自然语言处理之近义词包 Synonyms

Synonyms

目前很缺乏质量好的中文近义词库，于是考虑使用word2vec训练一个高质量的近义词库将“非标准表述”映射到“标准表述”，这就是synonyms的起源。在经典的信息检索系统中，相似度的计算是基于匹配的，而且是query经过分词后与文档库的严格匹配，因此没有利用词汇间的语义关系。而word2vec使用了大量数据，利用上下文信息进行训练，将词汇映射到高维空间，并将在高维空间中的距离作为词语语义相似度的计算依据。明确了“语义关系”的度量方法，我们就可以进一步利用词汇间的距离进行检索。所以，在算法层面上，检索更是基于“语义距离”而非“规则匹配”。

应用场景

近义词库的使用是非常广泛的，可以用于很多的自然语言处理任务，在信息检索、文本聚类、机器翻译、词义消歧等领域有着广泛的应用，比如：

- 自动摘要

自动摘要中，使用抽取式，比如TextRank算法，就需要计算不同句子之间的相似度，作为不同节点之间的权重值。

- 搜索引擎

  比较用户的查询和数据库中的文档的近似度，除了要通过文字之间匹配的方法，还要借助近义词的语义上的关系来查找，使搜索引擎更智能。

现有的近义词包

- 同义词词林

《同义词词林》是梅家驹等人于1983年编纂而成，现在使用广泛的是 哈工大社会计算与信息检索研究中心维护的《同义词词林扩展版》，它精细的将中文词汇划分成大类和小类，梳理了词汇间的关系，同义词词林扩展版包含词语77,343条，其中32,470被以开放数据形式共享。
知网, HowNet

- HowNet

也被称为知网，它并不只是一个语义字典，而是一个知识系统，词汇之间的关系是其一个基本使用场景。知网包含词语8,265条。

国际上对词语相似度算法的评价标准普遍采用 Miller&Charles 发布的英语词对集的人工判定值。该词对集由十对高度相关、十对中度相关、十对低度相关共 30 个英语词对组成，然后让 38个受试者对这 30 对进行语义相关度判断，最后取他们的平均值作为人工判定标准 。然后不同近义词工具也对这些词汇进行相似度评分，与人工判定标准做比较，比如使用皮尔森相关系数。在中文领域，使用这个词表的翻译版进行中文近义词比较也是常用的办法。

- 与Synonyms的对比

Synonyms的词表容量是125,792，下面选择一些在同义词词林、知网和Synonyms都存在的词汇，给出其近似度的对比：

词语	2016词林改进版	知网	synonyms	人工标准
“轿车”, “汽车”	0.82	1.0	0.73	0.98
“宝石”, “宝物”	0.83	0.17	0.71	0.96
“旅游”, “游历”	1.0	1.0	0.59	0.96
“男孩子”, “小伙子”	0.81	1.0	0.88	0.94
“海岸”, “海滨”	0.94	1.0	0.68	0.93
“庇护所”, “精神病院”	0.96	0.58	0.64	0.90
“魔术师”, “巫师”	0.85	0.58	0.66	0.88
“中午”, “正午”	1.0	1.0	0.81	0.86
“火炉”, “炉灶”	0.98	0.58	0.85	0.78
“食物”, “水果”	0.35	0.14	0.74	0.77
“鸟”, “公鸡”	0.64	1.0	0.67	0.76
“鸟”, “鹤”	0.1	1.0	0.64	0.74
“工具”, “器械”	0.53	1.0	0.62	0.73
“兄弟”, “和尚”	0.37	0.80	0.59	0.70
“起重机”, “器械”	0.53	0.35	0.61	0.42
“小伙子”, “兄弟”	0.39	0.80	0.63	0.41
“旅行”, “轿车”	0.10	0.07	0.56	0.29
“和尚”, “圣贤”	0.39	0.6	0.61	0.28

注：以上词林、知网和人工标准数据来源 final_word_similarity

N-gram 模型介绍

什么是语言模型

语言模型（LM）在自然语言处理中占有重要的地位，目前主要采用的是N元语法模型，这种模型结构简单、直接。语言模型的应用场景非常广泛，比如在机器翻译领域，拼写纠错领域，语音识别领域。

一个语言模型通常构建为字符串的概率分布p(s)，这里p(s)试图反映的是字符串作为句子出现的概率，值得注意的是，语言模型与句子是否合乎语法是没有关系的，即使一个句子是完全合乎语法逻辑的，但是我们仍然有可能认为它出现的概率接近为零。怎样来判断一个句子出现的概率呢，近似的任务就是在已经出现单词的概率下，预测下一个词语的概率。因此，这样的p(w)或者p(Wn|W1,W2…Wn-1)模型被称作语言模型。那么，这样的联合概率要如何来计算呢？这就要用到概率的链式法则。

一般的，由S=W1W2…Wn组成的句子，计算其概率公式为：

$$p(s) = p(w1) * p(w2|w1) \ldots * P(wn|w1, w2, \ldots wn-1)$$

也就是说，产生第N个单词的概率由前面的N-1个词语决定。可见这样的参数估计是非常惊人的，我们也不可能从现有的训练数据集中正确的估计出这些参数，因为大部分的训练历史是不会出现在训练语料中的。因此，为了简化这个问题，我们采用了马尔可夫假设，提出了N-元语法，即我们认为当前词语的出现概率只和前N个词相关。当时，当前词的出现是独立于历史数据的，称为unigram语法；当时，当前词的出现只和前一个词语相关，称为bigram语法；当时，当前词的出现只和前两个词语相关，成为trigram。

以二元语法为例：

$$p(IloveChina) = p(I|BOS) * p(love|I) * p(China|love) * p(EOS|China)$$

其中BOS和EOS是增加的开始和结束的标记。其中估计的$p(w_i|w_i\text{-}1)$条件概率可以简单的计算为$W_i\text{-}1W_i$在某一文本中出现的频率，然后归一化，这种方法就是最大似然估计。需要的训练语料一般要有几百万个词。

语言模型的评测

怎样的语言模型算好？直观来说，就是给予真实/频繁出现的句子更高的概率，给予错误/低频出现的句子更低的概率。具体地，我们在训练集上训练数据，在测试集上进行测试。

- 对比语言模型A和B
- 最优方法

将A和B同时进行具体任务的评测，验证其准确度。比如拼写纠正、翻译系统、语音识别等。
缺点：耗时

- 次优方法

我们认为，最好的语言模型能够给出的下一个词的可能尽可能的少，也就是语言模型预测下一个词的数量越少，越可能出现最理想的，这个想法就是困惑度（Perplexity）提出的基本思想。

word2vec 原理

早期的词向量模型采用One-hot的方法。统计语料中的所有词汇，然后对每个词汇编号，针对每个词建立V维的向量，向量的每个维度表示一个词，所以，对应编号位置上的维度数值维1，其他维度全为0。这种方式存在的问题是，无法衡量语义关系，维度往往较高，而且向量稀疏。

word2vec的效果证明了神经网络的优越性。事实上，word2vec所用的网络，是个很特殊的网络，输入层是一个超级大的用One-hot表示的词的向量，一个隐含层，然后就是输出层。输入层和隐含层之间使用矩阵 `Wv*n` 做了投射运算，而不是规范的神经网络的隐含层，因为隐含层没有使用激活函数，这些牺牲是为了减少计算量。

word2vec的直接输出结果实际上是语言模型，但是 `Wv*n` 的每一行都可以对应到词汇表中的一个词， `Wv*n` 是在训练过程中不断更新的，整个网络的执行就像是编码器，最终，带有语义上相近的词汇对应的 `Wv*n` 中的向量越来越接近，这也就是我们想要的词向量了。

两种模型和计算过程优化

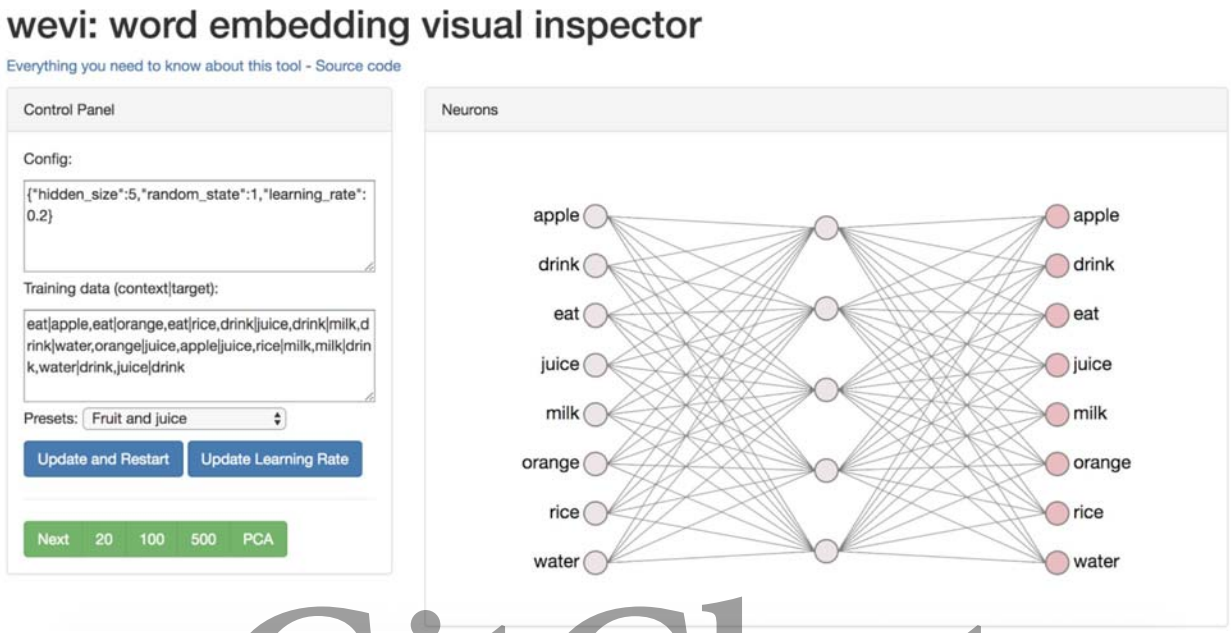
word2vec有两个模型：C-BOW模型， 即Continuous Bag of Words Model，是用上下文词语预测当前词语出现概率的模型；Skip-gram模型，已知当前词汇，预测出上下文词汇，网络的输入只包含一个单词Wi，输出是这个词的上下文[Wo1,Wo2…Woc]，这实际上是逆转了C-BOW模式的语言模型。

为了适应计算庞大的语料库，word2vec又做了两种优化方案： Hierarchical Softmax 和 Negative Sampling。这些都是为了在训练过程中，更快的计算一次前向网络的损失。hierarchical softmax是根据词频将词表进行了哈夫曼编码，这样对实际输出和理想输出词汇，做次计算就可以完成一次更新，在未经优化的方法中，要做log2(N)次计算，代表词表大小。Negative Sampling是随机选择不同于理想输出的词汇的若干词汇作为负例，计算损失，这样每次只是更新这些负例，而不是整个词表全部更新。这些技巧实际上是一种妥协，但也使训练变得可能，类似于一阶马尔可夫过程。

可视化的word2vec训练过程

目前，网络上解析word2vec原理的文章很多，笔者认为，最好的学习方式还是参考作者的发表的论文，Efficient Estimation of Word Representations in Vector Space，Distributed Representations of Words and Phrases and their Compositionality。作者发布的代码中，注释很少，变量名也很奇特，网络上有注释版，更加利于学习。

为了能更清楚的理解word2vec的训练过程，有人专门制作了可视化的工具来模拟参数的更新，可谓是煞费苦心。参考地址



训练过程

- 语料预处理
- 下载 维基百科中文语料

得到大约23万篇中文语料的 text 文档： `wiki.zh.text （～750MB）`

- 繁简转换：安装 opencc，进行繁简转换

```
opencc -i wiki.zh.text -o wiki.zh.jian -c zht2zhs.ini
```

- 分词

安装结巴分词，对语料使用精确模式进行分词

- 训练

在分词后，得到可以被用于训练的语料，目前有很多 word2vec 的版本，有些是在原版基础上做额外的优化，这些优化有的是在性能层面，也有的是在算法层面，还有的单纯是换了一下实现的语言，这些让 word2vec 更加流行，但也不容易说清楚哪个版本最好用。

Synonyms 使用的是 word2vec 论文作者发布的版本，可以在这里 下载。启动训练的命令如下：

```
word2vec -train $INPUT \
    -output $OUTPUT \
    -threads 10 \
    -size 100 \
    -min-count 100 \
    -window 5 \
    -sample 1e-5 \
    -negative 10 \
    -hs 0 \
    -binary 1 \
    -cbow 1 \
    -iter 100 \
    -save-vocab $OUTPUT_VOCAB
```

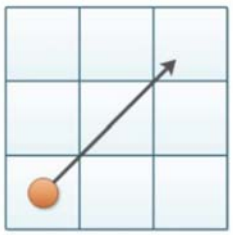
训练的过程会比较漫长，因为语料大，计算消耗CPU和内存，我们训练一次要1天时间，word2vec在训练过程中，会有随机读取负例等情况，所以，结果是随机的。网络的参数，比如维度、负例数目、窗口大小都需要根据结果调整。结果的好坏是通过选定一些词，对其距离进行测试决定的。目前，也有很多分析word2vec结果的工具。

计算句子相似度

- 计算距离

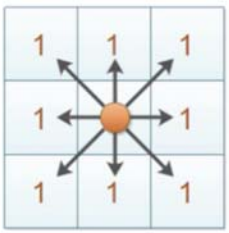
我们尝试了很多种相似度的计算方法，大家可以通过这个链接看到不同距离的python实现，比如，欧式距离、切比雪夫距离和曼哈顿距离，可以表示如下。

Euclidean Distance



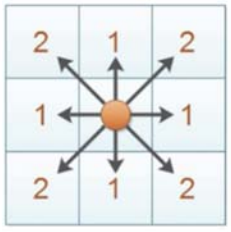
$$\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$$

Chebyshev Distance



$$\max(|x_1-x_2|,|y_1-y_2|)$$

Manhattan Distance



$$|x_1-x_2|+|y_1-y_2|$$

- 编辑距离

编辑距离（Edit Distance），又称Levenshtein距离，是指两个字串之间，由一个转成另一个所需的最少编辑操作次数。许可的编辑操作包括将一个字符替换成另一个字符，插入一个字符，删除一个字符。一般来说，编辑距离越小，两个串的相似度越大。

- 余弦距离

余弦距离是用向量空间中两个向量夹角的余弦值作为衡量两个个体间差异的大小的度量。

我们分别对上述的切比雪夫距离、余弦距离、杰卡德距离、曼哈顿距离等进行了测试，发现最好的是余弦距离，然后通过经验发现编辑距离应该也是一个需要考虑的度量方式，因此最后采用了编辑距离和余弦距离的加权组合。值得注意的是，现在的加权系数并不是最优的，而是根据经验设定的，在测试数据集上没有达到最优表现，因为在数据集上最优的参数在实际中遇到了一些问题（详见 `two sentence are partly equal#14` ），可见现在的测试数据集并没有覆盖足够多的场景，这也是后续需要着重优化的地方。

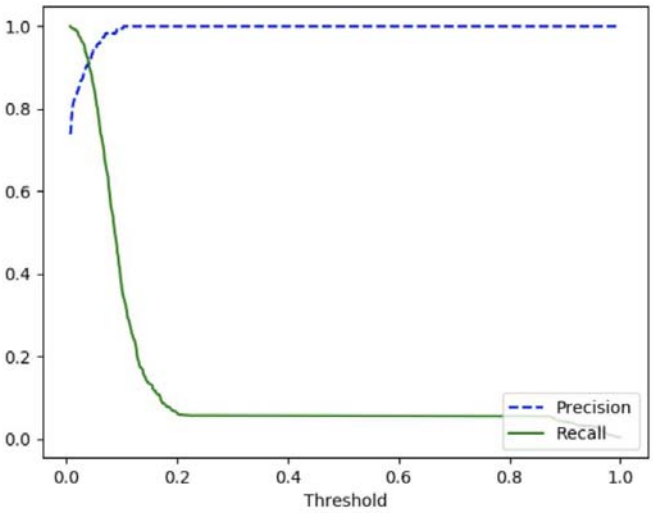
具体的测试过程，我们采用了pr曲线、roc曲线两种图像，来确定最优值。

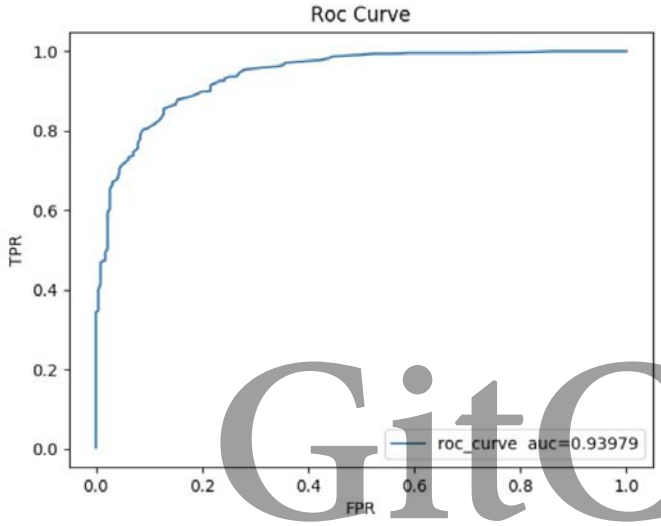
```
def draw_roc_curve(labels, scores, name, out='.'):
    fpr, tpr, th = m.roc_curve(labels, scores)
    roc_auc = round(m.auc(fpr, tpr),5)
    subtitle = name +'auc='+str(roc_auc)
    pl.plot(fpr, tpr, label=subtitle)
    pl.xlabel('FPR')
    pl.ylabel('TPR')
    pl.legend(loc=4)
    pl.title("ROC curve")
    pl.savefig(os.path.join(out, name+'.png'), format='png')
    pl.gcf().clear()
```

```
def draw_pr_with_thresholds(labels,scores,name,out='.'):
    p, r, th = m.precision_recall_curve(labels, scores)
    pl.plot(th, p[:-1], 'b--', label='precision')
    pl.plot(th, r[:-1],'g--', label='recall')
    pl.xlabel('threshold')
    pl.legend(loc=4)
    pl.savefig(os.path.join(out,name+'.png'),format='png')
```

```
def draw_pr_curve(labels, scores, name, out='.'):
    p, r, th = m.precision_recall_curve(labels, scores)
    plot = pl.plot(r, p, label=name)
    pl.legend(loc=4)
    pl.title('pr curve -', name)
    pl.savefig(os.path.join(out, name+'.png'), format='png')
```

那么什么样的值才算好的呢？简单来说，对于ROC曲线，越靠近右上角的越好，对于PR曲线，越靠近左上角的越好，对于PR_threshold曲线，Precision和recall相交点越小越好。

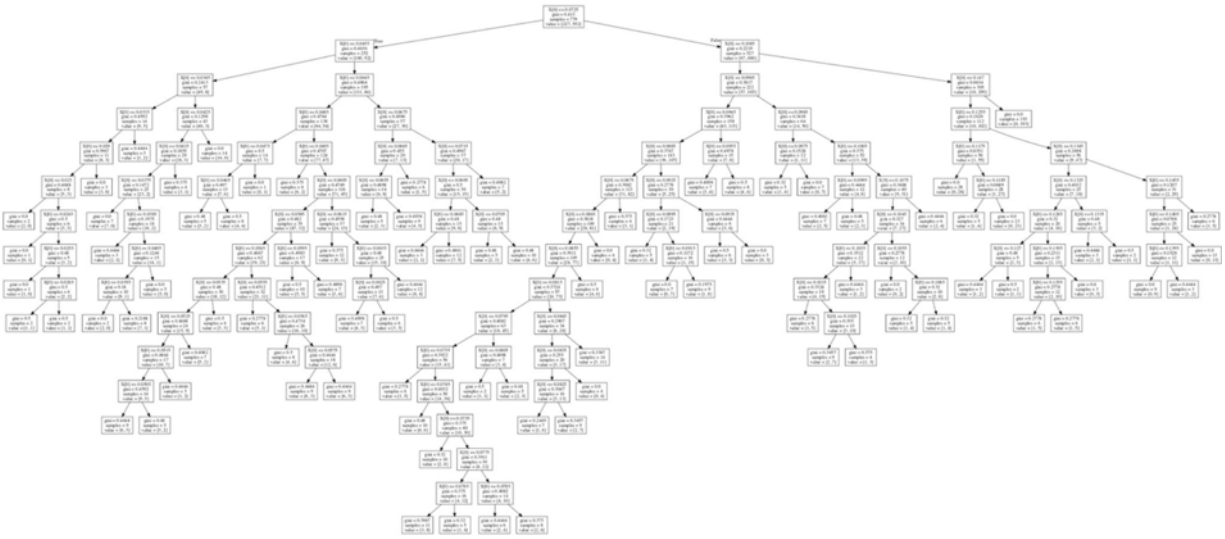




- 基于决策树的相似度阈值计算

```
from sklearn import tree
import graphviz
clf = tree.DecisionTreeClassifier()
clf = clf.fit(x,y)
dot_data = tree.export_graphviz(clf, out_file=None)
graph.render('tree')
```

根据公式输出的概率值和真实的标签，我们可以输入决策树确定阈值，决策树会输出一个这样的图，我们找到根结点，会看到有x[0]<0.0725，类似这样的，我们就可以模型小于等于0.0725的值会被划分为0类，大于0.0725的值会被划分为1类。



X[0] <= 0.0725
gini = 0.413
samples = 779
value = [227, 552]

由决策树得加权相似度计算公式：

```
score = min(g * 5 + u * 0.8, 1.0)
```

Synonyms的距离计算方案

```
def _similarity_distance(s1, s2):
    '''
    compute similarity with distance measurement
    '''
    a = _sim_molecule(_get_wv(s1))
    b = _sim_molecule(_get_wv(s2))
    # https://docs.scipy.org/doc/numpy-1.13.0/reference/g
    g = 1 / (np.linalg.norm(a - b) + 1)
    u = _levenshtein_distance(s1, s2)
    r = g * 5 + u * 0.8
    r = min(r, 1.0)

    return float("%.3f" % r)
```

这是我们目前的句子相似度判别公式，结合了编辑距离（u）和余弦距离（g），参数是通过一定的测试得出的，但显然还不是最优，这部分也是需要后续优化的地方。

总结

word2vec 是统计机器学习应用于实际的一个很棒的例子。语言处理是个很难的事情，随着数据量的增大，计算性能的提升和越来越多的人加入这个行列，相信在不远的将来，能出现更多的类似于 word2vec 这样的项目，Synonyms 还有很多需要改进的地方，比如近似度评价方案、干预词表方法、更多开箱即用的算法等等。

[help me with MathJax]

GitChat