实体识别及关系抽取体验报告

李易非 熊苗 林新迪

一、项目要求

对上市公司所发布的公告进行处理，运行一定的模型与规则，根据需求，自动抽取结构化数据。

二、数据清洗

利用BeautifulSoup遍历增减持、重大合同、定向增发三种披露信息的html文件，分别抽取html文件的文本与表格信息，并进行必要的格式转换，如全角转半角、数字单位归一化等。

1、HTML表格抽取

在遍历html文件，当遍历至 `<table>` 标签时，调用 `to_matrix(souptable)` ,将表格内容转换为 `pandas.DataFrame`，转换csv格式并保存至相应路径。

1.1 表格抽取要点

rowspan与rowspan：

html文件中有的表格中存在合并单元格的情况，及某个内容会跨越多行或多列，为解决该问题。我们先计算表格的大小（shape），建立和表格大小的DataFrame，再遍历表格内容填入对应的DataFrame的位置。

跨越多行

在pdf文件中，有的内容看似是一行，但是实际上再html中是占用了两行的空间，即该 `<tr></tr>` 中每个 `<td></td>` 的colspan属性都大于等于2。针对这种情况，如果直接计算一个表格中 `tr` 标签的个数会使得之后填充超过索引，因此遍历时用每个 `td` 中最小的最小的colspan代表该 `td` 所占的行数

表格空缺

表格的某一行的前几个空可能会空缺，但是在html文件中这部分空缺并没有提示，因此需要先计算一个 `td` 中 `tr` 的数目，若数目小于表格的列数，则先跳过前几个空格

hidden的跨页合并

存在某些跨页表格在html中分为两个table并且两个table之间有一个 `hidden` 标签。为了将两个表格合并，当遍历至 `hidden` 时，检查其 `next_sibling` 和 `previous_sibling`，若都为table标签，则将两个表格合并。

表格中有br标签

在某些html中表格中有br标签，是否有br标签在pdf看不出其影响，但是在抽取表格内容时，该 `td` 的字符串就为None，因此需要取 `td` 中的最后一个content作为内容进行填充。

img处理

在某些type为content的`div`中会有`img`标签代表表格中的图片，遍历时舍弃`img`标签的内容。

由于hmtl文件是用pdf文件格式转换而成，我们利用遍历表格的每一个标签计算表格大小。将所有`tr`中最小的colspan相加作为表格的行数。计算每个`tr`中所有`td`的colspan相加作为改行所占的列数，取最大的所`tr`占列数作为表格的列数。

抽取表格时，先计算该表格的大小，建立一个全为0的相应大小的DataFrame，cursorx和cursory记录遍历table时对应的DataFrame中的位置。遍历表格的每一个`td`标签，读取标签的属性，将其填入cursorx，cursory所致命的的位置。在遍历过程要注意，1）表头有可能会占两行，所以下一个tr其实是从第三行开始填；2）所填内容的位置应当随`td`的colspan和rowspan做相应的调整c3）填DataFrame的某一行时，该行有的位置由于前几行的colspan大于1已经有内容填入，需要跳过。

html遍历

函数采用递归方式遍历html，`traverse_html(string,soup,newdir_path,table_num)`中，string是在递归中需要传递html中的文本信息，soup就是读入html文件时生成的BeautifulSoup类的实体，由于函数中包含存储table为文件的操作，因此需要newdir_path作为该table应当存储的文件夹路径，table_num记录读到第几个table，用于存储table时的文件命名。递归时，判断soup的类型时，根据不同的类型进行不同的操作，最终所有的文本信息（包括title）都会在string中，而读到的表格将会直接写入相应路径的文件夹

2、数据格式转换

在上市公司的公告中充斥着大量非格式化的日期、金额等，将他们预先处理为标准格式的数据有利于我们在后续的文本信息抽取与表格信息抽取中抽取更多的有效信息。总的来说，我们进行了如下几个步骤进行处理：

1. 将文本中的英文转化为小写字母；
2. 去除文本中的空格以及空行（文本句子以中文句号作为划分）；
3. 将全角字符转化为半角字符；
4. 去除数字中的逗号，如 *100,000 -> 100000*
5. 将文本中的日期转化为标准格式，如 *二零一六年十月十二日 -> 2016-10-12*，*2016年12月10日 -> 2016-12-10*
6. 将文本中的数字单位全部转换为元，如 *100万元 -> 1000000元*，*一百万元 -> 1000000元*

   以上所有处理流程封装在 `FDDC.html_process.Transformer` 中，主要思路均为正则匹配加转换，不做赘述。

二、文本信息处理

1、overview

本次项目中，三种类型的文件大体上都可以分为文本信息抽取与表格信息抽取，本部分将着重于文本信息的完整处理流程。总体上，文本信息的处理依靠一个 lstm + crf 的实体标注神经网络标注文本中的实体，再利用正则模版去匹配实体出现的模式，从而提取有效信息。在细节上，三种类型的文件提取略有不同，我们将在后续部分中详述。神经网络需要优质的数据集才能表现出更加优秀的效果，我们在标注数据集上投入了不少的精力与时间，最后也获得了一个泛化能力不错的实体标注神经网络。

2、实体标注神经网络

数据集标注

总的来说，数据集标注遵循以下原则：

1. 尽量标注所有的 field，省去人工正则规则找实体的繁琐；
2. 句子级别标注；
3. 句子中必须有主键之一，加其他属性；

在对三种类型的文本进行一定的抽样调查后，针对每种文件进行特异化的标注：

- 增减持：增减持中包含 `股东简称` 这一属性（非主键），但一旦说明简称后，后续大量有效信息句子都以简称作为主语，所以在增减持标注中，我们加入了 *股东简称* + *其他属性* 的句子进入训练集；
- 合同：合同中包含 `联合体` 这一属性，如果有多个联合体需要则以顿号分离，一开始我们没有注意到顿号分离，直接按照联合体的一整个字段进行标注，导致联合体识别异常不准，重新审查数据集后发现了这一现象，更改后，联合体识别显著提升；
- 定增：在定增标注中，我们并没有标注 `认购方式`, `锁定期` 这两个属性，在我们的 data exploartion 中发现，锁定期、认购方式在整个数据集上只有不超过10类，可以通过正则直接提取；

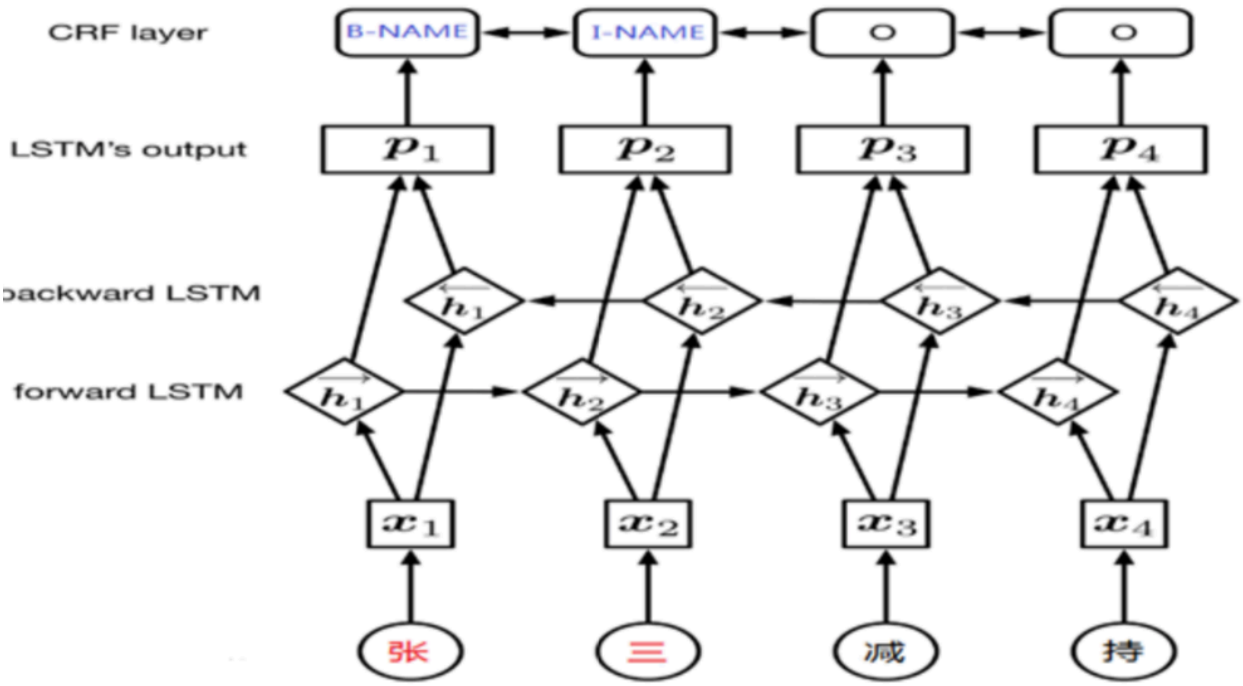
  所有标注代码在 `FDDC.data_annotation` 中，具体思路均为按字段在文中搜索进行标注，只是有很多细节需要处理，不再赘述。

绿色为标记的实体(Green are annotation entity)

公告类型	主键	第1列	第2列	第3列	第4列	第5列	第6列	第7列	第8列
股东增减持	1-2-4	公告id	股东全称	股东简称	变动截止日期	变动价格	变动数量	变动后持股数	变动后持股比例
重大合同	1-2-3	公告id	甲方	乙方	项目名称	合同名称	合同金额上限	合同金额下限	联合体成员
资产重组	1-2-3	公告id	交易标的	标的公司	交易对方	交易标的作价	评估方法		
定向增发	1-2	公告id	增发对象	增发数量	增发金额	锁定期	认购方式		

网络结构

网络的结构我们采用了 Chinese Information Extraction 的开源实现，该模型接受字向量，经过 lstm 层，输入 crf 层，最终输出实体标注信息。



BiLSTM-CRF

training settings

考虑到本次项目时间有限与我们的机器配置有限，我们均采取了 Chinese Information Extraction 的默认参数，但都将 epoch 设为 20，防止神经网络在训练集上的 overfitting，其余参数如下：

- lstm_dim: 100, lstm 层维度大小
- batch_size: 20.0
- emb_file: 字向量文件，from zjy-ucas/ChineseNER
- clip : 5.0，梯度裁剪（防止梯度爆炸）
- dropout_keep: 0.5, drop out 层参数，进一步增强网络泛化能力
- optimizer : "adam", adam optimizer, 多快好省
- lr: 0.001, 学习率
- max epoch: 20

3、文本信息抽取

这一部分中，我们已经拿到了神经网络实体标注的结果，不同于我们的标注数据集，输出结果不一定是主键+其他属性，且也有很多的错误标注，所以我们需要进行一定的实体约束剔除噪声句子标注，再进行关系提取，不同句子可能对同一记录有所贡献，最后再进行提取 record 的 merge, 得到最终结果，这一部分的代码在 `FDDC.content_extraction` 中。record merge 的大体思路为根据主键分组，并进行信息合并，不再赘述，下面将详述前两个过程：实体约束、实体组合。

```
@abc.abstractmethod
def constraint_policy(self, sentence_dict):
    """
    return True if the sentence dict is valid, False otherwise
    :param sentence_dict:
    :return:
```

```
    """

@abc.abstractmethod
def merge_records(self, records):
    """
    merge the found records
    :param records:
    :return:
    """
@abc.abstractmethod
def extract_records_from_sentence_dict(self, sentence_dict):
    """
    just as it's name indicates
    :param sentence_dict:
    :return:
    """
```

3.1 增减持

实体约束策略（代码中为 abstract method: `constraint_policy`）：增减持的实体约束策略与实体标注策略类似，通过下面的约束之后，我们留下了包含有效信息的句子：

- name + (short_name) + end_date + other attribute
- short_name + end_date + other attribute
- 若 name 与 short_name 在同一个句子中，short name 必须出现在 name 之后

实体组合策略（关系抽取，代码中为 abstract method: `extract_records_from_sentence_dict`）：在经过实体约束的句子中将实体标注序列转化为字符串，如 *name end_date quantity -> NEQ*，再利用我们定义的 关系 pattern 进行正则搜索，抽取出有效 record:

- `E?NS?[DC]{1,2}` 变动日期?，股东名称，股东简称?，持有股份数，持有股份比例
- `NS?E?[DC]{1,2}` 股东名称，股东简称?，变动日期?，持有股份数，持有股份比例
- `E?S[DC]{1,2}` 变动日期?，股东简称，持有股份数，持有股份比例
- `SE?[DC]{1,2}` 股东简称，变动日期?，持有股份数，持有股份比例
- `E?NS?[QP]{1,2}` 变动日期?，股东名称，股东简称?，变动数量，变动价格
- `NS?E?[QP]{1,2}` 股东名称，股东简称?，变动日期?，变动数量，变动价格
- `E?S[QP]{1,2}` 变动日期?，股东简称，变动数量，变动价格
- `SE?[QP]{1,2}` 股东简称，变动日期?，变动数量，变动价格

记录 merge 策略 (代码中为 abstract method: `merge_records`)：

3.2 定增

实体约束策略（代码中为 abstract method: `constraint_policy`）：定增的实体约束策略与实体标注策略类似，通过下面的约束之后，我们留下了包含有效信息的句子：

- receiver + quantity / price: 股东名称 + 变动名称 / 价格

  实体组合策略（关系抽取，代码中为 abstract method: `extract_records_from_sentence_dict`）：在经过实体约束的句子中将实体标注序列转化为字符串，如 *receiver quantity price -> RPQ*，再利用我们定义的 关系 pattern 进行正则搜索，抽取出有效 record:

- `RPQ?`: 股东，增持价格，增持数量?
- `RQP?`: 股东，增持数量，增持价格?
- `PRQ?`: 增持价格，股东，增持数量?（以200.2元向李易非增发500股）

重大合同

实体约束策略（代码中为 abstract method: `constraint_policy`）：定增的实体约束策略与实体标注策略类似，通过下面的约束之后，我们留下了包含有效信息的句子：

- 乙方 + 其他两个属性
- 一个句子除了联合体外，其余实体至多一个

  实体组合策略（关系抽取，代码中为 abstract method: `extract_records_from_sentence_dict`）：不同于前两个类型的实体组合，我们发现合同的有效语义信息是结构最为复杂的，仅仅依靠实体约束我们就可以获得不错的结果，过度约束反而影响最终表现

  entity count

再者，我们发现某些文件在实体约束之后没有任何有效记录，抽样调查这样的文件后发现，大多都只包含 *1* 个 记录，且篇幅较短，实体分散在各个句子中，针对这种情况，我们提出了 `entity count` 的策略，顾名思义，就是计算不同实体标注的数量，选取 count 最大的值作为实体的提取信息，比如在乙方标注中，文章中包含 {李易非：2个，熊苗：1个}，那么乙方标注就采用李易非，令人惊喜的是，这样简单的策略足以 handle 这种特殊情况，提高了我们的最终表现。

4、总结

本部分中我们详述了本次项目中文本抽取的全过程，总体上，文本信息的处理依靠一个 lstm + crf 的实体标注神经网络标注文本中的实体，再利用正则模版去匹配实体出现的模式，从而提取有效信息。尽管神经网络的输出包含大量的有效信息，但是仍有不少的噪声，我们通过实体约束、实体组合、记录组合、entity count 等方法剔除噪声，抽取有效信息，最终取得了令人满意的结果。

三、表格处理

整个表格处理的流程如下：

- 接受 htmlParser 传给我的单次table处理调用
  - 每篇文章通过 beautifulsoup抽取出了table，只删除空白符，剩下的清洗是在抽取出

record，在convert 或者record的normalize里面处理

- 从html读进来一篇文章，所有段落先调用 parse_content，并读取 pickle 文件获得神经网络判断的实体，利用规则 parse 出相应信息，把出现的所有公司和他的简称记录在对应dict里
- 再对每个table调用parse_table，通过表头进行模糊抽取，得到相应的record信息
  - 和dict匹配公司全名
  - 返回之后和文本抽取出的 record 信息merge
  - 返回record_list
  - to_result 输出
- 输出 record_list = [[record class]]

1、 增减持

文件结构

HtmlParser

```
def parse_content(self, html_file_path)
def parse_table(self, html_file_path)
@staticmethod
def parse_table_to_2d_dict(table)
```

`parse_content`读取文件，通过 beautifulsoup 抽取出所有的 content 标签的内容，并返回一个数组，没个数组代表一个 paragraph， 每个 paragraph 也是一个数组，由 该 paragraph 里面的 subParagraph 构成。

`parse_table` 读取文件，通过 beautifulsoup 抽取出所有的 table 标签的内容，并发挥一个数组，其中每个数组是一个 dict，代表该文件中的一个表格。

`parse_table_to_2d_dict` 是一个静态方法，类不需要实例化就可以调用，被`parse_table` 调用来处理每个table，返回一个二维数组，记录表格里每个 entry 的内容。

其中 parse_table 会将表格中文本的所有空白符号都去除，并且处理了 rowspan 和 colspan 现象。

Extractor

extractor 主要用来抽取表格和文本的重要信息。我们针对不同的文本文件设置了不同的类，如 Dingzeng_extractor 和 Zengjianchi_extractor 。我们重点以增减持讲述表格处理的抽取框架。

我们重点介绍一下我们的正则表达式 pattern。在这一点上我们吸取了 Github dmjvictory 对于模式的处理，该作者利用一个 Json 文件来存储所需要的 识别模式，并利用 TablePattern 这个类来进行存储和执行相应的操作，这样有利于正则和代码操作分离，更利于后期对于 匹配模式 的修改。

```
def extractTable(self, html_file_path, html_id, sentences_dict)
```

这个函数主要用来解析对应文件的 table，并返回 rs = [record1, record2,]这样一个list。

record 是一个类，记录从表格里解析出来的每一条记录。

首先利用 `pickle_test.build_short_name_ref_table(sentences_dict)` 这个函数。该函数是一个和神经网路输出的 pickle 文件对应的接口，我们从该文件里吸取BiLSTM神经网络识别出来的名字，并得到对应的 公司全称-简称对应dict。

再对 HtmlParser 分析出来的每个表进行逐个记录抽取，由于简称文件的特殊性，我们暂时对每个文件的第一个表作为基础，后面的表的信息用来对第一个表进行辅助。而增减持的处理和定增有所不同。

```
def extract_from_table_dict(self, table_dict, html_id)
```

该函数用来对单个表格的每一行进行抽取，首先是表头匹配，只对匹配好的行进行逐行转换成 record。

对表头的每个 column，我们都和 config 文件里的 pattern 进行匹配，如果匹配上则存进 field_col_dict。同时我们也会逐行扫描，如果每一行的entry 中出现了我们不希望看见的字词，我们将会把这一行排除掉。

同时我们在这个过程中也执行相应的单位转换，比如表头中识别出来 （万股）的万，我们会逐行将"万"加入到每一行的内容中，以待之后对每个 record 进行单位处理。

同时我们默认如果一个表只抽出来一个主键或属性，将不对该表格进行处理。

Record

```
def __init__(self, html_id, shareholderFullName, shareholderShortName,
finishDate, sharePrice, shareNum, shareNumAfterChg, sharePcntAfterChg)
```

以上传入的参数都是一条 record 所需要记录的属性。

```
def normalize(self, id2date_dict)
```

Normalize 主要用来对输出数据的格式进行修改，如将时间等修改为 2018-05-03 这样的格式。

```
def normalize_finish_date(self, text, id2date_dict)
```

这个函数主要用来对日期进行修改，先用正则匹配出相应的 年月日，然后根据格式输入，如果 日期的天数缺失，我们将会调用传入的 id2data_dict 来进行补充，如果月份和公告发布日期相同，则填入公告发布日期，如果不同则按照奇偶月数和是否闰年来对日期进行补充。

TablePattern

```
def __init__(self, field_name, convert_method, pattern, col_skip_pattern,
row_skip_pattern, unit_pattern)
```

这个类主要用于正则匹配表头，其中 field_name 标记了这个类的类型，属于哪个表头的匹配模型，convert method 则显示表明了应该用什么样的方法去转换它的格式，pattern 则给出了 匹配的关键字词，用于python 的re.compile 的环节，col_skip_pattern, row_skip_pattern声明了行列中不应该出现的关键字，用于排除，unit_pattern 记录了需要转换的单位，如"万|亿"等。

模式匹配

- 股东全称

```
{
  "fieldName": "shareholderFullName",
 "convertMethod": "getStringFromText",
 "pattern": "姓名|名称|对象|主体|增持人|减持人|参与人",
 "colSkipPattern": "受让人",
 "rowSkipPattern": "其中|限售|合计|小计|总计|共计|股东名称|减持|股数|比例"
},
```

其中合计等关键词是用来排除表格最后一行，因为合计等词多半伴随 colspan，很容易出现在其他列里。

- 股东性质

```
{
  "fieldName": "shareholderProperty",
  "convertMethod": "getStringFromText",
  "pattern": "性质",
  "colSkipPattern": "受让人",
  "rowSkipPattern": "其中|限售"
},
```

这一栏并非我们所需要的 column，但是对于抽取减持前后比例却非常有用。

通常减持前后比例的表格都为以下形式：

股东 名称	减持 方式	减持期间	减持均价 （元）	减持股数 （股）	减持比例 （%）
宁波睿勇投资有限公司	大宗交易	2014 年 5 月 27 日	9.32	1,950,000	1.46
		2014 年 5 月 28 日	9.30	1,940,000	1.44
	合计	-	-	3,890,000	2.90

股东名称	股份性质	本次减持前持有股份		本次减持后持有股份	
		股数（股）	占总股本比例 （%）	股数（股）	占总股本比例 （%）
宁波睿勇投资有限公司	合计持有股份	7,784,550	5.81	3,894,550	2.91
	其中：无限售条件股	7,784,550	5.81	3,894,550	2.91
	有限售条件股	0	0	0	0

我们可以通过读取到股份性质这一列，排除其中、限售等关键词所在列，成功获取正确信息。

- 截止日期

```
{
  "fieldName": "finishDate",
  "convertMethod": "getDateFromText",
  "pattern": "日期|期间|时间",
  "rowSkipPattern": "合计|小计|总计|共计"
},
```

对于表格 pattern 的识别主要关注对于文本的了解和表格的归类。

2、 定增

定增文件和增减持文件主要的不同在于定增对于文本的每个表格都要进行抽取，而非只抽第一个表格作为record，后面的所有的表格作为辅助信息。同时，定增由于抽取了所有表格，这中间涉及到重复的record，我们也进行了相应的去重复筛除。

模式匹配

- 表头所需模式匹配

```
{
  "fieldName": "PurchaseNum",
  "convertMethod": "getLongFromText",
  "pattern": "增加.*股|认购股数|增加数量|认购数量|获配数量|获配股数|申购股数|申购数量|配售股数|配售数量|认购股份数量[(股)]{0,3}",
  "colSkipPattern": "持|持股|限售|持有|持股数量|前|后|性质|金额|比例|价格|均价|发行前|发行后|累计|年度|分红|利润|比例|种类",
  "rowSkipPattern": "限售|合计|小计|总计|共计|代表人|协办人|电话|传真|住所|地址|累计"
},
```

```
{
  "fieldName": "PurchaseMoney",
  "convertMethod": "getLongFromText",
  "pattern": "[认购配售获配认缴]{2,}[出资金额]{1,}|认购资金|配售金额|申购金额|获配金额|配售资金|申购资金|获配资金|认缴出资|认购金额[(万元)]{0,4}",
  "colSkipPattern": "持|持股|限售|持有|持股数量|前|后|性质|比例|价格|均价|数量|发行前|发行后|累计|年度|分红|利润|比例|种类",
  "rowSkipPattern": "限售|合计|小计|总计|共计|代表人|协办人|电话|传真|住所|地址|累计"
},
```

- TableSkipPattern

```
"tableSkipPattern":"电话|传真|住所|地址"
```

这一栏是定增相比增减持多的一个识别属性，因为定增需要识别所有的表格，而这中间有很多表格都是冗余无用的，但是仅仅通过表头模式匹配则很有可能被选中。尤其是每个股东的信息，电话传真住所地址等几乎大部分公告都会出现，为了加快扫描的速度，我们只要出现电话传真等字样就立刻排除该表格。

```
def is_match_table_skip_pattern(self, text,col_skip_pattern):
  if col_skip_pattern is None:
    return False
  match = col_skip_pattern.search(text)
  return True if match else False
```

以上是该pattern的实现。这一行的设计同时也基于正确的抽取结果了不太可能包含这些字样。

重复Record排除

我们在每次新抽取到一个record之前都会检测是否该record的主键和已有的record完全符合，如果完全符合，那么该record将用来对原有的record信息进行补充，如果有value 冲突，我们先根据相关信息判断，如若无法判断则信赖原有的record， 这一点的设计也是基于"重大提示”这一个特性。在公告的开头很多文件都会出现重大提示等字样，并将结构化数据的表格呈现，所以先出现的 table 包含正确信息的概率更大。

这是找出重复 record 的过程。

```
for extract_record in record_list:
  if record.Buyer == extract_record.Buyer:
    redundant = True
    redun_record = extract_record
    break
```

下面是我们具体merge 两个 record 的增发数量的函数。

```
def merge2Record(self, new_record, old_record):
    # 已知buyer相等，把信息存到旧的里面
    if new_record.PurchaseNum != old_record.PurchaseNum:
        if new_record.PurchaseNum is not None:
            if old_record.PurchaseNum is not None:
            # 数据冲突，取其中大的数字
                print("new_record:",new_record.PurchaseNum,"old_record:",
old_record.PurchaseNum)
                old_record.PurchaseNum =
max(old_record.PurchaseNum,new_record.PurchaseNum)
            else:
                old_record.PurchaseNum = new_record.PurchaseNum
.......
    return old_record
```

五、文本表格 Merge

由于合同里的表格较少，包含信息也较少，我们不对合同文件的表格进行处理，仅仅依赖神经网络对于甲方乙方项目等信息的识别。

而对于定增文件和增减持文件，这两者都隐含大量表格，所以表格和文本的 merge 可以说是对于提升性能非常重要的一环。由于文本是非结构化信息，表格是半结构化信息，我们在文本和表格的merge过程中，优先选择 表格信息，即如果 文本的 record 和表格的 record 都读到了该条信息，我们将会采用表格的信息作为最终答案。

定增文件

对于定增文件来说，表格处理在相关其实已经能做到40%左右的F1值，但是在对 增发方式 和 锁定期的抽取稍显不足，故我们利用接入文本识别结果来进行相关的处理。

如上述函数所示，表格处理接受 文本处理传过来的 pickle_file 作为我们的输入，每个pickle_file 是一个双重嵌套字典，外层字典的 key 是 html_id , 内层字典的 key 为 `contains_method`,`interval_set`,`receiver_dict`

其中 contains_method 表明该文本中是否 包含截止日期的信息，interval_set 表明截止日期的类型有多少种，receiver_dict 形如 {'园正集团': 12} 这样的格式。

我们先看表格是否抽取到正确信息，如果没有抽取到则看 上述三个label，如果 contains method 若为真，则抽出来的属于这条 html 的记录的认购方式为 "现金", 否则空置。
interval_set 是本篇 html 所有可能的锁定期，若只有一个，则全填这一个值，若有两个，需要看 receiver_dict , receiver_dict 记载了那个较为特殊的锁定期对应的股东，key 为股东名称，value 为他的锁定期，其余股东的锁定期均为 interval_set 中的另一个锁定期 ，若没有值，则文本中我没有看到锁定期。

增减持文件

增减持文件对于文本信息的需求则更加急迫，它需要文本提供的全称、简称对来做替换。由于表格提取到的只有全称 or 简称 一个信息，我们需要查阅从文本抽取得到的全称简称对。

我们仍然通过神经网络导出的 pickle 文件 作为输入，以下是对输入的一个简介。

我们使用以下函数获取全称、简称字典。

接下来我们匹配全程和简称。我们从表格中抽取出来的公司名称，先看它是否出现在全称字典里，如果出现则将对应的简称填充，否则查看是否出现在简称字典里，如果出现则填充对应的简称，否则将简称添至全称的位置，简称空置。同时对于出现的是股东名称，我们利用长度来判断，一是公司简称全称对里不太可能出现人名，二是人名的长度多为2-3位，而公司名字鲜有2-3位的，我们用长度也可做一次排除。

```
def getShareholder(self, shareholder):
    """
    归一化公司全称简称，返回股东全称 + 缩写
    如果出现了全程查阅全称，如果出现了简称，查阅全称
    """
    # print('全称', self.com_full_dict.keys())
    # print('简称', self.com_abbr_dict)
    if shareholder in self.com_full_dict.keys():
        if len(shareholder) > 3:
            return shareholder, self.com_full_dict.get(shareholder, "")
        else:
            return shareholder, ""
    if shareholder in self.com_abbr_dict.keys():
        return self.com_abbr_dict.get(shareholder, ""),shareholder
    #股东为自然人时不需要简称
    return shareholder, ""
```

六、经验总结

这一次的项目走了很多弯路。

一是沉迷于细节，对于一些特例或者是微不足道的错误，我都花大把的时间去 debug，而非没有把时间花在能对性能有显著提升的地方。

二是项目经验欠缺，对代码的复用以及设计不够干净。比如定增和增减持的代码，我分开成两个文件夹来写，这样一是代码重复冗长，二是每次修改都需要修改两处，经常是A文件发现代码有错改了，B文件忘记改，之后发现怎么定增的单位转换没有成功，不停一层层print下去，半个小时或一个小时发现问题出在 一个 normalize的地方，或者 unit_pattern 的代码只修改了一个，这样的错这次数不胜数。

三是善用 Git 进行版本控制。我从验收前一天的晚上开始，就开始不停经历修改、改错、F1下降、想要回滚、不知道回滚哪个版本、自己手动Debug、回忆修改了哪些地方、是哪个地方会导致性能下降、然后又测、又改、又不对劲，还有改了定增增减持不对劲、改了增减持定增不对劲，回滚之后那些后来修改的错误也不知道怎么办的问题，总之是乱入一团。

八、分工

李易非：数据格式转换，三种类型数据集标注，神经网络训练，文本信息提取，文本信息与表格信息合并

熊苗：html parser 抽取 raw 表格，表格信息提取，文本信息与表格信息合并

林新迪：html parser 抽取 raw 文本内容