**查询纠错：**

**数据准备：**

训练模型用到的数据集（trainset）：

1. query.pv.qc.norm query对应的pv数据，去掉需要qc的query

2. taobao.slots.titles.unique 淘宝标题，去重

3. query.tb.merge.pv.norm 1.和2.的合并数据，淘宝标题的pv设置为1

4. qc\_sbt sogou baidu taobao三家纠错结果相同的数据集，用于模型评测等

数据3用于训练 unigram bigram skipgram模型，由于数据量较大，取前1000W条。

数据的预处理包括：全角转半角、大写转小写

**线上模型数据文件config/data：**

1. char\_emiss 常见错字转移表 第一列表示可能需要纠错的字，后面对应纠错候选字，根据qc\_sbt计算单字纠错的纠错转移概率大于0.1得到

2. char\_emiss\_manual 同上，手动标注

3. manual\_qc\_dict 手标qc pair对

4. qc\_bigram bigram模型 p(x1|x0) = p(x0, x1) / p(x0)

5. qc\_skipgram skipgram模型 p(x2|x0) = p(x0, x2) / p(x0)

6. qc\_unigram unigram模型

7. query\_freq 第一列表示查询词，第二列表示查询词在query中的pv次数，第三列表示查询词在商品库标题的出现次数，未出现值为-1 训练方法见CalcQueryFreq.java

8. sg\_sim\_font 搜狗相似字表

9. spell\_mistake\_dict 老版纠错库

**算法流程：**

qc纠错算法流程DictTermCorrector.java

qc Ngram算法及相关参数、规则见NgramAnalyzer.java

本地测试参考DictTermCorrector.java main函数

1. 加载模型数据 SearchQcApplication.initData()
2. getQc()传入query进行是否为空检查
3. 1）并进行归一化处理：大写转小写、全角转半角BaseAnalyzer.regularize\_base(query);

2）对于命中本地字典或者redis字典的查询串直接纠错并返回；redis保存两类数据：a) 纠错词对，(A, A’)；b) 防止正确的查询词被算法纠错，(A, A)；

**if**(qc\_redis.equals(query\_norm))

{

**return** Tuple.*of*(query, *L0*);

}

**else** {//需要做qc，redis的value

**return** Tuple.*of*(qc\_redis, *L1*);

}

本地纠错库的逻辑处理同redis，区别在于纠错词和防止纠错词存放在不同的文件对应不同的变量中

if(manualQcRevertDict.contains(query\_norm))

{//过不需要做qc的词典，直接返回

return Tuple.of(query\_norm, L0);

}

if (manualQcDict.contains(query\_norm))

{//过qc固定词典，直接返回qc结果

return Tuple.of(manualQcDict.get(query\_norm), L1);

}

#redis添加操作

[dafeng@sedev1sh] ~/query-correct-redis$ java -jar build/libs/query-corrector-redis.jar -type 2 -index 1 -job add -key "果真饮料" -val 果珍饮料

#redis 查看操作

[dafeng@sedev1sh] ~/query-correct-redis$ java -jar build/libs/query-corrector-redis.jar -type 2 -index 1 -job get

1. 根据查询词的长度走不同的纠错处理流程：
2. 短查询纠错getShortQueryCorrect：

Step1.基本条件检查、字典检查；同步骤3

Step2.尝试同声纠错*getSameTongueQc*；

Step3.如果纠错效果差即qc\_level为L4，尝试模糊音纠错*getFuseTongueQc*

Step4.返回qc结果和qc\_level二元组

1. 长查询纠错getLongQueryCorrect：

Step1. 先进行分词处理，并进行纠错位置预测：

common\_segmentor.parse(query, CommonSegmentor.SegMode.Search);

ngram\_term\_algo.predictQcPosition(query);

Step2. 对分词后的词进行纠错检查、返回需要纠错的子串的下标偏移及纠错标记的三元组sub，并进行对应的纠错处理getLongQuerySubSeqence

Step3. 对于可能需要纠错的子串,但Step2未标记的即sub\_level为0, 走短查询纠错流程getShortQueryCorrect、并根据结果评级决定是否返回

Step4. 如果未找到合适的纠错子串：即start\_offset= end\_offset

或者level为L0,则对预测的异常位置前后进行纠错尝试，优先考虑异常位前后的单字，如果没有单字，前后结合的窗口长度设置为4，并根据纠错的评级确定是否纠错。

1. 同声纠错流程*getSameTongueQc*：

Step1.制作同音词候选集，并根据同音词特征及编辑距离对pv加权处理

Step2.排序并输出

Step3.再进入纠错基础条件后，符合纠错策略则纠错并进行评级；对于未找到同声候选或者同声纠错评级较差的继续走模糊音的纠错流程

1. 模糊音纠错流程 getFuseTongueQc（包括模糊音和字形相似的纠错）

Step1. 先进行纠错位置预测ngram\_term\_algo.predictQcPosition(query);

如果预测位置为-1，则返回

Step2. 查找纠错位置字的模糊音候选和形似字候选、并优选纠错结果

query.charAt(qc\_pos);

fuse\_candidates= ngram\_term\_algo.getFuseTongueCandidates(ch\_qc);

font\_candidates = emissDict.get(ch\_qc);

String qc = distillCandidate(query, qc\_pos, fuse\_candidates, font\_candidates);

Step3. 对Step2的结果评估*qcEvaluate，*根据评估结果对应调整纠错位置及策略，然后对候选字再次优选。

Mark为1时，调整纠错位置、重新获取模糊音候选和形似候选并优选纠错结果；

Mark为2时，更换纠错策略用同音纠错策略；

Mark为3并且查询词长度为2，不进行纠错；

最后分级返回levelize。

1. levelize分级规则：
2. 如果纠错前后查询序列相同即未做纠错，返回L0
3. 检查查询序列的最小互信息mut是否有增加，该值越大越好
4. 检查查询序列的异常位比值diff变化程度，该值越小越好
5. 评估qc候选的匹配程度*ruleQcEntry*
6. 根据候选查询序列的*pv及title\_freq*加入评价规则

最后返回评级结果L2或L3

1. getLongQuerySubSeqence 查找可能需要纠错的子序列，优先考虑单字。
2. 检查连续单字，并返回偏移下标，这里主要检查连续两个单字、三个单字的情况；其中遇到连续两个字时，继续向后补充一个字，并进行是否需要纠错检查，符合纠错条件则返回相应的位置偏移下标和纠错标记；
3. 如果只有一个单字且不是导致分词成单字的停词如“的”，评估前位置词和后位置词的纠错优先程度，并根据前后位置优先标记进入相应纠错策略；
4. 如果单字位置是句首或者句尾，直接和后位置词或者前位置词结合，尝试同声纠错；
5. 如果没有单字，遍历前后词窗口，大小为4，尝试同音纠错SeqNeedSameTongueQc，如果前后词结合的长度大于4,则进行切割再尝试纠错，如果纠错SeqNeedSameTongueQc返回false则尝试当前词的shortSeqNeedQc纠错尝试，返回对应的偏移下标和评级。

去词训练数据特征构造(数据存于32-ranklib文件夹，构造训练特征见脚本BuildLambdaMartFeatures.py)：

去词算法共有22个特征 5~26，构造好的数据保存在taobao.feature.ranklib.1422中；用于lambdaMart训练格式的数据存于taobao.feature.ranklib.1422.train，格式：

label qid:$id $feaid:$feavalue $feaid:$feavalue … #description

1 query #查询

2 hints\_num #淘宝对应的hints个数

3 term #查询分词后对应的词

4 term\_wight #term被hints索引的个数 label值 l

5 term\_num #当前query的term个数 l

6 term\_index\_from\_start #term位置索引 l

7 term\_index\_from\_start\_ratio #term位置索引平均值 g

8 term\_index\_from\_end #term位置逆序索引 l

9 term\_index\_from\_end\_ratio #term位置逆序索引平均值 g

10 term\_length #term长度 g

11 hints\_total\_freq\_log #在查询query的词(200w query 30w unique term )，跟商品title中的热词 大于等于3 的词表）做交集（大概剩余10w个term） 对词频取 log， 另外20w个取平均词频 取log作为一个常数4.327 g

12 hints\_avg\_weight #在查询query的词(200w query 30w unique term )，跟商品title中的热词 大于等于3 的词表）做交集（大概剩余10w个term） 对词weight取 log， 另外20w个取平均词weight 取log作为一个常数0.1277 g 同上

13 dict\_freq\_log #在商品title数据中的词频前10W取log 10w词之后取默认值0 g

14 querylog\_pv\_log #在pv数据中的词频大于10取log 并用于后续的过滤 不大于10取默认值0.1？ （pv）

15 querylog\_sentence\_ratio #单个term成句的概率， 该term词频不满足14中的过滤条件即大于等于10 算平均值作为默认值 g

16 querylog\_avglen #term所在query log的平均长度， 该term词频不满足14中的过滤条件即大于等于10 算平均值作为默认值 g

17 querylog\_avgpos #term所在query log的平均位置， 该term词频不满足14中的过滤条件即大于等于10 算平均值作为默认值 g

18 querylog\_avgpos\_ratio # 当前词term的平均位置除以平均长度 即17 / 16， 该term词频不满足14中的过滤条件即大于等于10 算平均值作为默认值 g

19 querylog\_avgpos\_from\_end #term的逆序索引位置， 该term词频不满足14中的过滤条件即大于等于10 算平均值作为默认值 g

20 querylog\_avgpos\_from\_end\_ratio #当前词term的平均逆序位置除以平均长度（len+1-pos）/len， 该term词频不满足14中的过滤条件即大于等于10 算平均值作为默认值 g

21 n #在hints query中名词的概率 g

22 v #在hints query中动词的概率 g

23 a #形容词 g

24 b #修饰词 g

25 z #其他词 g

26 s #是否为停词 g