文本相似度计算方法总结

1.VSM 简介

空间向量模型 **VSM**，是将文本表示成数值表示的向量。在使用 VSM 做文本相似度计算时，其基本步骤是：

1）将文本分词，提取特征词 s:（t1,t2,t3,t4）；

2）将特征词用权重表示，从而将文本表示成数值向量 s:(w1,w2,w3,w4)，权重表示的方式一般使用 tfidf；

3）计算文本向量间的余弦值，判断文本间的相似度。

缺点：空间向量模型以词袋为基础，没有考虑词与词间的关系，近义词等。

2.LSI、PLSI

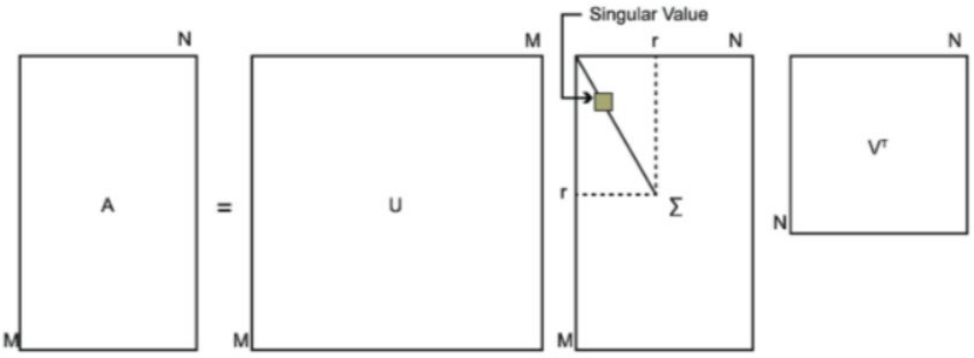
潜在语义索引(Latent Semantic Indexing,以下简称 LSI)，也叫 Latent Semantic Analysis（LSA）。是一种简单实用的主题模型。LSI 是基于奇异值分解（SVD）的方法来得到文本的主题的。

SVD：对于一个 $m \times n$ 的矩阵 A，可以分解为下面三个矩阵：

$$A_{mxn} = U_{mxm} \Sigma_{mxn} V^{T}_{nxn}$$

有时为了降低矩阵的维度到 k（在奇异值矩阵中也是按照从大到小排列，而且奇异值的减少特别的快，在很多情况下，前 10%甚至 1%的奇异值的和就占了全部的奇异值之和的 99%以上的比例。也就是说，我们也可以用最大的 k 个的奇异值和对应的左右奇异向量来近似描述矩阵。），SVD 的分解可以近似的写为：

$$A_{mxn} \approx U_{mxk} \Sigma_{kxk} V^{T}_{kxn}$$



如果把上式用到我们的主题模型，则 SVD 可以这样解释：

输入:有 m 个文本，每个文本有 n 个词。

Aij :第 i 个文本的第 j 个词的特征值，这里最常用的是基于预处理后的标准化 TF-IDF 值。

K:是我们假设的主题数，一般要比文本数少。SVD 分解后，

*Uil:*对应第 i 个文本和第 l 个主题的相关度。

*Vjm:*对应第 j 个词和第 m 个词义的相关度。

Σ*lm:*对应第 l 个主题和第 m 个词义的相关度。

也可以**反过来解释：**

输入:有 m 个词，对应 n 个文本。

*Aij:*则对应第 i 个词档的第 j 个文本的特征值，这里最常用的是基于预处理后的标准化 TF-IDF 值。

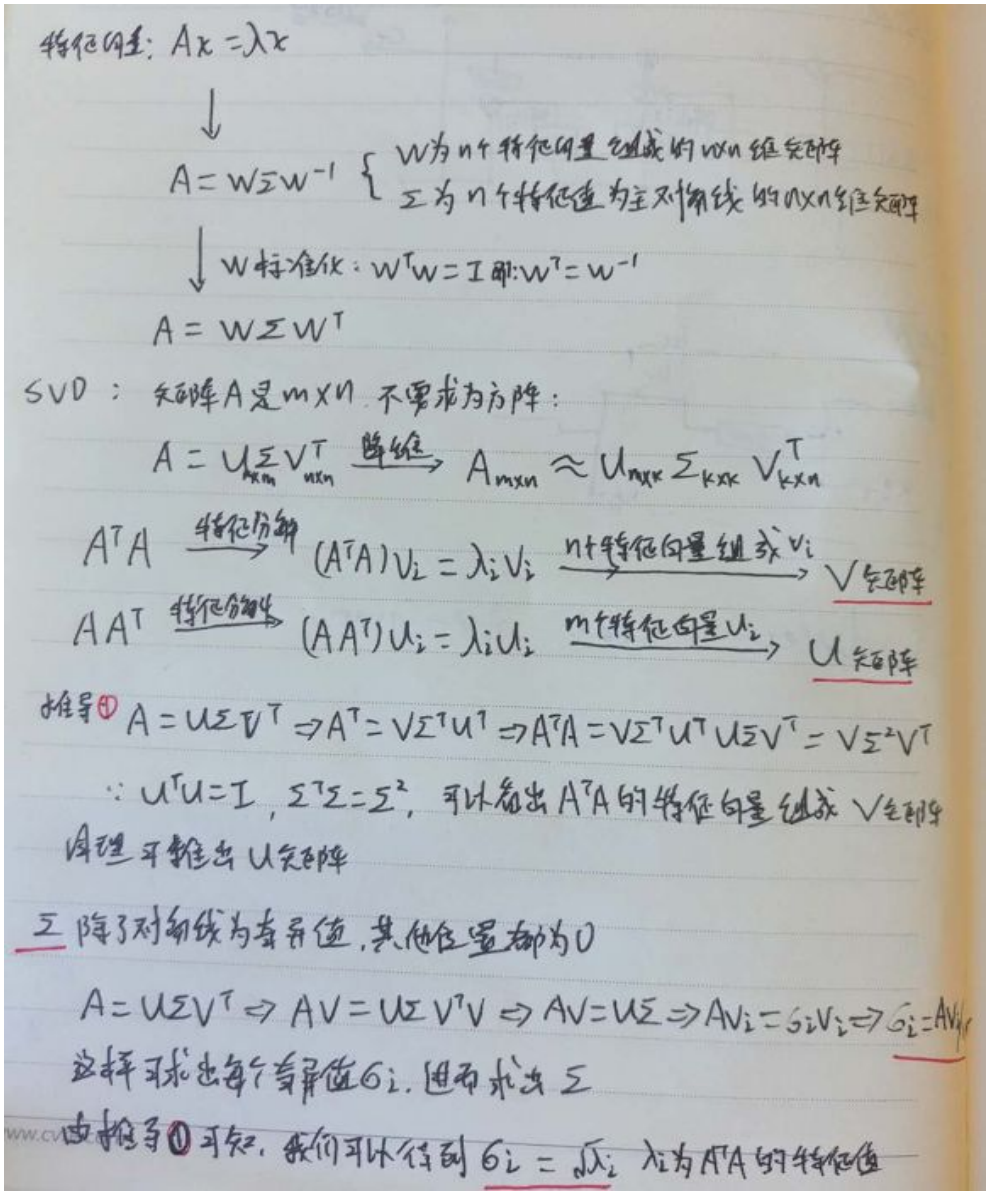
K:是我们假设的主题数，一般要比文本数少。SVD 分解后，

*Uil:*对应第 i 个词和第 l 个词义的相关度。

*Vjm:*对应第 j 个文本和第 m 个主题的相关度。

Σ*lm:*对应第 l 个词义和第 m 个主题的相关度。

SVD 分解：



LSI 举例：

假设我们有下面这个有 11 个词三个文本的词频 TF 对应矩阵如下：

Terms		d1	d2	d3		q
a		1	1	1		0
arrived		0	1	1		0
damaged		1	0	0		0
delivery		0	1	0		0
fire		1	0	0		0
gold	A =	1	0	1	q =	1
in		1	1	1		0
of		1	1	1		0
shipment		1	0	1		0
silver		0	2	0		1
truck		0	1	1		1

假定对应的主题数为 2，则通过 SVD 降维后得到的三矩阵为：

$$U \approx U_k = \begin{bmatrix} -0.4201 & 0.0748 \\ -0.2995 & -0.2001 \\ -0.1206 & 0.2749 \\ -0.1576 & -0.3046 \\ -0.1206 & 0.2749 \\ -0.2626 & 0.3794 \\ -0.4201 & 0.0748 \\ -0.4201 & 0.0748 \\ -0.2626 & 0.3794 \\ -0.3151 & -0.6093 \\ -0.2995 & -0.2001 \end{bmatrix} \qquad k = 2$$

$$\Sigma \approx \Sigma_k = \begin{bmatrix} 4.0989 & 0.0000 \\ 0.0000 & 2.3616 \end{bmatrix}$$

$$V \approx V_k = \begin{bmatrix} -0.4945 & 0.6492 \\ -0.6458 & -0.7194 \\ -0.5817 & 0.2469 \end{bmatrix} \qquad V^T \approx V_k^T = \begin{bmatrix} -0.4945 & -0.6458 & -0.5817 \\ 0.6492 & -0.7194 & 0.2469 \end{bmatrix}$$

从矩阵 *Uk* 可以看到词和词义之间的相关性。而从 *Vk* 可以看到 3 个文本和两个主题的相关性。

非负矩阵分解（NMF）：可以解决矩阵分解的速度问题

计算文本相似度：

LSI 得到的文本主题矩阵可以用于文本相似度计算。而计算方法一般是通过余弦相似度。比如对于上面的三文档两主题的例子。我们可以计算第一个文本和第二个文本的余弦相似度如下 ：

$$\text{sim}(d1,d2)=\frac{(-0.4945)*(-0.6458)+(0.6492)*(-0.7194)}{\sqrt{(-0.4945)^2+(0.6492)^2}\sqrt{(-0.6458)^2+(-0.7194)^2}}$$

模型评价指标：

目前没有相关评价指标，有提到的一种是将计算的相识度结果进行聚类，通过聚类的评价指标来评价相识度模型计算的好与坏。

参考：https://www.cnblogs.com/pinard/p/6805861.html

PLSI:

在 LSI 的基础上引入统计概率

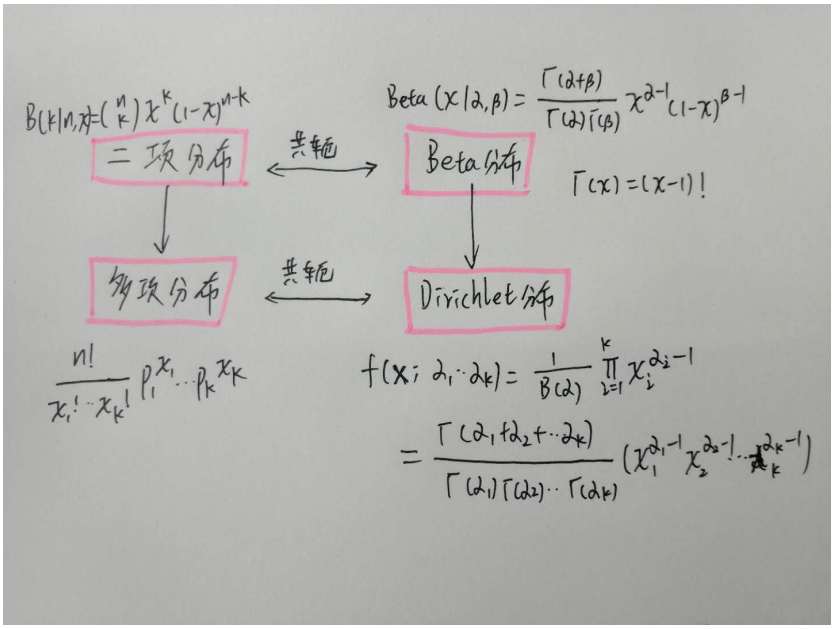
参考：https://www.cnblogs.com/pinard/p/6805861.html

3. LDA、HDP

前置知识：先验分布 + 数据（似然）= 后验分布

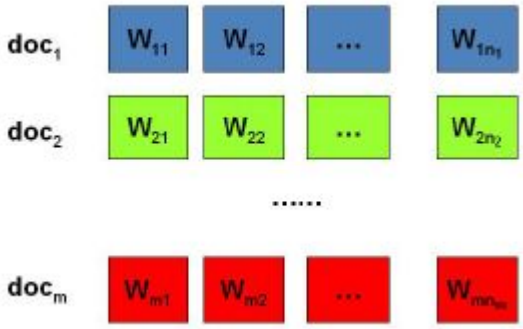
Beta 分布 + 二项分布 = Beta 分布

Dirichlet 分布 + 多项分布 = Dirichlet 分布

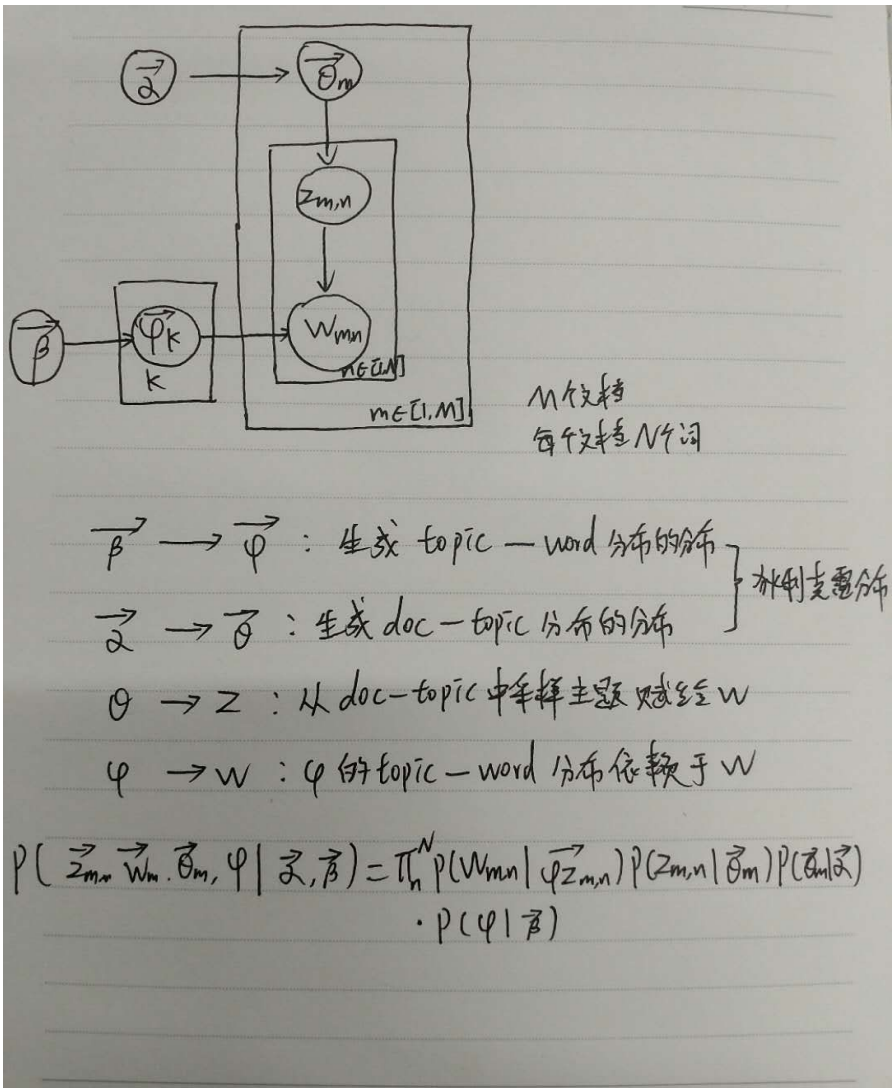


隐含狄利克雷分布(Latent Dirichlet Allocation，以下简称LDA)。LDA是一个基于贝叶斯概率的主题模型，其假设背景是“一篇文档包含多个主题，文档中的每一个词由其中的一个主题生成”。可以理解为LDA的过程就是文本的重新生成过程。

假设有*M*篇文档，对应第d个文档中有有*Nd*个词。即输入为如下图：



目标是找到每一篇文档的主题分布和每一个主题中词的分布。在LDA模型中，我们需要先假定一个主题数目*K*，这样所有的分布就都基于*K*个主题展开。那么具体LDA模型如下图：



参考：https://www.cnblogs.com/pinard/p/6831308.html

LDA 使用 Gibbs 采样算法计算每篇文档的主题分布和每个主题的词分布的

训练流程：

1）选择合适的主题数 K，选择合适的超参数向量α ,η

2） 对应语料库中每一篇文档的每一个词，随机的赋予一个主题编号 z

3) 重新扫描语料库，对于每一个词，利用 Gibbs 采样公式更新它的 topic 编号，并更新语料库中该词的编号。

4） 重复第 2 步的基于坐标轴轮换的 Gibbs 采样，直到 Gibbs 采样收敛。

5） 统计语料库中的各个文档各个词的主题，得到文档主题分布 θd，统计语料库中各个主题词的分布，得到 LDA 的主题与词的分布 βk。

预测流程：

1） 对应当前文档的每一个词，随机的赋予一个主题编号 z

2) 重新扫描当前文档，对于每一个词，利用 Gibbs 采样公式更新它的 topic 编号。

3） 重复第 2 步的基于坐标轴轮换的 Gibbs 采样，直到 Gibbs 采样收敛。

4） 统计文档中各个词的主题，得到该文档主题分布。

计算文本相识度：

类似 LSI 模型的文本相识度计算，LDA 是计算文本主题分布来计算文本间的相似度。从而免去计算词项之间的相似度，（计算两个文

本相似度可以计算与之对应的主题概率分布来实现：KL 距离、JS 距离等）

模型评价指标：

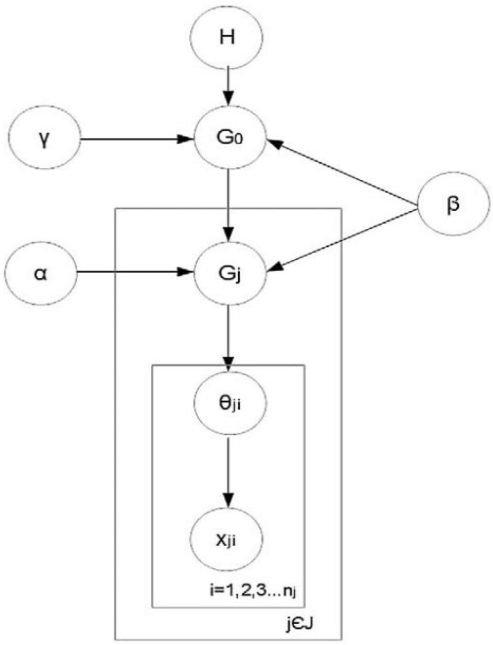
1）Perplexity(困惑度)

2）topic coherence（主题相关度）：

参考：https://www.kdnuggets.com/2016/07/americas-next-topic-model.html

层次狄利克雷过程（HDP）：

HDP 是基于 Dirichlet process（DP）的概率模型，是两层的 DP 模型。HDP 是可以自动选择主题个数。



参考：http://www.cnblogs.com/todoit/archive/2014/06/03/3753871.html

4. LSH、LSHForest

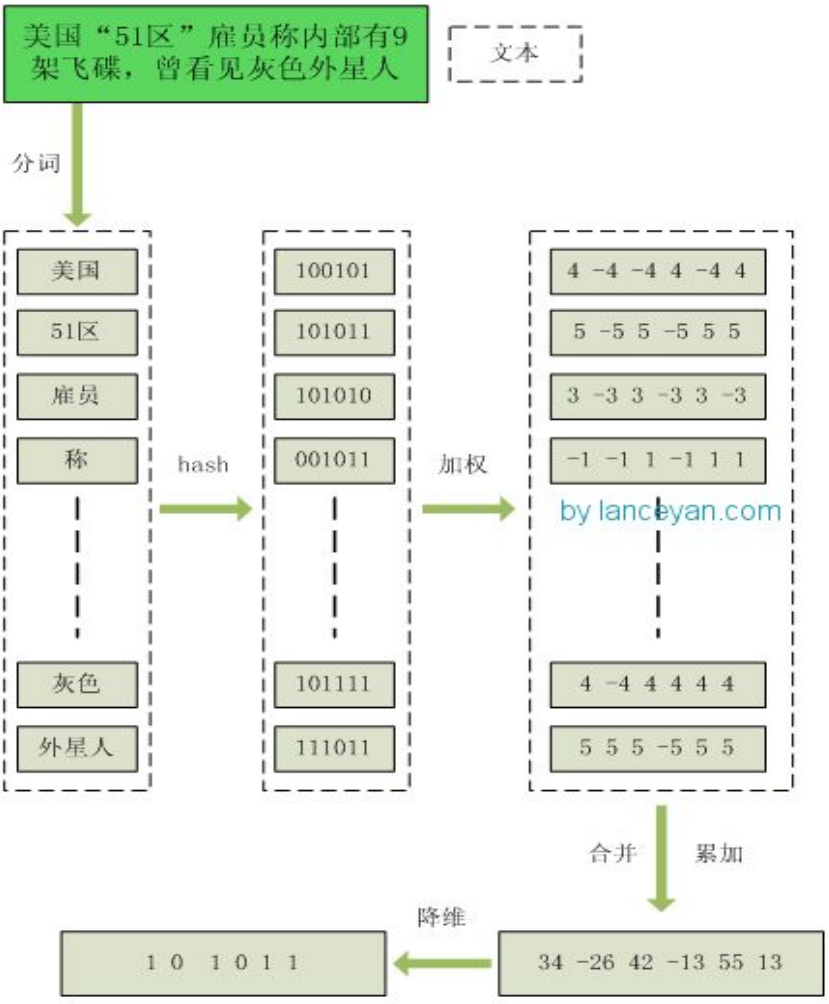
4.1 前置知识：

1）Hash 叫哈希，也叫散列，可以叫散列算法，也可以叫哈希算法；

2）hash 与其他诸如 simhash/minhash 的区别：

一般的 hash 可能两个文档本来相似，但是 hash 之后的值反而不相似了；simhash/minhash 可以做到两个文档相似，hash 之后仍然相似；

simhash:



minhash：

元素	S1	S2	S3	S4
他	0	0	1	0
成功	0	0	1	1
我	1	0	0	0
减肥	1	0	1	1
要	0	1	0	1

最小哈希值：h(S1)=3，h(S2)=5，h(S3)=1，h(S4)=2.

3）simhash 与 Minhash 的区别：

simhash 和 minhash 可以做到两个文档 Hash 之后仍然相似，但是 simhash 计算相似的方法是海明距离；而 minhash 计算距离的方式是 Jaccard 距离。

4）局部敏感哈希（LSH）与 simhash、minhash 的区别。

局部敏感哈希可以在这两者基础上更快的找到相似、可匹配的对象，而且继承了 simhash/minhash 的优点，相似文档 LSH 计算之后还是保持相似的。

4.2 LSH、LSHForest

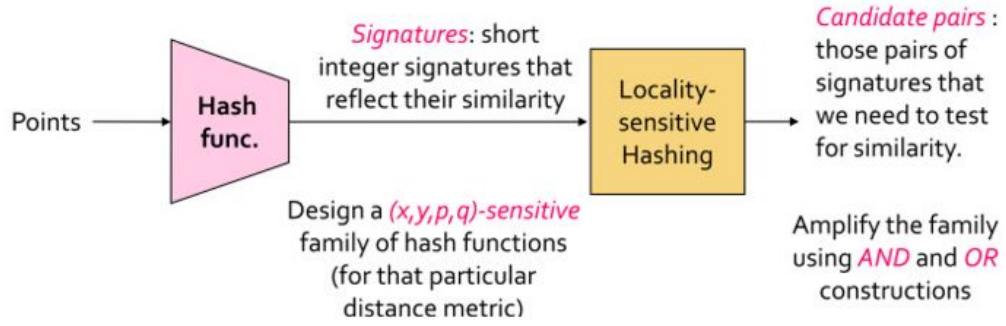
LSH 流程中有两个流程:

1)使用 simhash,minhash 将文本用装换成 hash 矩阵="**签名矩阵（Signature Matrix）**"，降维。

2)将签名矩阵进行行划分，分割成不同的 n 个 band。

3)利用 hash 算法再次 hash 签名矩阵，将相似数据点（band）映射到相同 bucket(桶)里,将不同 band 映射到不同 bucket 里。

4)计算两个句子中的 band 在相同 bucket 的统计概率，计算句子的相似性。



参考：https://blog.csdn.net/sinat_26917383/article/details/52451028

4.3 LSHForest 局部敏感随机投影森林：LSH+随机投影深林

数据个数比较大的时候，线性搜索寻找 KNN 的时间开销太大，而且需要读取所有的数据在内存中，这是不现实的。因此，实际工程上，使用近似最近邻也就是 ANN 问题。其中一种方法是利用随机投影树，对所有的数据进行划分，将每次搜索与计算的点的数目减小到一个可接受的范围，然后建立多个随机投影树构成随机投影森林，将森林的综合结果作为最终的结果。

参考：https://blog.csdn.net/sinat_26917383/article/details/70243066

5. 词向量加权求平均+余弦相识度

5.1 Word2vec 词向量求平均=句向量--》句向量相识度计算

5.2 对一个句子中所有词的词向量进行加权平均

Algorithm 1 Sentence Embedding

Input: Word embeddings $\{v_w : w \in \mathcal{V}\}$, a set of sentences $\mathcal{S}$, parameter a and estimated probabilities $\{p(w) : w \in \mathcal{V}\}$ of the words.
Output: Sentence embeddings $\{v_s : s \in \mathcal{S}\}$
1: **for all** sentence s in $\mathcal{S}$ **do**
2: $v_s \leftarrow \frac{1}{|s|} \sum_{w \in s} \frac{a}{a+p(w)} v_w$
3: **end for**
4: Form a matrix X whose columns are $\{v_s : s \in \mathcal{S}\}$, and let u be its first singular vector
5: **for all** sentence s in $\mathcal{S}$ **do**
6: $v_s \leftarrow v_s - uu^\top v_s$
7: **end for**

参考：https://blog.csdn.net/walker_hao/article/details/78974781

6. 基于 LSTM 的自动编码器（AE）求句向量+余弦相识度

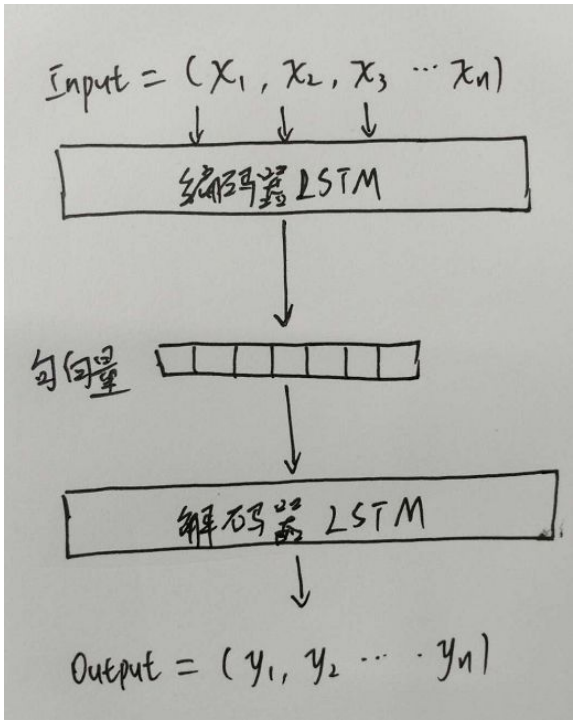
构造句向量最简单的方式就是把词向量进行加权求平均。然而这把句子看成了一个词袋，没有考虑词的顺序性。LSTM 网络被设计成处理句子的输入，并考虑了词的顺序性，因而为句子提供一个更好的表示。

自动编码器由两部分组成：

1）编码器：这部分能将输入压缩成潜在空间表征，可以用编码函数 h=f(x)表示。

2）解码器：这部分能重构来自潜在空间表征的输入，可以用解码函数 r=g(h)表示。

通过计算 Input 与 Output 的相似度衡量模型



其他改进算法：

1) KATE: K-Competitive Autoencoder for Text (一种改进的自动编码器应用，得到文本的向量表示 KDD2017)

Algorithm 1 KATE: K-competitive Autoencoder

1: **procedure** TRAINING
2: Feedforward step: $z = tanh(Wx + b)$
3: Apply k-competition: $\hat{z}$ = k-competitive_layer(z)
4: Compute output: $\hat{x} = sigmoid(W^T \hat{z} + c)$
5: Backpropagate error (cross-entropy) and iterate

1: **procedure** ENCODING
2: Encode input data: $z = tanh(Wx + b)$

与 AE 不同的是第 3 步，这里对得到的中间 code（句向量）进行了一个 K-Competitive 的操作，其思想是，Encoder 得到中间向量 code 的过程中，神经元之间的权重是不一样的，有些神经元重要，有些神经元不重要，所以作者让这些神经元进行竞争来学习得到输入的一个

模板表达，其中，选择经过激活函数得到 z 中最具竞争力的 k 个神经元(神经元的取值绝对值最大的 k 个)作为胜利者，其余的作为失败者，胜利者中又分为 positive 胜利者和 Negative 胜利者，pos 胜利者得到取值为正的失败者神经元的能量，neg 胜利者得到取值为负的失败者神经元的能量，如下图所示：

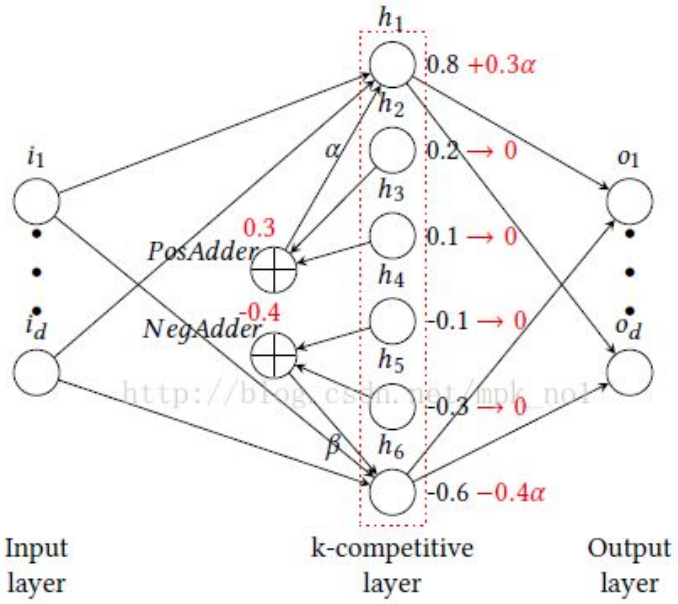


Figure 1: Competitions among neurons. Input and hidden neurons, and hidden and output neurons are fully connnected, but we omit these to avoid clutter.

这样就对神经元的能量进行了再分配，使得重要的神经元得到了加强，不重要的神经元失活。

参考：https://blog.csdn.net/mpk_no1/article/details/75201582

2) 其他对于 Autoencoder 和 LSTM 的改进及增加 Attention、正则等

7. 总结：