1 算法思路说明

1.1） 调研阶段：

为了解决近义词的识别问题，比如 美白/亮白/净白，洗面奶/洁面膏，口红/唇膏 ,优先调研了目前业内常用的方法，但是都不是完全适用我们电商平台的场景，下面会依次简要说明：

1）基于词林/知识库 （https://github.com/ashengtx/CilinSimilarity)

目前业内常用的 比如 哈工大词林，哈佛的wordnet等，由于基于知识库的方法是最简易的，只要输出一个词，就会获得相似的词，于是优先进行了尝试，尝试结果是：虽然准确度高，但词太少了，覆盖不到足够的样本，而且存在明显的行业隔阂的问题，绝大多数词在平台根本起不到作用，于是弃用了这个方案。

2） 预测词是否语义相似

* 基于同义词词库作为标注数据，去预测 ： 交集比较多的同义词词库/知识库不好找，样本不够，不好做预测（原因 参考 基于词林的方案），这个方案可行的条件是 方案1）要可行，覆盖样本足够多，但如果方案1）可行 ，也不需要纠结这个方案了，所以，不推荐。

3）现成的同义词包 synonyms(https://github.com/huyingxi/Synonyms)

pip install -U synonyms

调用了其nearby方法来获取同义词，但是发现输出的结果准确率比较低，有些根本不是近义词，这有可能是由于行业隔阂，出现语料不适用的问题引起，也可能是算法不够完善，所以这个方案获得结果也无法在平台直接使用。

（注意 ：synonyms主要使用的是word2vec词向量 ，虽然synonyms方案无法直接使用在平台上，但是 word2vec词向量计算相似度是一个基础性方法，是有价值的，只是需要联合其他算法，才能达到更好的效果，这个会在可行的策略思路里说明）

4） 基于共现，PMI-IR （http://www.docin.com/p-1393259156.html ）

sim_xy=hits_xy/(hits_x*hits_y)

x ,y 是两个词，

hits_xy： x y 共现的item （一般为文章）

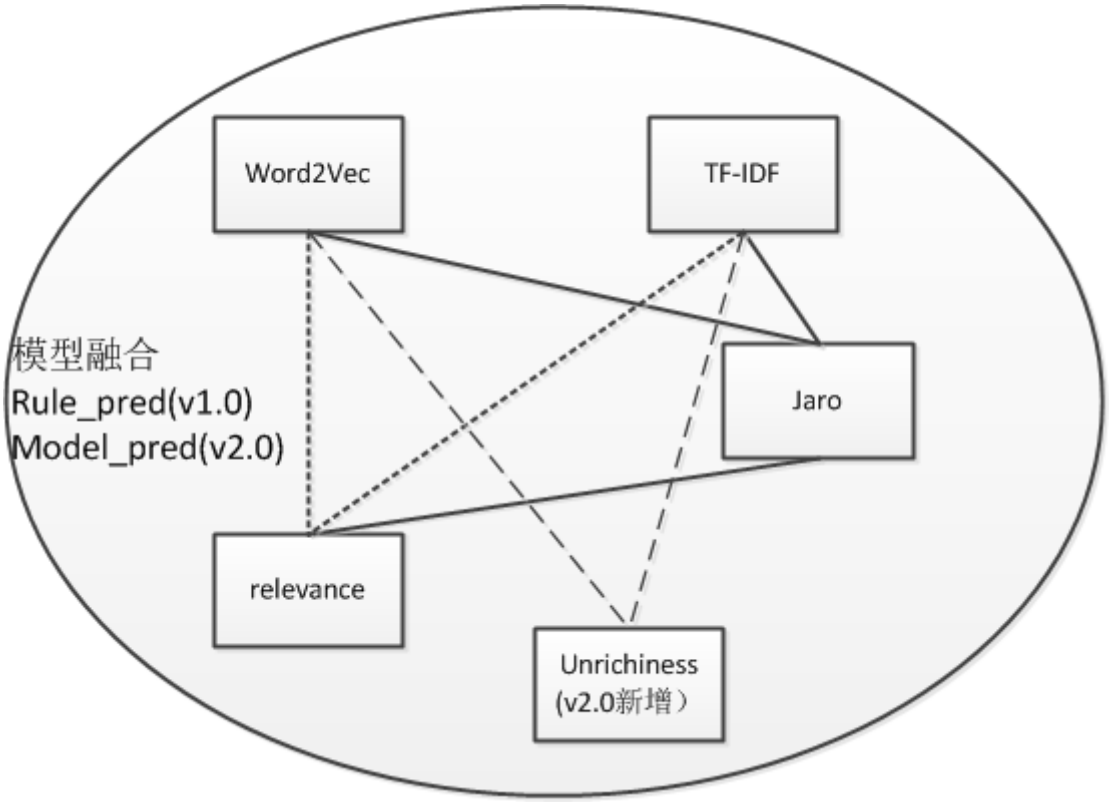
hits_x : x包含的item （一般为文章）

hits_y: y包含的item （一般为文章）

该方法 的假设场景是 两个词同时描述相同的item的比例越高，则两个词是同义词的几率越高，

这种假设场景 一般是我们在写一篇文章时，为了让表达更加丰富，对于需要多次使用的对象，往往会用多个同义词语来表达。
但是对于较短的语料来说（非文章性的），由于一般会比较精简，很少重复使用词语，也没法触发丰富表达的需求，则使用该方法就会出现很多意义不同 但关联强的词语，这种反而是被保留的。 所以，这个方案单独使用 根本无法获得近词。 （注意 共现系数 PMI-IR虽然无法直接作为判断近义词的标准，但是近义词会存在一定的共现度是合理的，这个会在可行策略的思路中对引入关联度relevance说明）

1.2） 可行的策略思路
近义词的识别 核心就是计算两个词之间的关系，而两个词之间关系的计算，一个是单纯的文本层面来计算，一个是通过生成词向量来计算。
但是单纯的从文本层面来计算是不具有语义相似的代表性的（比如jaro)，虽然对于近义词，从形态上，可分为两种同形近义词和异形近义词，但是同形的词未必都是近义词，所以同形与否只能作为辅助识别标准。
而词向量，目前常见的有基于上下文训练的词向量，比如word2vec，还有基于主题计算的词向量，比如TF-IDF。
基于上下文训练的词向量，计算的相似度 往往是语法相似度，语法相似度是**包含**语义相似的，即 如果语义相似，则使用的语法往往是相似的 。
基于主题计算的词向量，获得的相似度 往往是表达对象时的侧重习惯的相似度，即 如果语义相似，则往往**侧重**描述的对象也是相似的。
但是由于基于主题计算时，主题本身就是比较粗的粒度，不如上下文的粒度细，相应的，主题向量相似度高的top-N个，未必会包含语义相似高的（因为粒度粗，噪音词会比较多），而基于上下文计算的语法相似度高Top-N个，则可以更大限度的包含语义相似的词。
除此，近义词会同时去描述相同对象（共现），但是一般的语法使用习惯上，不会将两个近义词在较短的文本窗口中频繁联合使用（关联relevance），即近义词的上下文的关联度应该比较低。
基于以上分析和考虑，我们优先使用基于上下文的词向量计算的语法相似度 和 基于主题的词向量计算的侧重相似度，然后在此基础上，融合主文本层面的相似度及上下文的关联度作为进一步的筛选策略。
在V2.0版本中我加入了语料丰富度(单调性)这个指标，因为通过对使用规则rule预测出的结果的检查，错误率大概在20%,这20%被错误识别为近义词的词对中，有大量是由于两个词所在的语料不够丰富引起的，即如果两个非近义词存在同时描述相同主题，但是若该主题下的语料不够丰富，会导致每个词训练的word2vec词向量不具有代表性，从而出现假相似的情况。
最终 算法的整体框架包含的核心子算法模块是：



2、算法设计

2.1 word2vec

1）使用已有的包和方法计算词向量和相似度

2）训练词向量时的设置：

上下文的文本窗口window=3

输出的词向量长度size=100

3）计算词向量相似度时设置：

输出top-N个 ，另N为10

2.2 tf-idf 向量

* 使用已有的包计算tf-idf 权重 然后自己构建tf-idf向量 和实现相似度计算方法。
* 构建tf-idf向量方法：

由于 tf-idf向量我们想表达的近义词的核心概念是 **主题的侧重性**

所以在构建向量时需要：

首先对tf-idf权重在主题内去中心化 --> 每个词在每个主题上的代表性

然后对每个词的tf-idf 向量进行归一化 --> 每个词对所有主题的侧重性

* tf-idf相似度计算：

由于tf-idf向量是非常稀疏的，计算起来复杂度非常高，而且由于稀疏，计算结果往往无法充分体现相似度差异。

为解决这个问题，起初我们首先想到是进行MF形成latent-feature ，

但是MF实际是牺牲了矩阵精度的情况下进行相似度计算的，

这种方法比较适合于粗粒度层面的相似度计算，比如LSI方法计算文章的主题相似度，
而对于词层面的语义相似度，牺牲精度后的计算结果就会误差很大，所以这里摒弃了MF方案。

** 基于交集占比衰减的并集非稀疏余弦相似度 **

tfidf_sim=[cosine(union_v1,union_v2)*0.5+0.5]*decay_coef

decay_coef=min(min(r_v1,r_v2)/min_intersect_rate ,1.0)

说明：

v1: 词1的tfidf-vec的非稀疏部分

v2：词2的tfidf-vec的非稀疏部分

union: v1和v2的并集位置

union_v1: v1中属于union的补集位使用na_val补全后的向量，这里na_val取0.0

union_v2: v2中属于union的补集位使用na_val补全后的向量，这里na_val取0.0

> 示例：

> 词1的非稀疏位置是 [3，10，40] 对应的值是[1.0,0.4,0.7]

> 词2的非稀疏位置是 [3，15，40 ，68] 对应的值是[0.3,0.5,0.4,1.0]

> 则union为[3，10，15，40，68]

> 则 union_v1是[1.0 , 0.4 ,0.0 ,0.7 ,0.0]

> union_v2 是[0.3 ,0.0 , 0.5 ,0.4 ,1.0]

decay_coef: 基于交集占比的衰减系数

r_v1 : v1和v2的交集位置的长度占 v1的长度比例

r_v2 : v1和v2的交集位置的长度占 v的长度比例

min_intersect_rate： 常参，开始衰减的交集占比的阈值，低于这个值时，decay_coef<1

> 根据上面的示例：

> v1 v2的交集位置是 [3 ,40],长度是2

> r_v1=2/3

> r_v2=2/4

tfidf_sim公式 仅使用非稀疏部分，在不牺牲矩阵值的精度的情况下，比较准确的计算了词的主题侧重性的相似度。

2.3 上下文关联度relevance

这个关联度在之前很多算法中使用过，比如产品卖点个性词提取，在词组识别时使用过，这里我们简单粗暴，仅仅看两个词在共现时的距离系数的平均值

avg_relevance=sum(1/power(pos_diff,pos_diff/2))/N

pos_diff： 两个词在某个句子中距离

N： 两个词共现的句子数

这个具体可参考 产品卖点个性词提取时的词组识别公式，对于近义词的这个应用场景，我们只使用了avg_relevance和共现比例co_r=N/total_N

2.4 jaro

略，不重复赘述，请参考品牌ID聚合wiki文档

2.5 词的上下文语料单调性：unrichness （不丰富度）

定义： 在指定的上下文窗口内，对于某个词A的上下文关联词序列B：

Bi在A的上下文语料中的占比：

$$P(B_i|A) = \frac{N_D(B_i \cap A)}{N_D(A)} \quad (B_i \in B)$$

(ND: 出现的Doc的Number)

Bi在共现主题C下的频繁度：

$$f_A(B_i) = \frac{P_C(B_i)}{Avg(P(C)) + Std(P(C))} \quad (B_i \in B)$$

(C: A和Bi共现的主题及下的所有词对象)

基于以上两个基本公式，下面三个公式的目标值均是用来衡量词A的上下文语料单调性的：

A的上下文语料中，B的频繁子序列Bf 的加权占比：

$$P(B^f|A) = \frac{\sum_{f_A(B_i)>1} P(B_i|A)}{\sum P(B_i|A)} \quad (f\text{:频繁标志})$$

包含频繁序列Bf时，A的上下文语料的频繁度：

$$PF(B^f|A) = \log_2\left(\sum_{f_A(B_i)>1} P(B_i|A) \times f_A(B_i)\right) \quad (f\text{:频繁标志})$$

则A的上下文语料的单调性：

$$\text{Unrichness(A)} = P(B^f|A) \times Pf(B^f|A)$$

即，上下文词序列B中包含的在主题下频繁的对象越多，频繁对象的频繁度越高，则认为词A的上下文语料的单调性越强。

另一个辅助衡量，A的上下文语料的分布密集差异性：

$$Pover(A) = \frac{Avg(P(B_i|A))}{thr} \quad (B_i \in B, thr = {}^{10.0}/_{w2vSize})$$

即，上下文次序列B的平均占比越高，则说明语料越不丰富，存在少量词大量出现在上下文的情况。

2.6 融合策略

** V1.0 **

融合策略是保证最后效果的关键:

> 1) if word2vec_sim>0.7 & tfidf_sim>0.55 ：

> if tfidf_sim>=0.7 and w2v_sim+tfidf_sim>=1.5 ： maybe_sim=1

> elif tfidf_sim>=0.7 and w2v_sim+tfidf_sim<1.5 and jaro>0.8 : maybe_sim=1
> elif tfidf_sim<0.7 and tfidf_sim+jaro>0.95 : maybe_sim=1
> 2) if maybe_sim==1:
if co_r<4.0:
if 0<avg_relevance<1/window**(window/2.0) : keep_sim=1
elif jaro>0 and flag_sim>=0.5:
if 1/window**(window/2.0) <=avg_relevance<0.9 : keep_sim=1
elif avg_relevance>0.9 and co_r<1.0 and co_N<N_thr : keep_sim=1
> 3) if keep_sim==1 and tf>N_thr and df>df_thr: is_sim=1

** v2.0 **
包含3个方式：
1) 类似v1.0的基于策略的融合 rule_pred;
或者使用模型预测，融合word2vec_sim/tfidf_sim/jaro_sim/relevance/unrichness的结果作为feature
2) 使用GBDT单模型预测；
3) K折交叉验证的stacking方式多模型预测；