自然语言处理导论

#L2

N-gram语言模型

袁彩霞

yuancx@bupt.edu.cn

智能科学与技术中心

语言模型

- 语言模型能解决的问题：How likely is a string of words good language?
  - the house is big ! good
  - house big is the ! bad
  - the house is xxl ! worse
- 可以用于词的排序：the is small house
  - the house is small 优于 small the is house
- 可以用于词的选择：I am going ___ (home/house)
  - I am going home 优于 I am going house
- 其它用途：
  - 语音识别 (Speech Recognation)
  - 机器翻译 (Machine Translation)
  - 字符识别(OCR, Optical character recognition)
  - 手写字体识别(Handwriting recognition)
  - .....
- 如何对以上问题进行建模？

语言模型

- n-gram语言模型
  - 模型参数
  - 数据稀疏及平滑
- 模型质量评价
- 神经网络语言模型

语言模型

- n-gram语言模型
  - 模型参数
  - 数据稀疏及平滑
- 模型质量评价
- 神经网络语言模型

从概率的角度看

- 目标：评价词串$W = w_1, w_2, w_3, \ldots, w_n$的质量
- 从概率的角度看：计算概率$p(W)$

$$p(W)=p(w_1,w_2,w_3,\ldots,w_n)$$

- 根据链式法则：

$$p(W)=p(w_1)p(w_2|w_1)p(w_n|w_1,w_2,w_3,\ldots,w_{n-1})$$

  - 其中，$w_1,w_2,w_3,\ldots,w_{i-1}$为第i个词的历史词
  - 链式法则将联合概率分而治之

- 例句： likely connects audiences with content

p(likely connects audiences with content)

=p(likely|sentence start) × p(connects|likely) × p(audience|likely, connects) × p(with|likely, connects, audience) × p(content|likely, connects audience, with)

概率语言模型

- 广义上讲：用于计算p(W)或$p(w_n|w_1, \ldots, w_{n-1})$的模型均可以称为（概率）语言模型

- 问题：假设给定**语料库**（大量的语言使用数据），如何计算以上条件概率？

概率语言模型

- 最简单的方式：极大似然估计(Maximum Likelihood Estimation, MLE)：

$$p(w_m \mid w_1..w_{m-1}) = \frac{C(w_1..w_{m-1}, w_m)}{C(w_1..w_{m-1})}$$

$$C(w_1..w_{m-1}) = \sum_{w_j \in W} C(w_1..w_{m-1}, w_j)$$

- 例如，采用MLE估计：

  p(content|likely, connects, audience, with)

  =C(likely connects audience with contents) / C(likely connects audience with)

概率语言模型

- 该方法背后隐藏的假设：当前词出现的概率依赖于它前面的词
- 然而，如果考虑的前面的词个数很大：
  - 一方面，语料库中$C(w_1, \ldots, w_i)$或$C(w_1, \ldots, w_{i-1})$极有可能为零，这样就导致条件概率为零或无法计算
  - 另一方面，随着历史长度的增加，不同的历史数目按指数级增长
    - 如果历史长度为i，则有$|V|^{i-1}$个不同的历史，而我们必须考虑在所有不同的历史下产生第i个词的概率

概率语言模型

- 一个可能的方案：当前词的出现概率仅依赖于较短的历史词
  - P(content|connects, audience, with)
    - 进行如上的统计
    - 如果C(connects audience with contents)=0
    - 则 ➔
  - P(content|audience, with)
  - P(content|with)
  - P(content)

n-gram 模型

- 马尔科夫假设：
  - 位于某个特定状态的概率取决于前n-1个状态

  $P(w_i|w_1, w_2, \ldots, w_{i-1}) \approx P(w_i|w_{i-n+1}, \ldots, w_{i-1})$

  - ***n-1*阶马尔科夫链**

- 马尔科夫假设应用于语言模型：假设每个词的出现仅取决于它前面的n-1个词
- -->n-gram模型（n元语法或n元文法）

n-gram 模型

- 1-gram 语言模型 (unigram):

  $p(w_1,w_2,w_3, ...,w_n) \approx p(w_1)\ p(w_2)\ p(w_3)...p(w_n)$

  – E.g., p(please close the door) = p(please)p(close)…p(door)

- 2-gram 语言模型 (bigrams):

  $p(w_1,w_2,w_3, ...,w_n) \approx p(w_1)\ p(w_2|w_1)\ p(w_3|w_2)$

  $...p(w_n|w_{n-1})$

  (此处w_{n-1} 被称为历史词)

  - E.g., p(please close the door) = p(please|START)p(close|please)…p(END|door)

n-gram 模型

- 3-gram 语言模型 (trigrams):

$p(w_1, w_2, w_3, \ldots, w_n) \approx$

$p(w_1)\ p(w_2|w_1)\ p(w_3|w_1, w_2) \ldots p(w_n|w_{n-2}, w_{n-1})$

  - E.g., p(please close the door) = p(please|START)p(close|START, please)…p(END|the, door)

n-gram 模型

- 可以扩展到更高阶的模型：4-grams, 5-grams……
- 由于N-gram模型的马尔科夫假设，其对语言来说是一个“不充分”的模型
  - 因为语言本身是长距离相依的
- 但N-gram模型极大程度上满足了语言建模需要
  - 例如：There's a whole lot of laughing gas in the atmosphere these days
    - days←these
    - days←of???

n-gram模型的参数

- n-gram： n=?
- 概率：

$$P(w_m \mid w_1..w_{m-1}) = \frac{C(w_1..w_{m-1}, w_m)}{C(w_1..w_{m-1})}$$

n-gram模型的参数

- n-gram： n=?
- 一个实例：
  - 采用Shakespeare的著作作为统计数据，分别设定n=1, 2, 3, 4时，生成的不同句子的比较

n-gram模型的参数

Unigram	• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have • Every enter now severally so, let • Hill he late speaks; or! a more to leg less first you enter • Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like
Bigram	• What means, sir. I confess she? then all sorts, he is trim, captain. •Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. •What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman? •Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt
Trigram	• Sweet prince, Falstaff shall die. Harry of Monmouth's grave. • This shall forbid it should be branded, if renown made it empty. • Indeed the duke; and had a very good friend. • Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
Quadrigram	• King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; • Will you not tell me who I am? • It cannot be but so. • Indeed the short and the long. Marry, 'tis a noble Lepidus.

n-gram模型的参数

- 结论：n越大，生成的句子越好
- 然而：
  - n太大（比如n=10）是否可行？
  - n-gram模型中概率参数的个数
  - n-gram模型中概率参数的估计

1-gram — 0-th Markov

$$p(w_1 \ldots w_N) = \prod_{i=1}^{N} p(w_i)$$

  - 因此需要知道所有词w的p(W)
- p(.) 的个数取决于考虑的词汇个数
  - 对于10000个词，需要计算10000个参数
- 但1-gram假设词之间是相互独立的，模型过于简单
  - p(the the the the) >> p(I like ice cream)!

2-gram —— 1-th Markov

$$P(w_1 \ldots w_N) = P(w_1) \prod_{i=2}^{N} P(w_i \mid w_{i-1})$$

  - 因此需要知道任意两个词的p(.|.)
- p(.|.) 的个数取决于考虑的两元词汇组合个数
  - 对于10000个词，需要计算10000*10000=1.0e+8个参数

更高阶的模型...

- 3-gram：

$$P(w_1 \ldots w_N) = P(w_1) \prod_{i=2}^{N} P(w_i \mid w_{i-1}, w_{i-2})$$

  - 则需要计算任意三个词之间的p(.|..)
  - 参数个数：10000^3=1.0e+12
- 10-gram：
  - 参数个数：10000^{10}
  - 很难学习及维护如此大规模的模型
  - 会出现非常多的0值参数

一个实际的例子

- 以Gigaword (文本来自New York Times, Wall Street Journal, and news wire sources)中的1000万句子、约2.75亿个词作为统计数据

1-gram	716,706
2-gram	12,537,755
3-gram	22,174,483

- 同时，n-gram的个数随着语料规模的增大而增大
- 实际应用中，3-gram最为常用

n-gram模型的参数估计

- 最显而易见的方法：MLE统计相对概率

$$\hat{P}(w|w_{-1}) = \frac{c(w_{-1}, w)}{\sum_{w'} c(w_{-1}, w')}$$

- 一般过程：
  - 准备一个训练数据集X
  - 从X中计算模型的参数
  - 将计算得到的参数用于为测试数据集X'中的句子进行打分

198015222 the first
194623024 the same
168504105 the following
158562063 the world
…
14112454 the door

23135851162 the *

n-gram模型的参数估计

- Please close the door

198015222 the first
194623024 the same
168504105 the following
158562063 the world
…
14112454 the door

23135851162 the *

197302 close the window
191125 close the door
152500 close the gap
116451 close the thread
87298 close the deal

3785230 close the *

3380 please close the door
1601 please close the window
1164 please close the new
1159 please close the gate
900 please close the browser

13951 please close the *

$$p(door|closethe) = \frac{191125}{3785230}$$

$$p(door|the) = \frac{14112454}{2313585116\ 2}$$

$$p(door|\ pleaseclosethe) = \frac{3380}{13951}$$

语言模型

- n-gram语言模型
  - 模型参数
  - 数据稀疏及平滑
- 模型质量评价
- 神经网络语言模型

数据稀疏

- 在给定的训练数据集X中：
- 一些词或词组高频出现
- 一些词或词组低频出现
  - $C(w_j)/C(.)$; 而 C(.) 很小 (1,2,etc.)
    - 造成估计不可靠
    - 极大似然估计在采样样本总数趋于无穷的时候达到最小方差
- 同时也有一些词或词组在X中未可见
  - 未知词（Unkown words）或未知词组（Unkown n-grams）
  - $C(w_j)/C(.)$; 而C(.)=0
    - 无法进行计算！

数据稀疏

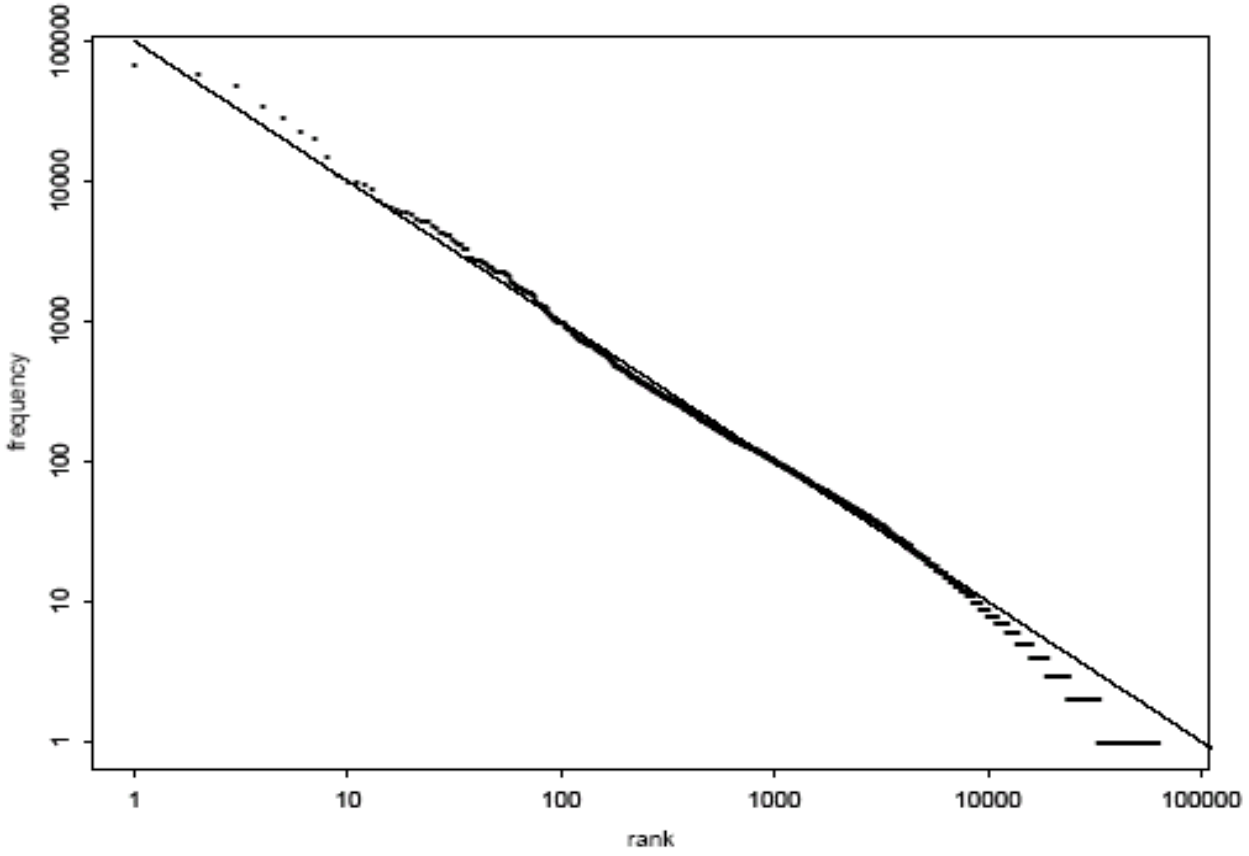
- 未知事件（Unknown events）：
  - 例如句子： s=they buy a new house
  - 如果一个bigram "*a new*"在训练语料中未出现过
    - 即： p(new|a) = 0
    - 则： p(s) ≈ p(they|"start") * p(buy|they) * p(a|buy) * p(new|a) * P(house|new) =.15 * .132 * .265 * .0 * .092 = 0
    - zeroes will propagate!
  - ... 但s是一个很好的句子

数据稀疏

- 数据稀疏的严重性：
  - 1983, Bahal: 150millions IBM patent corpus, estimated parameters for a 3-gram(training), then use to analyze anther corpus with same source. He found that 23% of 3-gram do not occur in training corpus.
  - 1992, Essen and Steinbiss: LOB corpus (millions) 75% as training, 25% as test, 12% of 2-gram do not occur in training corpus.
  - 1992, Brown and Dellapietra: 366millions as training, new test corpus, 14.7% new 3-gram
- 解决办法：
  - 扩大训练语料？

Zipf定律

- Zipf's law：在自然语言的语料库里，一个单词出现的频率与它在频率表里的排名成反比。
  - 频率最高的单词出现的频率大约是出现频率第二位的单词的2倍
  - 出现频率第二位的单词则是出现频率第四位的单词的2倍



数据稀疏

- 扩大训练语料规模有所帮助
  - 但增加的训练语料中高频词占据绝大部分

- 其它缓解数据稀疏的办法？

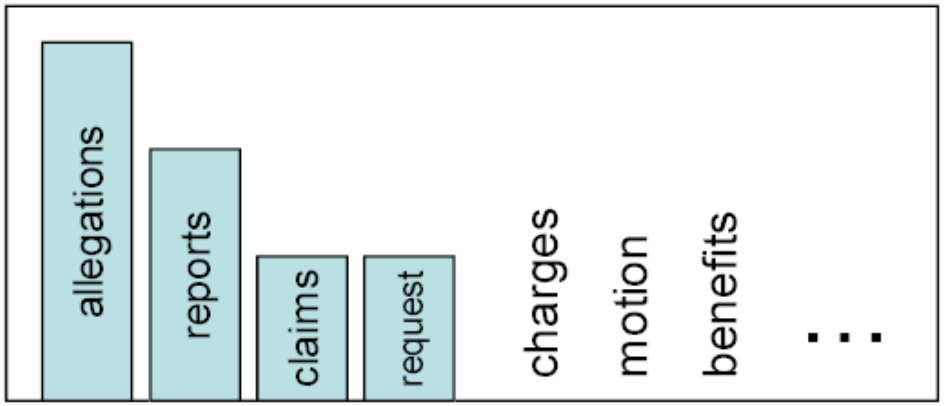
数据稀疏

- 平滑（smothing）：重新估计零概率及低值概率，赋予它们一个非零值
  - Add-one (Laplace Smoothing)
  - Good-Turing
  - …
- 回退（back-off）：高阶n-grams的概率难以计算时，采用较低阶的n-grams来统计
  - Backoff
  - Linear interpolation

Smoothing

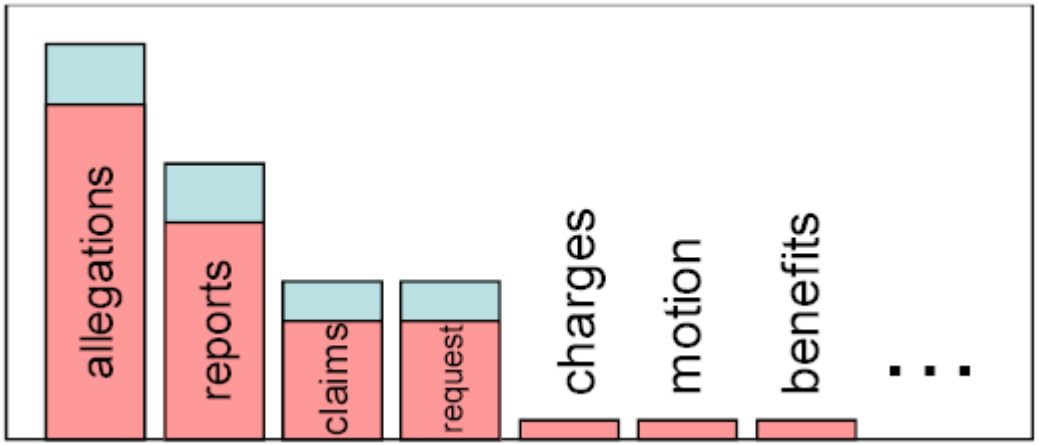
- 通常需要从稀疏数据中做估计

P(w | denied the)
3 allegations
2 reports
1 claims
1 request
7 total



- 平滑技术使得较锥形分布变得更平滑，因此具有更好的泛化能力

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total



Laplace Smoothing (Add-one)

- unigram:
  - 原始的MLE值 $p(w_i) = \frac{c_i}{N}$
    - c_i为词w_i在语料中出现的次数，N为语料中的token总数
  - Add-one $p_{Laplace}(w_i) = \frac{c_i + 1}{N + |V|}$
    - $|V|$为词表大小，分母+$|V|$的原因：归一化?

Laplace Smoothing (Add-one)

- unigram:
  - 令 $c_i^* = \frac{(c_i + 1)N}{N + |V|}$
  - 则 $p_{Laplace} = \frac{c_i^*}{N}$
  - c*可以看做是c_i的discount

Laplace Smoothing (Add-one)

- bigram:
  - 原始的MLE值 $p(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$
  - Add-one $p_{Laplace} = \frac{c(w_{i-1}, w_i)+1}{c(w_{i-1})+|V|}$
  - c*()是c()的discount: $c_i^*(w_{i-1}, w_i) = \frac{(c(w_{i-1}, w_i)+1)c(w_{i-1})}{c(w_{i-1})+|V|}$
  - 则 $p_{Laplace} = \frac{c^*(w_{i-1}, w_i)}{c(w_{i-1})}$

Laplace Smoothing (Add-one)

- 例子： 以Berkeley Restaurant Project数据为例
  - 9332个句子
  - 1446个不同的词
- 关注其中如下8个词之间的Bigram
  - i want to eat chinese food lunch spend

原始计数

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

加1计数

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

原始计数

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

加1折扣计数

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

原始概率

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

加1概率

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Laplace Smoothing (Add-one)

- Add-one 平滑对概率值的影响很大
  - E.g.: p(to|want) 从 .66 变化到 .26！（为什么？）
- 从训练语料观察到的n-grams中“移走”了太多的概率
  - more precisely: the ‘discount factor’
- Church and Gale(1991)的实验结果:
  - 46.5%的概率被移给了0概率的n-grams

Lidstone(add-Delta) smoothing

- Add-one平滑的一种扩展：

$$P(w_i \mid w_1 \ldots w_{i-1}) = \frac{c(w_1 \ldots w_i) + \delta}{c(w_1 \ldots w_{i-1}) + |V|\delta}$$

$$0 \le \delta \le 1$$

- delta=1时即为Laplace平滑
- 问题：如何选择较合适的参数delta?

Lidstone(add-Delta) smoothing

- 常用的方法：held-out estimation
  - 从训练数据D中分离出一部分数据H（held-out data, validation data）
  - 采用数据D-H训练具有不同delta值的语言模型
  - 分别测试其在数据H上的表现
  - 选取表现最好的模型对应的delta作为最优delta

Good-Turing Smoothing

- 基本思路:
  - 根据仅出现一次的unigram的个数，来确定那些未见unigram的概率
- 仅出现一次的unigram的个数:

$$N_1 = \sum_{w:count(w)=1} 1$$

- 出现c次的unigram个数:

$$N_c = \sum_{w:count(w)=c} 1$$

Good-Turing Smoothing

- 折扣：
  - 出现c次的折扣为出现c*：$c^* = (c+1)\frac{N_{c+1}}{N_c}$
  - 则出现c次的词的概率为：$p_c = \frac{c^*}{N} = \frac{(c+1)N_{c+1}}{N_c N}$
  - 出现0次的词在折扣后的出现次数：

$$c_0^* = (0+1)\frac{N_{0+1}}{N_0} = \frac{N_1}{N_0}$$

  - 则出现0次的词的概率为：

$$p_0 = \frac{c_0^*}{N} = \frac{N_1}{N_0 N}$$

Good-Turing Smoothing

- 对于n-gram W,
  - 如果其频次 $r>0$：
$$p(W)=\frac{r^*}{N}$$
  - 反之（即$r=0$时）：
$$p(W)=\frac{1-\sum_{r=1}^{\infty} n_r \frac{r^*}{N}}{n_0} \approx \frac{n_1}{n_0 N}$$
$$r^* = \frac{(r+1)n_{r+1}}{n_r}$$
  - 其中，n_r 表示出现r次的n-grams个数

Good-Turing Smoothing

Good-Turing的归一化特性

$$\sum_{c=0}^{\infty} n_c p_c = \sum_{c=0}^{\infty} n_c \frac{c^*}{N} = \sum_{c=0}^{\infty} n_c \frac{(c+1)\frac{n_{c+1}}{n_c}}{N}$$

$$= \sum_{c=0}^{\infty} \frac{(c+1)n_{c+1}}{N} = \frac{\sum_{c=0}^{\infty}(c+1)n_{c+1}}{N}$$

$$= \frac{\sum_{c=1}^{\infty} cn_c}{N} = 1$$

Good-Turing Smoothing

- 问题1：

$$p_0 = \frac{c_0^*}{N} = \frac{N_1}{N_0 N}$$

N_0？

- 问题2：

$$p_c = \frac{c^*}{N} = \frac{(c+1)N_{c+1}}{N_c N}$$

$N_{c+1} = 0$怎么办？

Backoff

- 基本思路：

$$p(w_i \mid w_{i-2}, w_{i-1}) = \begin{cases} p(w_i \mid w_{i-2}, w_{i-1}) & if \quad C(w_{i-2}, w_{i-1}, w_i) > 0 \\ \alpha_1 p(w_i \mid w_{i-1}) & if \quad C(w_{i-2}, w_{i-1}, w_i) = 0 \; and \; C(w_{i-1}, w_i) > 0 \\ \alpha_2 p(w_i) & otherwise \end{cases}$$

Interpolation

- 基本思路：

$$p(w_n \mid w_{n-2}, w_{n-1}) = \lambda_1 p(w_n) + \lambda_2 p(w_n \mid w_{n-1}) + \lambda_3 p(w_n \mid w_{n-2}, w_{n-1})$$

$$0 \le \lambda_i \le 1 \quad i = 1, 2, 3 \quad \lambda_1 + \lambda_2 + \lambda_3 = 1$$

Kneser-Ney Smoothing

- 回退中的问题
  - *回退总是选择高频词*
  - Shannon game: There was an unexpected ____?
    - delay? or Francisco?
    - 若“Francisco”较“delay”更常出现，则回退会选择“Francisco”
    - 但“Francisco”总是出现在“San”后面
  - 直观：*出现在较多上下文中的词更可能出现在新的上下文中*
- *K-N平滑：*以词w出现在新的上下文中的概率来取代w的出现概率（Context counting）
  - 对于一个词计算它的不同bigram的个数

$$p(w_i) \propto \sum_{w_{i-1}} c(w_{i-1}, w_i)$$

$$p(w_i) \propto | w_{i-1} : c(w_{i-1}, w_i) | > 0$$

语言模型

- n-gram语言模型
  - 模型参数
  - 数据稀疏及平滑
- 模型质量评价
- 神经网络语言模型