语言模型

# 一、语言模型的概念

统计语言模型是**词序列上的概率分布**。给定这样的序列，例如长度m，它将概率P(W1,……,Wn)分配给整个序列。有一种估计不同短语的相对可能性的方法在许多自然语言处理应用程序中是有用的，特别是生成文本作为输出的应用程序。语言建模用于语音识别，机器翻译，词性标注，解析，手写识别，信息检索等应用。

在语音识别中，计算机尝试将声音与词语序列进行匹配。语言模型提供上下文以区分听起来相似的单词和短语。例如，在美国英语中，短语"recognize speech"和"wreck a nice beach"发音几乎相同，但意思非常不同。当来自语言模型的证据与发音模型和声学模型结合在一起时，这些模糊性更容易解决。

**数据稀疏**是构建语言模型的主要问题。在训练中将不会观察到最可能的词语序列。一个解决方案是假设**一个单词的概率只取决于它前面的n个单词**。这是众所周知的**N-gram模型**或一元语法模型当n = 1。

# 二、语言模型原理

假设S为某一个句子，由一连串特定顺序排列的词w1,w2,w3,...,wn构成，n为S的句子长度。

**p(S) = p(w1,w2,w3,...,wn) = p(w1) x p(w2|w1) x p(w3|w1, w2) x ... x p(wn|w1, w2, w3,...,wn-1)**

即**p(w1)表示w1这个词出现的概率**，p(w2|w1)表示词w2在词w1之后出现的概率，以此类推。根据**马尔科夫模型**，某一时刻的状态只与前一时刻的状态有关，而与其他时刻无关。用于统计语言模型中，可表述为词wi只与它前一个词wi-1有关，而与其他词无关，这样概率模型采用马尔可夫模型之后得到.

**二元模型（Bigram Model）:** 下一个词的出现依赖它前面的一个词

**p(S) = p(w1,w2,w3,...,wn) = p(w1) x p(w2|w1) x p(w3| w2) x ... x p(wn|wn-1)**

**三元模型（trigram model）**: 下一个词的出现依赖它前面的两个词

**p(S)=p(w1)p(w2|w1)p(w3|w1,w2)...p(wn|wn-1,wn-2)**

如果假设**一个词由它前面N-1个词决定，则被称为N元模型**。

二元模型：估计条件概率p(wi|wi-1)：

（1） 选择大量合适的语料库（机读文本），数量为#；

（2） 统计词wi-1出现的次数#(wi-1)；（ #(wi-1)表示wi-1的数量 ）

（3） 统计词wi-1,wi前后相邻出现的次数#(wi-1,wi)；

（4） 计算词语二元组的相对频度：f(wi-1) = #(wi-1)/#；f(wi-1,wi) = #(wi-1,wi)/#。

（5） 根据大数定理，**只要统计量足够，相对频度等于概率**，则p(wi-1) ≈ #(wi-1)/#；p(wi-1,wi) ≈ #(wi-1,wi)/#。

（6） 根据条件概率公式计算条件概率p(wi|wi-1) = p(wi-1,wi)/p(wi-1) =  #(wi-1,wi)/#(wi-1)。

当然对于一个语言模型**还要考虑很多问题**，远不止这么简单。还有很多问题要解决：

（1）零概率问题；

（2）句子中词只与前一个词相关似乎过于简化，对于高阶语言模型，多少阶合适；

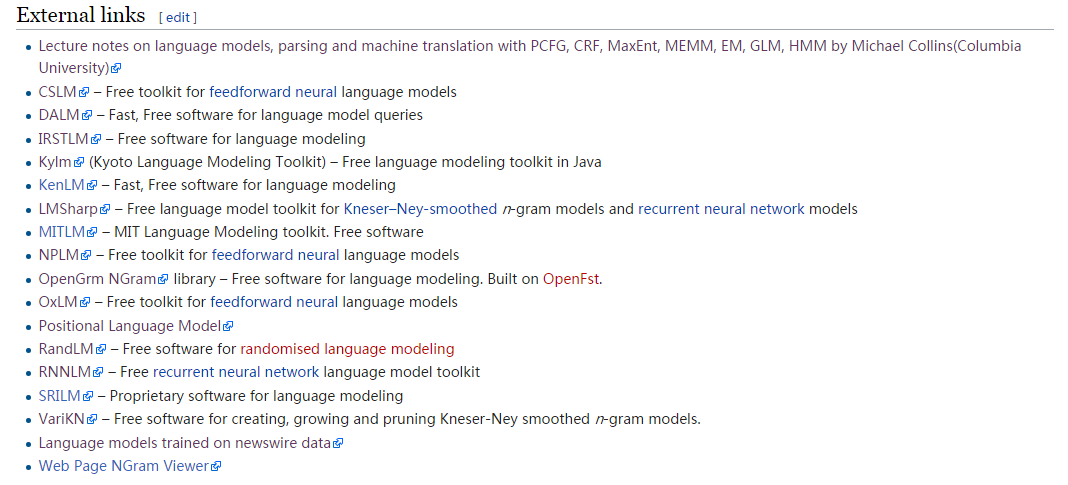
（3）语料库的设计。

要解决这些问题，可以参考一下链接：

<http://blog.csdn.net/xmdxcsj/article/details/50373554>（解决零概率算法）

<http://blog.csdn.net/xmdxcsj/article/details/50321613>（解决模型阶数算法）

# 三、语言模型常用开源工具



## srilm语言模型环境搭建

**环境简述**：Ubuntu14.04, docker1.12.6

srilm工具，下载**地址**：http://www.speech.sri.com/projects/srilm/download.html

安装 tcl，sudo apt-get install tcl.

安装相关工具：sudo apt-get install make gawk gzip bzip2 p7zip

**安装srilm依赖的包：**

1）解压srilm包,路径为/opt/yangzhanku/software/srilm。解压命令：tar –zxvf srilm-1.7.2.tar.gz

2）**修改srilm/MakeFile**：

　修改或在第7行下面加上一行

　# SRILM = /home/speech/stolcke/project/srilm/devel (原）

　 SRILM = /opt/yangzhanku/software/srilm

3）**再修改srilm/common/Makefile.machine**.\*\*\*\*\*:\*\*\*\*\*所填的内容和本机硬件平台有关。可以在终端输入一下命令查看： uname -i

比如我的机子是x86\_64，那我修改的是Makefile.machine.i686-m64这个文件。

找到：

　　　　TCL\_INCLUDE =

　　　　TCL\_LIBRARY =

修改为：

　　　　TCL\_INCLUDE =

　　　　TCL\_LIBRARY =

NO\_TCL = X

找到：

　　　　GAWK = /usr/bin/awk

修改为：

　 GAWK = /usr/bin/gawk

4）**编译SRILM**

srilm目录下输入

make World

5）**修改环境变量**

在终端输入

export PATH=/opt/yangzhanku/software/srilm /bin/:/opt/yangzhanku/software/srilm /bin/i686-m64/:$PATH

这个地址要看自己的安装位置，因人而异

6） 测试

在终端输入依次输入一下命令：

make test

# 四、使用方法

## 1、小数据

假设有去除特殊符号的训练文本trainfile.txt，以及测试文本testfile.txt，那么训练一个语言模型以及对其进行评测的步骤如下：

1：**词频统计**

ngram-count -text trainfile.txt -order 3 -write trainfile.count

**例子**： **备注：**trainfile.txt 分词后的，但是一行不能太多分词

**bin/i686-m64/ngram-count -text /opt/yangzhanku/result/test/words\_20.txt -order 3 -write /opt/yangzhanku/result/test/word.count**

其中-order 3为3-gram，trainfile.count为统计词频的文本

2：**模型训练**

ngram-count -read trainfile.count -order 3 -lm trainfile.lm -interpolate –kndiscount

**例子：备注： 命令需要在txt，如果直接从word中复制过来，就不会报错语言模型。**

**bin/i686-m64/ngram-count -read /opt/yangzhanku/result/test/word.count -order 3 –lm /opt/yangzhanku/result/test/dict.lm -interpolate –kndiscount | tee /opt/yangzhanku/result/test/dict1.lm**

**bin/i686-m64/ngram-count -read /opt/yangzhanku/result/test/word.count -order 3 –lm /opt/yangzhanku/result/test/dict.lm | tee /opt/yangzhanku/result/test/dict1.lm**

其中trainfile.lm为生成的语言模型，-interpolate和-kndiscount为插值与折回参数

3：**测试（困惑度计算）**

ngram -ppl testfile.txt -order 3 -lm trainfile.lm -debug 2 > file.ppl

**例子：**

**bin/i686-m64/ngram -ppl /opt/yangzhanku/result/test/newsgroup.txt -order 3 -lm /opt/yangzhanku/result/test/dict1.lminterpolate -debug 2 > /opt/yangzhanku/result/test/result.ppl**

其中testfile.txt为测试文本，-debug 2为对每一行进行困惑度计算，类似还有-debug 0 , -debug 1, -debug 3等，最后 将困惑度的结果输出到file.ppl。

## 2、大数据（BigLM）

对于大文本的语言模型训练不能使用上面的方法，主要思想是将文本切分，分别计算，然后合并。步骤如下：

1：**切分数据**

split -l 10000 trainfile.txt filedir/

split -l 10000 newsgroup.txt 1/

即每10000行数据为一个新文本存到filedir目录下。

2：**对每个文本统计词频**

make-bath-counts filepath.txt 1 cat ./counts -order 3

其中filepath.txt为切分文件的全路径，可以用命令实现：ls $(echo $PWD)/\* > filepath.txt，将统计的词频结果存放在counts目录下

3：**合并counts文本并压缩**

merge-batch-counts ./counts

不解释

4：**训练语言模型**

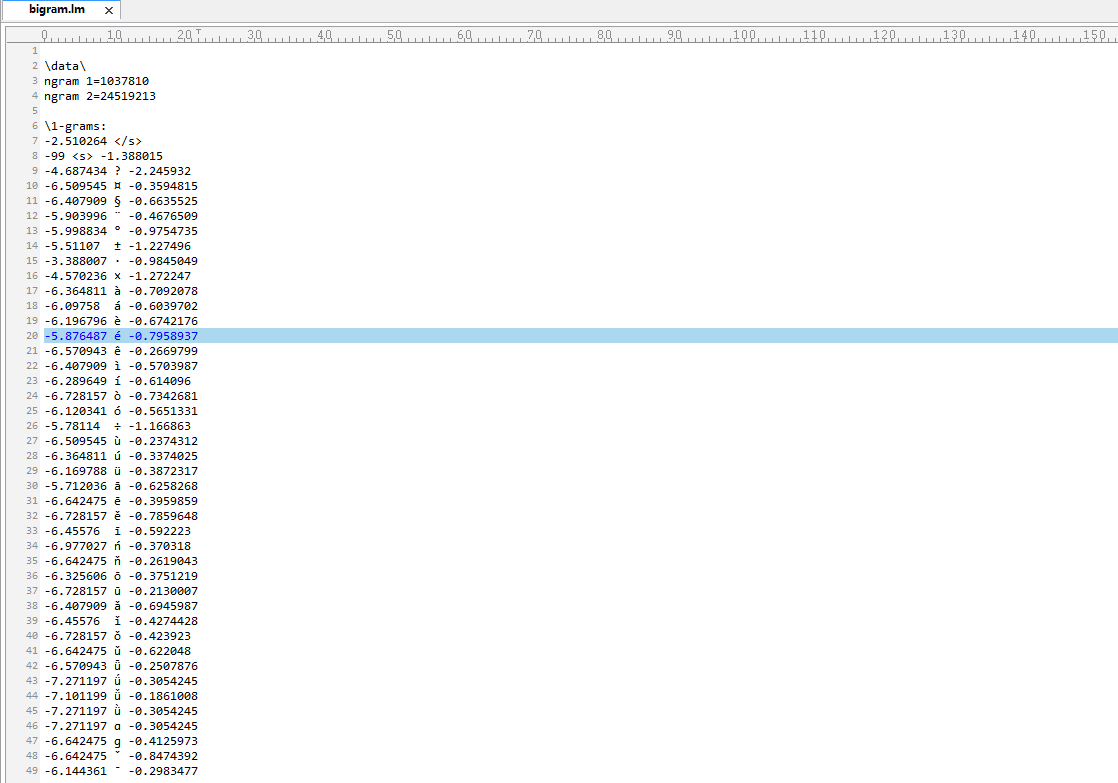
make-big-lm -read ../counts/\*.ngrams.gz -lm ../split.lm -order 3

用法同ngram-counts

5: **测评（计算困惑度）**

ngram -ppl filepath.txt -order 3 -lm split.lm -debug 2 > file.ppl

# 五、语言模型结果说明



其中文件格式：

log10(f(a\_z)) <tab> a\_z <tab> log10(bow(a\_z))

注：f(a\_z)是条件概率即P(z|a\_)，bow(a\_z)是**回退权重**

第一列表示**以10为底对数的条件概率P(z|a\_)**

第二列是n元词

第三列是以10为底的对数回退权重(它为未看见的n+1元词贡献概率)

# 六、参考资料：

<https://en.wikipedia.org/wiki/Language_model>

<http://blog.csdn.net/a635661820/article/details/43939773>

<http://blog.csdn.net/chenlei0630/article/details/23916921>

<http://blog.csdn.net/atcmy/article/details/53780619>

<http://blog.csdn.net/u011500062/article/details/50781101>

# 备注：

<http://blog.csdn.net/chenlei0630/article/details/23916921>

## 语言模型的格式：如dict.lm （language model）

1. \data
2. ngram 1=nr # 一元语言模型
3. ngram 2=nr # 二元语言模型
4. ngram 3=nr # 三元语言模型
6. \1-grams:
7. pro\_1 word1 back\_pro1
9. \2-grams:
10. pro\_2 word1 word2 back\_pro2
12. \3-grams:
13. pro\_3 word1 word2 word3
15. \end\

**备注**：

1. 如果是三元语言模型，则语言模型的文件中就包含一元、二元、三元的模型；
2. 求词的概率：

**import kenlm**

**model = kenlm.Model('/opt/yangzhanku/result/bigram.lm')**

**print (model.score('任命 无党派', bos=False, eos=False))**

**#res=p(任命,无党派) = p(任命)\*p(无党派|任命) 使用二元模型计算“任命”、“无党派”的联合概率；模型中的第一列是概率取对数，目的是在求联合概率时将乘变成加。**

**print (model.score('任命', bos=False, eos=False))**

**#res = p(任命) 使用一元模型计算，其实就是直接使用第一列的概率**

1. **语言模型的最后一列是回退概率取对数，如果该词语言模型没有就会使用回退概率。比如：**

比如P(平|习，进）的意思是已经出现了“习进”两个字，后面是平的概率，这个概率这么计算：

1. if(存在(word1,word2,word3)的三元模型){
3. return pro\_3(word1,word2,word3) ;
5. }else if(存在(word1,word2)二元模型){
7. return **back\_pro2**(word1,word2)\*P(word3|word2) ;
9. }else{
11. return P(word3 | word2);
13. }