：

1. 开始做事前，要先搞清楚目标，最好是能有个量化的目标；
2. 开始敲代码之前，要先做系统设计：主要分哪些模块？模块间流转的数据是怎样的？方案的时间、空间复杂度如何?
3. 考虑方案前，多搜集资料，参考别人的做法；
4. 先定实验方案，再跑程序：实验目的是什么？有哪些假设？要做多少组实验?
5. 开始跑程序之前，最好先估计执行时间；
6. 在数据结果出来之后，自己先多做点数据校验；
7. 日常学习算法时，还是得多看开源代码；