详细

<http://blog.csdn.net/lom9357bye/article/details/73136117>

先用svr练手，熟悉一下流程

下面要完善特征

【基本概念】

语料（corpus）：一组原始文本的集合，用于无监督地训练文本主题的隐层结构。语料中不需要人工标注的附加信息。在Gensim中，corpora是文档集的表现形式，也是后续进一步处理的基础。从本质上来说，corpora其实是一种格式或者说约定，其实就是一个二维矩阵。

向量（vector）：由一组文本特征构成的列表。是一段文本在Gensim中的内部表达。

词典（dictionary）：是所有文档中所有单词的集合，而且记录了各词的出现次数等信息。

获取数据

【数据预处理】

【生成语料词袋表示】

把所有文档转换成BOW形式，也即是文本转成了向量的初步；数组（词id,词频），每行对应一个文档，对于整个语料来看其实这是一个大的稀疏矩阵。M\*N,M表示文档数，N表示词数；

【计算tfidf向量】

把重点信息的权重增加，是属于特征选择的范畴；词频大的表示这个词对于这篇文章的重要性不一定大，如果这个词对于每篇文档都多，这个词在某篇文档就可以显得不太重要了，对全部文档进行一个综合计算，这样，就应用到了tfidf模型。

【LSA模型】

对高维度，有199252维，建立LSA模型，把整个大的矩阵映射到100~300之间的维度。这里选择150维，如把所有语料都降维到150维，把稀疏的高级矩阵变成一个计算起来会比较轻松的小矩阵，也把一些没有用的噪音给过滤掉了，这个模型可以被后来的语料查询与分类所调用。

【文本相似】

1. 直接对文本进行比较，例如，判断是否相等，判断文档最小修改次数转换等等；

2. 把文本转接成局部敏感的hash编码，通过计算编码的差异来判断文本的相似性，例如simhash；

3. 采用文本的主题模型来分析，主题相同或相似的词、文档也是很相似的；

对于计算的方法方面，各种vec距离等等都可以用；gensim求相似的用了余弦来求解的，实现方法是先把两向量换算成单位向量，然后作点乘。

【LSI潜在语义索引】

引入

主题模型是非监督的算法，得到文本按照主题的概率分布。

|  |  |  |
| --- | --- | --- |
|  | 描述 | 举例 |
| 聚类算法 | 样本特征的相似度方面 | 通过数据样本之间的欧式距离，曼哈顿距离的大小聚类等。 |
| 主题模型 | 对文字中隐含主题的一种建模方法 | 人民的名义和达康书记，很容易发现对应的文本有很大的主题相关度。但如果通过词特征来聚类的话很难找出，因为聚类方法不能考虑到到隐含的主题这一块。 |

如何找到隐含的主题呢？

常用的方法一般都是基于统计学的生成方法。即假设以一定的概率选择了一个主题，然后以一定的概率选择当前主题的词。最后这些词组成了我们当前的文本。所有词的统计概率分布可以从语料库获得，具体如何以“一定的概率选择”，这就是各种具体的主题模型算法的任务了。

还有一些不是基于统计的方法，比如LSI。

概述

潜在语义索引(Latent Semantic Indexing,以下简称LSI)，有的文章也叫Latent Semantic Analysis(LSA)。其实是一个东西，后面我们统称LSI，它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。。

简要回顾下SVD：对于一个m×n的矩阵AA，可以分解为下面三个矩阵：



有时为了降低矩阵的维度到k，SVD的分解可以近似的写为：



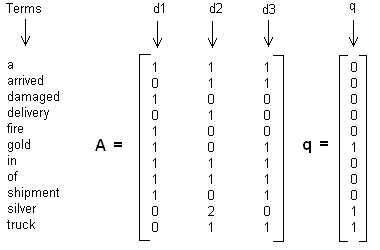
　如果把上式用到我们的主题模型，则SVD可以这样解释：我们输入的有m个文本，每个文本有n个词。而Aij则对应第i个文本的第j个词的特征值，这里最常用的是基于预处理后的标准化TF-IDF值。k是我们假设的主题数，一般要比文本数少。SVD分解后，Uil对应第i个文本和第l个主题的相关度。Vjm对应第j个词和第m个词义的相关度。Σlm对应第l个主题和第m个词义的相关度。

也可以反过来解释：我们输入的有m个词，对应n个文本。而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。Σlm对应第l个词义和第m个主题的相关度。

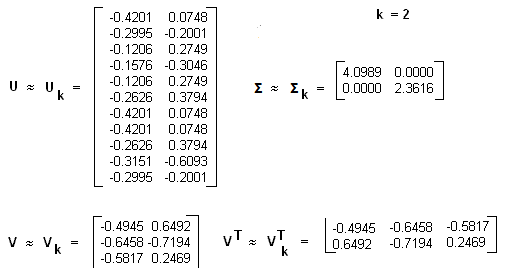
这样通过一次SVD，就可以得到文档和主题的相关度，词和词义的相关度以及词义和主题的相关度。

举例

假设有下面10个词三个文本的词频TF对应矩阵如下



假定对应的主题数为2，则通过SVD降维后得到的三矩阵为：



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。

相似度计算

上面我们通过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 ：



总结

LSI是最早出现的主题模型，它的算法原理很简单，一次奇异值分解就可以得到主题模型，同时解决词义的问题，非常漂亮。但是LSI有很多不足，导致它在当前实际的主题模型中已基本不再使用。

主要的问题有：

1）SVD计算非常的耗时，尤其是我们的文本处理，词和文本数都是非常大的，对于这样的高维度矩阵做奇异值分解是非常难的。

2）主题值的选取对结果的影响非常大，很难选择合适的k值。

3）LSI得到的不是一个概率模型，缺乏统计基础，结果难以直观的解释。

对于问题1），主题模型非负矩阵分解（NMF）可以解决矩阵分解的速度问题。对于问题2），这是老大难了，大部分主题模型的主题的个数选取一般都是凭经验的，较新的层次狄利克雷过程（HDP）可以自动选择主题个数。对于问题3），牛人们整出了pLSI(也叫pLSA)和隐含狄利克雷分布(LDA)这类基于概率分布的主题模型来替代基于矩阵分解的主题模型。

回到LSI本身，对于一些规模较小的问题，如果想快速粗粒度的找出一些主题分布的关系，则LSI是比较好的一个选择，其他时候，如果你需要使用主题模型，推荐使用LDA和HDP。

收获：

首先，感谢助教大佬，共享了测试demo。我一脸懵逼时，仔细学习了一下他的代码。才知道处理NLP的主要步骤就是将其转换成词向量，然后应用模型。所以，baseline就在他demo的基础上运用了Day6里的TFIDF进行文本向量化，跑出来结果0.76还不错~

第二，感谢队友大腿，在我可怜巴巴只知道两个特征时，分享了他的语料和参考网址。才意识到要特征多一些，准确率才能上去。所以，开始各处扒拉特征。

第三，感谢群里小伙伴的热烈讨论。学习到解决这个问题的两个方案：神经网络和特征+传统模型。选择了后者，因为神经网络没怎么学过。最后选了11个特征+三个模型，得分上了0.81。

第四，第一次接触NLP，第一次完成一个小项目，这种赶在deadline前完成任务的感觉很开心。但是因为自己学习的不够多，所以没什么经验分享给大家，下周一定努力学习。

不足之处：最后觉得堆特征太麻烦了，就弃疗了。后来意识到特征工程是项目实现的重要一步，要全面把握各个特征，才能取得好成绩。

【POS tagging】

part-of-speech tagging , or word classes or lexical categories . 说法很多其实就是词性标注。

http://blog.csdn.net/fxjtoday/article/details/5841453

【N-gram】

http://blog.csdn.net/baimafujinji/article/details/51281816

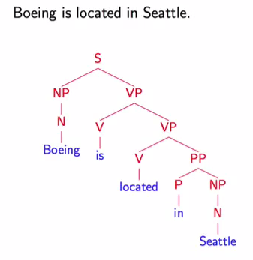
【Parsing语法解析】

http://blog.csdn.net/lanxu\_yy/article/details/37700841

语法解析有两个主要的问题：

- 句子语法在计算机中的表达与存储方法，以及语料数据集；

用树状结构图来表示，如下图所示：S表示句子；NP、VP、PP是名词、动词、介词短语（短语级别）；N、V、P分别是名词、动词、介词。



(S (NP (N Boeing)) (VP (V is) (VP (V located) (PP (P in) (NP (N Seattle))))))

- 语法解析的算法。

上下文无关语法（Context-Free Grammer）：

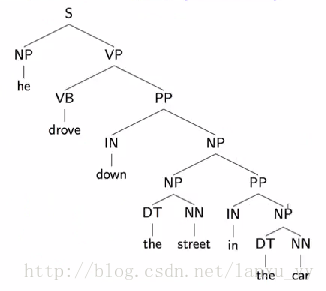
1）N表示一组非叶子节点的标注，例如{S、NP、VP、N...}

2）Σ表示一组叶子结点的标注，例如{boeing、is...}

3）R表示一组规则，每条规则可以表示为X->Y1Y2...Yn，X∈N，Yi∈(N∪Σ)

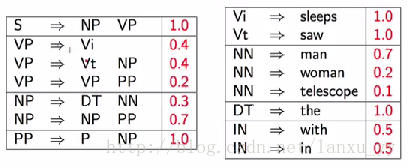
4）S表示语法树开始的标注

这种上下文无关的语法可以很容易的推导出一个句子的语法结构，但是缺点是推导出的结构可能存在二义性。例如下面两张图中的语法树都可以表示同一个句子。常见的二义性问题有：1）单词的不同词性，如can一般表示“可以”这个情态动词，有时表示罐子；2）介词短语的作用范围，如VP PP PP这样的结构，第二个介词短语可能形容VP，也可能形容第一个PP；3）连续的名字，如NN NN NN。



概率分布的上下文无关语法（Probabilistic Context-Free Grammar）

由于语法的解析存在二义性，我们就需要找到一种方法从多种可能的语法树种找出最可能的一棵树。一种常见的方法既是PCFG （Probabilistic Context-Free Grammar）。如下图所示，除了常规的语法规则以外，我们还对每一条规则赋予了一个概率。对于每一棵生成的语法树，我们将其中所有规则的概率的乘积作为语法树的出现概率。



当我们或得多颗语法树时，我们可以分别计算每颗语法树的概率p(t)，出现概率最大的那颗语法树就是我们希望得到的结果，即arg max p(t)。

【训练算法】

我们已经定义了语法解析的算法，而这个算法依赖于CFG中对于N、Σ、R、S的定义以及PCFG中的p(x)。上文中我们提到了Penn Treebank通过手工的方法已经提供了一个非常大的语料数据集，我们的**任务**就是从语料库中训练出PCFG所需要的参数。

1）统计出语料库中所有的N与Σ；

2）利用语料库中的所有规则作为R；

3）针对每个规则A -> B，从语料库中估算p(x) = p(A -> B) / p(A)；

在CFG的定义的基础上，我们重新定义一种叫Chomsky的语法格式。这种格式要求每条规则只能是X -> Y1 Y2或者X -> Y的格式。实际上Chomsky语法格式保证生产的语法树总是二叉树的格式，同时任意一棵语法树总是能够转化成Chomsky语法格式。

【预测算法】

假设我们已经有一个PCFG的模型，包含N、Σ、R、S、p(x)等参数，并且语法树总数Chomsky语法格式。当输入一个句子x1, x2, ... , xn时，我们要如何计算句子对应的语法树呢？

* 暴力遍历，每个单词x可能有m = len(N)种取值，句子长度是n，每种情况至少存在n个规则，所以在时间复杂度O(m\*n\*n)的情况下，我们可以判断出所有可能的语法树并计算出最佳的那个。
* 动态规划，我们定义w[i, j, X]是第i个单词至第j个单词由标注X来表示的最大概率。直观来讲，例如xi, xi+1, ... , xj，当X=PP时，子树可能是多种解释方式，如(P NP)或者(PP PP)，但是w[i, j, PP]代表的是继续往上一层递归时，我们只选择当前概率最大的组合方式。特殊情况下，w[i, i, X] = p(X -> xi)。因此，动态规划的方程可以表示为w[i, j, X] = max (p(X -> Y Z) \* w(i, s, Y) \* w(s+1, j, Z))。关于动态规划方法，leetcode里有不少案例可以说明。

PCFG也有一些缺点，例如：1）缺乏词法信息；2）连续短语（如名词、介词）的处理等。但总体来讲它给语法解析提供了一种非常有效的实现方法。

【textstat】

Calculate statistical features from text

from textstat.textstat import textstat

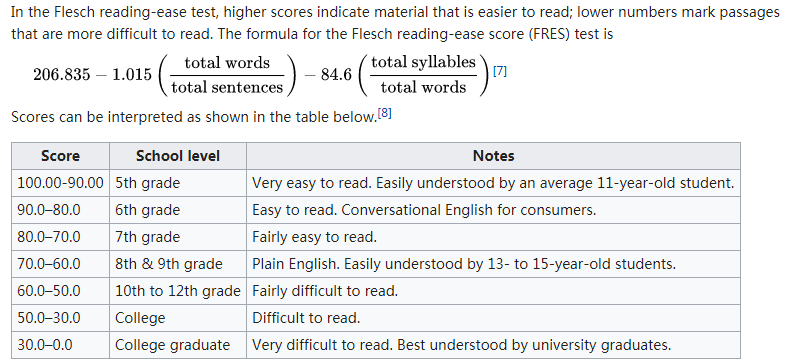
syllable\_count(text)

lexicon\_count(text, TRUE/FALSE)

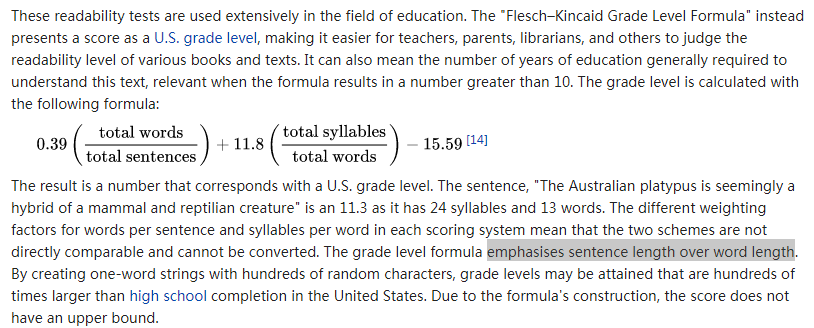
sentence\_count(text)

flesch\_reading\_ease(text)

returns the Flesch Reading Ease Score. Following table is helpful to access the ease of readability in a document.

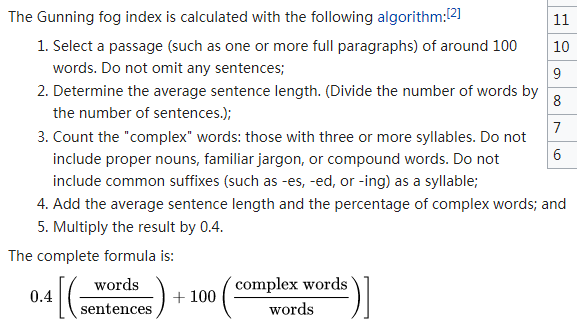


flesch\_kincaid\_grade(text)

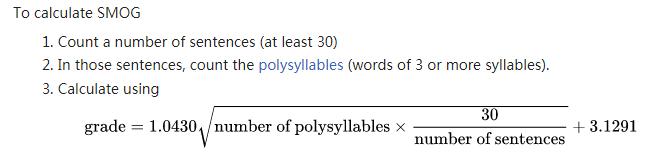


gunning\_fog(text)

The index estimates the years of formal education a person needs to understand the text on the first reading. For instance, a fog index of 12 requires the reading level of a U.S. high school senior (around 18 years old).



smog\_index(text)



automated\_readability\_index(text)



1 5-6 Kindergarten

2 6-7 First Grade

3 7-8 Second Grade

4 8-9 Third Grade

5 9-10 Fourth Grade

6 10-11 Fifth Grade

7 11-12 Sixth Grade

8 12-13 Seventh Grade

9 13-14 Eighth Grade

10 14-15 Ninth Grade

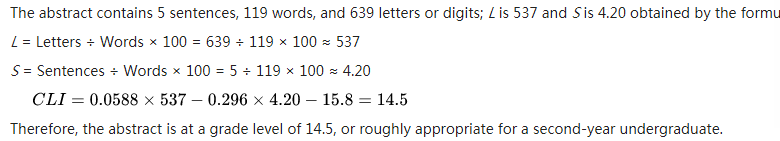
11 15-16 Tenth Grade

12 16-17 Eleventh grade

13 17-18 Twelfth grade

14 18-22 College

coleman\_liau\_index(text)



linsear\_write\_formula(text)

returns the grade level using the Linsear Write Formula

dale\_chall\_readability\_score(text)

Different from other tests, since it uses a lookup table of the most commonly used 3000 English words.Thus it returns the grade level using the New Dale-Chall Formula.

text\_standard(text)

Based upon all the above tests returns the best grade level under which the given text belongs to.

difficult\_words(test\_data)

【readability】

Measure the readability of a given text using surface characteristics

These measures are basically linear regressions based on the number of words, syllables, and sentences.

text = "A tokenized sentence .\nAnother sentence ."

result = getmeasures(text)

OrderedDict([('readability grades',

OrderedDict([('Kincaid', 8.985000000000003), ('ARI', 10.905999999999999),

('Coleman-Liau', 11.171300600000006), ('FleschReadingEase', 35.09750000000002),

('GunningFogIndex', 9.0), ('LIX', 82.5), ('SMOGIndex', 6.872983346207417),

('RIX', 2.0)])),

('sentence info',

OrderedDict([('characters\_per\_word', 6.6),

('syll\_per\_word', 2.0), ('words\_per\_sentence', 2.5), ('sentences\_per\_paragraph', 2.0),

('type\_token\_ratio', 0.8), ('characters', 33), ('syllables', 10), ('words', 5),

('wordtypes', 4), ('sentences', 2), ('paragraphs', 1), ('long\_words', 4),

('complex\_words', 1)])),

('word usage',

OrderedDict([('tobeverb', 0), ('auxverb', 0),('conjunction', 0),

('pronoun', 0), ('preposition', 0), ('nominalization', 2)])),

('sentence beginnings',

OrderedDict([('pronoun', 0), ('interrogative', 0), ('article', 1),

('subordination', 0), ('conjunction', 0), ('preposition', 0)]))])

【卷积神经网络】

https://morvanzhou.github.io/tutorials/machine-learning/tensorflow/5-03-A-CNN/

**卷积**

神经网络不再是对每个像素的输入信息做处理了,而是图片上每一小块像素区域进行处理, 这种做法加强了图片信息的连续性. 使得神经网络能看到图形, 而非一个点. 这种做法同时也加深了神经网络对图片的理解.

具体来说, 卷积神经网络有一个批量过滤器, 持续不断的在图片上滚动收集图片里的信息,每一次收集的时候都只是收集一小块像素区域, 然后把收集来的信息进行整理, 这时候整理出来的信息有了一些实际上的呈现, 比如这时的神经网络能看到一些边缘的图片信息, 然后在以同样的步骤, 用类似的批量过滤器扫过产生的这些边缘信息, 神经网络从这些边缘信息里面总结出更高层的信息结构,比如说总结的边缘能够画出眼睛,鼻子等等. 再经过一次过滤, 脸部的信息也从这些眼睛鼻子的信息中被总结出来. 最后我们再把这些信息套入几层普通的全连接神经层进行分类, 这样就能得到输入的图片能被分为哪一类的结果了.

**池化**(pooling)

在每一次卷积时, 神经层可能会无意地丢失一些信息. 这时, 池化 (pooling) 就可以很好地解决这一问题. 而且池化是一个筛选过滤的过程, 能将 layer 中有用的信息筛选出来, 给下一个层分析. 同时也减轻了神经网络的计算负担 (具体细节参考). 也就是说在卷积的时候, 我们不压缩长宽, 尽量地保留更多信息, 压缩的工作就交给池化了,这样的一项附加工作能够很有效的提高准确性. 有了这些技术,我们就可以搭建一个属于我们自己的卷积神经网络啦.

**流行的 CNN 结构**

从下到上的顺序, 首先是输入的图片(image), 经过一层卷积层 (convolution), 然后在用池化(pooling)方式处理卷积的信息, 这里使用的是 max pooling 的方式. 然后再经过一次同样的处理, 把得到的第二次处理的信息传入两层全连接的神经层 (fully connected),这也是一般的两层神经网络层,最后在接上一个分类器(classifier)进行分类预测. 这仅仅是对卷积神经网络在图片处理上一次简单的介绍.

我们的一般的神经网络在理解图片信息的时候还是有不足之处, 这时卷积神经网络就是计算机处理图片的助推器. Convolutional Neural Networks (CNN) 是神经网络处理图片信息的一大利器. 有了它, 我们给计算机看图片,计算机理解起来就更准确.

卷积神经网络包含输入层、隐藏层和输出层，隐藏层又包含卷积层和pooling层，图像输入到卷积神经网络后通过卷积来不断的提取特征，每提取一个特征就会增加一个feature map，所以会看到视频教程中的立方体不断的增加厚度，那么为什么厚度增加了但是却越来越瘦了呢，哈哈这就是pooling层的作用喽，pooling层也就是下采样，通常采用的是最大值pooling和平均值pooling，因为参数太多喽，所以通过pooling来稀疏参数，使我们的网络不至于太复杂。

《使用 Keras 手把手介绍神经网络构建》

http://yangguang2009.github.io/2016/11/27/deeplearning/develop-neural-network-model-with-keras-step-by-step/

《深度学习的activation function》

<https://zhuanlan.zhihu.com/p/25110450>

优先使用ReLU (Rectified Linear Unit) 函数作为神经元的activation function:

ReLU线性整流函数 ReLU = max(0,x)

【one-hot编码】

独热编码，又称一位有效编码，其方法是使用N位状态寄存器来对N个状态进行编码，每个状态都由他独立的寄存器位，并且在任意时候，其中只有一位有效。

例如：

自然状态码为：000,001,010,011,100,101

独热编码为：000001,000010,000100,001000,010000,100000

可以这样理解，对于每一个特征，如果它有m个可能值，那么经过独热编码后，就变成了m个二元特征（如成绩这个特征有好，中，差变成one-hot就是100, 010, 001）。并且，这些特征互斥，每次只有一个激活。因此，数据会变成稀疏的。

这样做的好处主要有：

解决了分类器不好处理属性数据的问题

在一定程度上也起到了扩充特征的作用

《主观题自动评分（一）：神经网络上篇》

<https://zhuanlan.zhihu.com/p/30718570>

https://github.com/keras-team/keras/blob/master/examples/pretrained\_word\_embeddings.py