transaction：实体关系抽取实验教程

一、先验数据导入

进入 transaction 文件夹后，修改用户权限

```
chmod -R 777 ./
```

创建数据库，并指定所有者

```
psql -U ke -d trans_test -h 127.0.0.1 -p 5432
```

在本地postgresql中为项目建立数据库，再在项目文件夹下建立数据库配置文件

```
echo "postgresql://ke@localhost:5432/trans_test" >db.url
```

在transaction下建立输入数据文件夹 input，脚本文件夹 udf，用户配置文件 app.ddlog，模型配置文件 deepdive.conf， 可参照给定的transaction文件夹样例格式。（提供的transaction文件夹中是已经建立完毕的项目，后面所需的脚本和数据文件都可以直接使用）

从知识库中获取已知具有交易关系的实体对（结构化的监督数据）来作为训练数据，从国泰安数据库(http://www.gtarsc.com)中公司关系-股权交易模块中下载。通过匹配有交易的股票代码对和代码-公司对，过滤出存在交易关系的公司对，存入transaction_dbdata.csv中。将csv文件放入 input/文件夹下。

在 app. ddlog 中定义相应的数据表

```
@source
transaction_dbdata(
    @key
    company1_name text,
    @key
    company2_name text
).
```

生成 postgresql 数据表

```
deepdive compile && deepdive do transaction_dbdata
```

二、待抽取文章导入

将待抽取的文章命名为articles.csv，放在input文件夹下（可以减少articles的行数来缩短后面程序运行时间）

在app.ddlog中建立对应的articles表

```
@source
articles(
        @key
        @distributed_by
    id      text,
    @searchable
    content text
).
```

导入文章到 postgresql 中

```
deepdive do articles
```

deepdive 可以通过query语句查询数据库数据。查询 id 检验导入是否成功：

```
deepdive query '?- articles(id, _).'
```

```
ke@ubuntu:~/Desktop/transaction$ deepdive query '?- articles(id, _).'
     id
------------
 1201734370
 1201734454
 1201734455
 1201734457
 1201734458
 1201734460
 1201734461
 1201734464
 1201738707
 1201738753
(10 rows)
```

三、用nlp模块进行文本处理

deepdive 默认采用 standford nlp 进行文本处理。输入文本数据，nlp 模块将以句子为单位，返回每句的分词、lemma、pos、NER 和句法分析的结果，为后续特征抽取做准备。

在 app.ddlog 文件中定义sentences表，用于存放 nlp 结果

```
@source
sentences(
        @key
    @distributed_by
    doc_id          text,
    @key
    sentence_index int,
    @searchable
    sentence_text  text,
    tokens          text[],
    lemmas          text[],
    pos_tags        text[],
    ner_tags        text[],
    doc_offsets     int[],
    dep_types       text[],
    dep_tokens      int[]
).
```

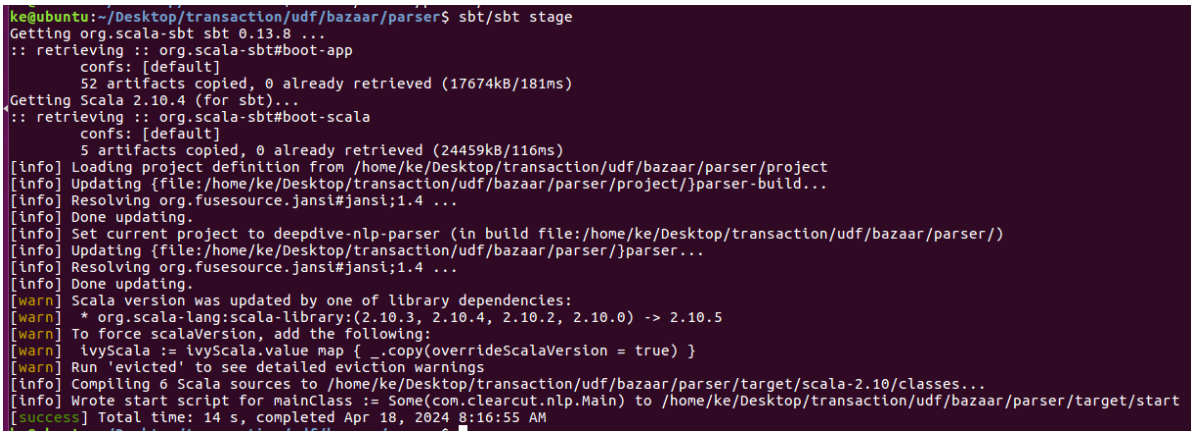
定义 NLP 处理函数 nlp_markup

```
function nlp_markup over (
        doc_id text,
        content text
    ) returns rows like sentences
    implementation "udf/nlp_markup.sh" handles tsv lines.
```

进入 bazzar/parser 目录下，执行编译命令

```
sbt/sbt stage
```

编译完成后会在target中生成可执行文件



调用nlp_markup函数，从articles表中读取输入，输出存放在sentences表中

```
sentences += nlp_markup(doc_id, content) :-
    articles(doc_id, content).
```

编译并执行生成sentences数据表

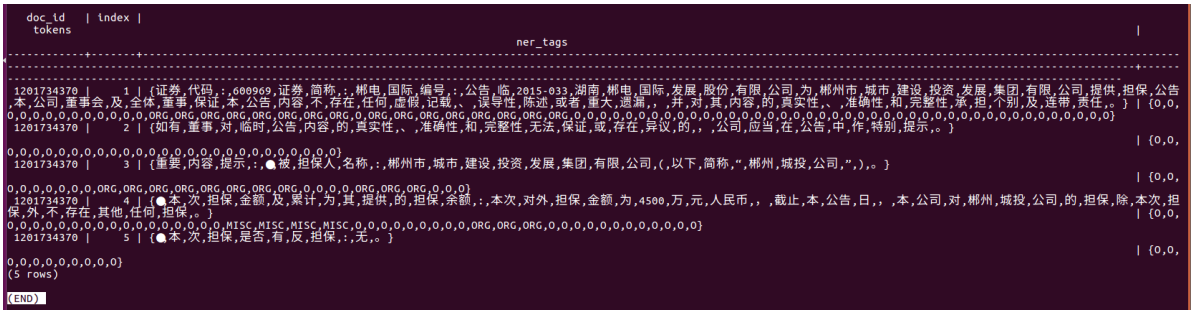
```
deepdive compile && deepdive do sentences
```

可以看到 sentences 给出的 plan 中包含 articles 表的执行。plan 中前面有冒号的行表示默认已经执行，不会重做。如果 articles 有更新，需要通过 deepdive redo articles 或者 deepdive mark todo articles 将 articles 标记为未执行，这样在生成 sentences 的过程中就会默认更新 articles。

执行以下命令来查询生成结果

```
deepdive query '
doc_id, index, tokens, ner_tags | 5
?- sentences(doc_id, index, text, tokens, lemmas, pos_tags, ner_tags, _, _, _).'
```

可以看到 id 为 1201734370 文章的前五句的解析结果



四、实体抽取及候选实体对生成

在 app.ddlog 中定义实体数据表，每个实体都是表中的一列数据，同时存储了实体在句中的起始位置和结束位置

```
@extraction
company_mention(
        @key
    mention_id     text,
    @searchable
    mention_text   text,
    @distributed_by
    @references(relation="sentences", column="doc_id",         alias="appears_in")
    doc_id         text,
    @references(relation="sentences", column="doc_id",         alias="appears_in")
    sentence_index int,
    begin_index    int,
    end_index      int
).
```

定义实体抽取的函数

```
function map_company_mention over (
        doc_id         text,
        sentence_index int,
        tokens         text[],
        ner_tags       text[]
    ) returns rows like company_mention
    implementation "udf/map_company_mention.py" handles tsv lines.
```

调用函数，从sentences表中输入，输出到company_mention中

```
company_mention += map_company_mention(
    doc_id, sentence_index, tokens, ner_tags
) :-
    sentences(doc_id, sentence_index, _, tokens, _, _, ner_tags, _, _, _).
```

编译并执行

```
deepdive compile && deepdive do company_mention
```

生成实体对，即要预测关系的两个公司。在这一步我们将实体表做笛卡尔积，同时按自定义脚本过滤一些不符合形成交易条件的公司。定义数据表如下

```
@extraction
transaction_candidate(
    p1_id   text,
    p1_name text,
    p2_id   text,
    p2_name text
).
```

统计每个句子的实体数

```
num_company(doc_id, sentence_index, COUNT(p)) :-
    company_mention(p, _, doc_id, sentence_index, _, _).
```

定义过滤函数

```
function map_transaction_candidate over (
        p1_id         text,
        p1_name       text,
        p2_id         text,
        p2_name      text
    ) returns rows like transaction_candidate
    implementation "udf/map_transaction_candidate.py" handles tsv lines.
```

描述函数的调用

```
transaction_candidate += map_transaction_candidate(p1, p1_name, p2, p2_name) :-
    num_company(same_doc, same_sentence, num_p),
    company_mention(p1, p1_name, same_doc, same_sentence, p1_begin, _),
    company_mention(p2, p2_name, same_doc, same_sentence, p2_begin, _),
    num_p < 5,
    p1_name != p2_name,
    p1_begin != p2_begin.
```

简单的过滤操作可以直接通过 app.ddlog 中的数据库语法执行，如 p1_name != p2_name 过滤掉两个相同实体组成的实体对

编译并执行，生成候选实体表

```
deepdive compile && deepdive do transaction_candidate
```

如果出现路径错误，需要将 transform.py 中 company_full_short.csv 的路径改为绝对路径

五、特征提取

定义特征表，其中 feature 列是实体对间一系列文本特征的集合

```
@extraction
transaction_feature(
        @key
        @references(relation="has_transaction", column="p1_id", alias="has_transaction")
    p1_id   text,
    @key
    @references(relation="has_transaction", column="p1_id", alias="has_transaction")
    p2_id   text,
    @key
    feature text
).
```

生成 feature 表需要的输入为实体对表和文本表，输入和输出属性定义如下：

```
function extract_transaction_features over (
        p1_id          text,
        p2_id          text,
        p1_begin_index int,
        p1_end_index   int,
        p2_begin_index int,
        p2_end_index   int,
        doc_id         text,
        sent_index     int,
        tokens         text[],
        lemmas         text[],
        pos_tags       text[],
        ner_tags       text[],
        dep_types      text[],
        dep_tokens     int[]
    ) returns rows like transaction_feature
    implementation "udf/extract_transaction_features.py" handles tsv lines.
```

函数调用 extract_transaction_features.py 来抽取特征。这里调用了 deepdive 自带的 ddlib 库，得到各种 POS / NER /词序列的窗口特征。此处也可以自定义特征。

把sentences表和mention表做join，得到的结果输入函数，输出到transaction_feature表中

```
transaction_feature += extract_transaction_features(
    p1_id, p2_id, p1_begin_index, p1_end_index, p2_begin_index, p2_end_index,
    doc_id, sent_index, tokens, lemmas, pos_tags, ner_tags, dep_types, dep_tokens
) :-
    company_mention(p1_id, _, doc_id, sent_index, p1_begin_index, p1_end_index),
    company_mention(p2_id, _, doc_id, sent_index, p2_begin_index, p2_end_index),
    sentences(doc_id, sent_index, _, tokens, lemmas, pos_tags, ner_tags, _, dep_types, dep_tokens).
```

编译并执行，生成特征数据库

```
deepdive compile && deepdive do transaction_feature
```

执行如下语句，查看生成结果

```
deepdive query '| 20 ?- transaction_feature(_, _, feature).'
```

```
ke@ubuntu:~/Desktop/transaction$ deepdive query '| 20 ?- transaction_feature(_, _, feature).'
                          feature
-----------------------------------------------------------
 IS_INVERTED
 INV_WORD_SEQ_[郴州市 城市 建设 投资 发展 集团 有限 公司]
 INV_LEMMA_SEQ_[郴州市 城市 建设 投资 发展 集团 有限 公司]
 INV_NER_SEQ_[ORG ORG ORG ORG ORG ORG ORG ORG]
 INV_POS_SEQ_[NR NN NN NN NN NN JJ NN]
 INV_W_LEMMA_L_1_R_1_[为]_[提供]
 INV_W_NER_L_1_R_1_[O]_[O]
 INV_W_LEMMA_L_1_R_2_[为]_[提供 担保]
 INV_W_NER_L_1_R_2_[O]_[O O]
 INV_W_LEMMA_L_1_R_3_[为]_[提供 担保 公告]
 INV_W_NER_L_1_R_3_[O]_[O O O]
 INV_W_LEMMA_L_2_R_1_[公司 为]_[提供]
 INV_W_NER_L_2_R_1_[ORG O]_[O]
 INV_W_LEMMA_L_2_R_2_[公司 为]_[提供 担保]
 INV_W_NER_L_2_R_2_[ORG O]_[O O]
 INV_W_LEMMA_L_2_R_3_[公司 为]_[提供 担保 公告]
 INV_W_NER_L_2_R_3_[ORG O]_[O O O]
 INV_W_LEMMA_L_3_R_1_[有限 公司 为]_[提供]
 INV_W_NER_L_3_R_1_[ORG ORG O]_[O]
 INV_W_LEMMA_L_3_R_2_[有限 公司 为]_[提供 担保]
(20 rows)
```

六、样本打标

样本打标的目的是在候选实体对中标出部分正负例，利用已知的实体对和候选实体对关联以及利用规则打部分正负标签

在 app.ddlog 里定义 transaction_label 表，存储监督数据：

```
@extraction
transaction_label(
        @key
    @references(relation="has_transaction", column="p1_id", alias="has_trasaction")
    p1_id   text,
    @key
    @references(relation="has_transaction", column="p2_id", alias="has_transaction")
    p2_id   text,
    @navigable
    label   int,
    @navigable
    rule_id text
).
```

rule_id 代表在标记决定相关性的规则名称。label 为正值表示正相关，负值表示负相关。绝对值越大，相关性越大。

初始化定义，复制 transaction_candidate 表，label 均定义为零。

```
transaction_label(p1,p2, 0, NULL) :- transaction_candidate(p1, _, p2, _).
```

将前面准备的 db 数据导入 transaction_label 表中，rule_id 标记为"from_dbdata"

```
transaction_label(p1,p2, 3, "from_dbdata") :-
    transaction_candidate(p1, p1_name, p2, p2_name), transaction_dbdata(n1, n2),
    [ lower(n1) = lower(p1_name), lower(n2) = lower(p2_name) ;
      lower(n2) = lower(p1_name), lower(n1) = lower(p2_name) ].
```

如果只利用下载的实体对，可能和未知文本中提取的实体对重合度较小，不利于特征参数推导，因此可以通过一些逻辑规则，对未知文本进行预标记。

```
 function supervise over (
        p1_id text, p1_begin int, p1_end int,
        p2_id text, p2_begin int, p2_end int,
        doc_id         text,
        sentence_index int,
        sentence_text  text,
        tokens         text[],
        lemmas         text[],
        pos_tags       text[],
        ner_tags       text[],
        dep_types      text[],
        dep_tokens     int[]
    ) returns (
        p1_id text, p2_id text, label int, rule_id text
    )
    implementation "udf/supervise_transaction.py" handles tsv lines.
```

调用标记函数，将规则抽到的数据写入transaction_label表中。

```
transaction_label += supervise(
    p1_id, p1_begin, p1_end,
    p2_id, p2_begin, p2_end,
    doc_id, sentence_index, sentence_text,
    tokens, lemmas, pos_tags, ner_tags, dep_types, dep_token_indexes
) :-
    transaction_candidate(p1_id, _, p2_id, _),
    company_mention(p1_id, p1_text, doc_id, sentence_index, p1_begin, p1_end),
    company_mention(p2_id, p2_text,      _,              _, p2_begin, p2_end),
    sentences(
        doc_id, sentence_index, sentence_text,
        tokens, lemmas, pos_tags, ner_tags, _, dep_types, dep_token_indexes
    ).
```

建立 transaction_label_resolved 表，统一实体对间的 label 。利用 label 求和，在多条规则和知识库标记的结果中，为每对实体做 vote。

```
transaction_label_resolved(p1_id, p2_id, SUM(vote)) :- transaction_label(p1_id, p2_id, vote, rule_id).
```

执行以下命令，得到最终标签。

```
deepdive do transaction_label_resolved
```

七、模型构建

定义最终存储的表格，？ 表示此表是用户模式下的变量表，即需要推导关系的表

```
@extraction
has_transaction?(
    @key
    @references(relation="company_mention", column="mention_id", alias="p1")
    p1_id text,
    @key
    @references(relation="company_mention", column="mention_id", alias="p2")
    p2_id text
).
```

根据打标的结果，灌入已知的变量，此时变量表中的部分变量label已知，成为了先验变量。

```
has_transaction(p1_id, p2_id) = if l > 0 then TRUE
                      else if l < 0 then FALSE
                      else NULL end :- transaction_label_resolved(p1_id, p2_id, l).

#has_transaction(p1, p2) = NULL :- transaction_candidate(p1, _, p2, _).
```

最后编译执行决策表：

```
deepdive compile && deepdive do has_transaction
```

因子图构建

将每一对has_transaction中的实体对和特征表连接起来，通过特征factor的连接，全局学习这些特征的权重

```
@weight(f)
has_transaction(p1_id, p2_id) :-
    transaction_candidate(p1_id, _, p2_id, _),
    transaction_feature(p1_id, p2_id, f).
```

指定两张变量表间遵守的规则，并给这个规则以权重，变量表间的依赖性使 deepdive 很好地支持多关系下的抽取

```
# Inference rule: Symmetry
@weight(3.0)
has_transaction(p1_id, p2_id) => has_transaction(p2_id, p1_id) :-
    transaction_candidate(p1_id, _, p2_id, _).
```

编译并生成最终的概率模型

```
deepdive compile && deepdive do probabilities
```

查看预测的公司间交易关系概率

```
deepdive sql "SELECT p1_id, p2_id, expectation FROM
has_transaction_label_inference ORDER BY random() LIMIT 20"
```

```
ke@ubuntu:~/Desktop/transaction$ deepdive sql "SELECT p1_id, p2_id
        p1_id         |          p2_id          | expectation
----------------------+-------------------------+-------------
 1201734457_3_64_68   | 1201734457_3_5_10       |       0.001
 1201738707_4_42_45   | 1201738707_4_17_23      |       0.338
 1201734460_1_32_36   | 1201734460_1_91_93      |           0
 1201738707_4_17_23   | 1201738707_4_30_33      |       0.466
 1201734461_17_10_15  | 1201734461_17_136_137   |       0.042
 1201734457_18_7_11   | 1201734457_18_13_18     |       0.898
 1201734460_1_91_93   | 1201734460_1_32_36      |           0
 1201734457_18_13_18  | 1201734457_18_7_11      |       0.966
 1201734457_3_46_51   | 1201734457_3_5_10       |       0.003
 1201734457_3_22_27   | 1201734457_3_46_51      |       0.037
 1201738707_7_22_28   | 1201738707_7_17_20      |       0.615
 1201734457_24_19_20  | 1201734457_24_28_29     |       0.472
 1201734457_3_64_68   | 1201734457_3_22_27      |       0.003
 1201738707_2_66_71   | 1201738707_2_117_120    |       0.004
 1201734457_3_64_68   | 1201734457_3_46_51      |       0.094
 1201734460_1_91_93   | 1201734460_1_53_54      |       0.006
 1201734457_3_46_51   | 1201734457_3_64_68      |       0.066
 1201734457_3_22_27   | 1201734457_3_64_68      |           0
 1201734457_3_22_27   | 1201734457_3_5_10       |       0.137
 1201734457_3_5_10    | 1201734457_3_46_51      |       0.002
(20 rows)
```

至此，交易关系抽取就完成了。