

## 2.3 n-gram语言模型

林洲汉  
上海交通大学

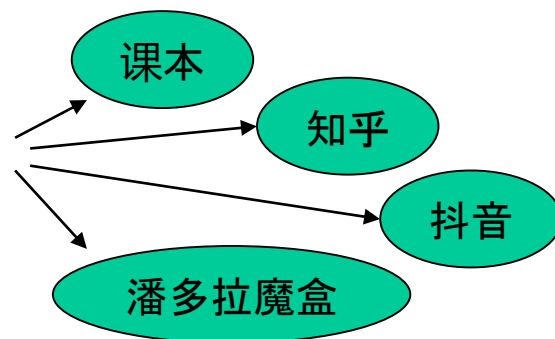
2023年秋季学期

- ▶ **什么是语言模型**
  - ▶ 语言模型的应用场景
  - ▶ 语言模型的种类
  - ▶ 如何评价语言模型的性能：混淆度 (perplexity)
- ▶ **N-gram语言模型**
  - ▶ N-gram的定义
  - ▶ 如何确定N-gram语言模型中的参数
  - ▶ 最大似然估计
  - ▶ Unigram、bigram示例
- ▶ **N-gram语言模型中的数据稀疏问题及解决办法**
  - ▶ 折扣法 (Discounting)
  - ▶ 回退 (Backoff) 及插值 (interpolation)

- ▶ **什么是语言模型**
  - ▶ 语言模型的应用场景
  - ▶ 语言模型的种类
  - ▶ 如何评价语言模型的性能：混淆度 (perplexity)
- ▶ **N-gram语言模型**
  - ▶ N-gram的定义
  - ▶ 如何确定N-gram语言模型中的参数
  - ▶ 最大似然估计
  - ▶ Unigram、bigram示例
- ▶ **N-gram语言模型中的数据稀疏问题及解决办法**
  - ▶ 折扣法 (Discounting)
  - ▶ 回退 (Backoff) 及插值 (interpolation)

- ▶ **语言建模 (language modeling)** 是一个学习任务;
- ▶ 在该任务中, 模型被要求给定上下文预测下一个词

学生们在课堂上打开了他们的\_\_\_\_\_



- ▶ 更正式的定义: 给定一个词序列

$$x^{(1)}, x^{(2)}, \dots, x^{(t)}$$

计算下一个词  $x^{(t+1)}$  的条件概率分布:

$$P(x^{(t+1)} | x^{(1)}, x^{(2)}, \dots, x^{(t)})$$


其中,  $x^{(t+1)}$  可以是词典  $V = \{w_1, w_2, \dots, w_{|V|}\}$  中任意词

- ▶ **做语言建模任务的系统称为语言模型 (language model)**

# 语言模型用于计算文本概率

- ▶ 语言模型可被用于**计算一段文本出现的概率**。
- ▶ 假设我们有一段文本  $x^{(1)}, x^{(2)}, \dots, x^{(T)}$ ，那么根据语言模型，这段文本的概率可计算为：

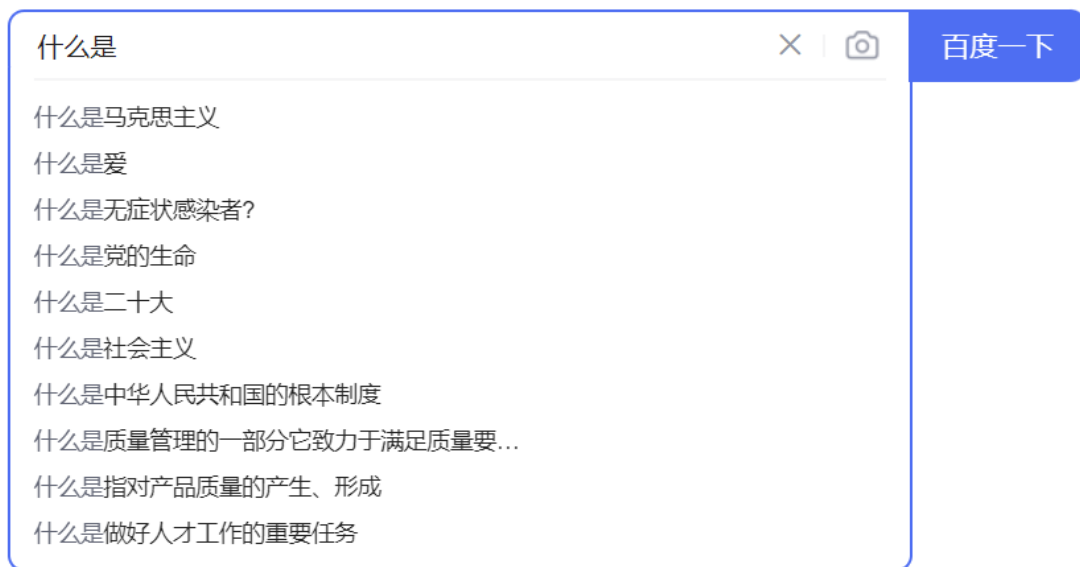
$$P(x^{(1)}, x^{(2)}, \dots, x^{(T)}) = P(x^{(1)}) \times P(x^{(2)} | x^{(1)}) \times \dots \times P(x^{(T)} | x^{(T-1)}, \dots, x^{(1)})$$

$$= \prod_{t=1}^T P(x^{(t)} | x^{(t-1)}, \dots, x^{(1)})$$


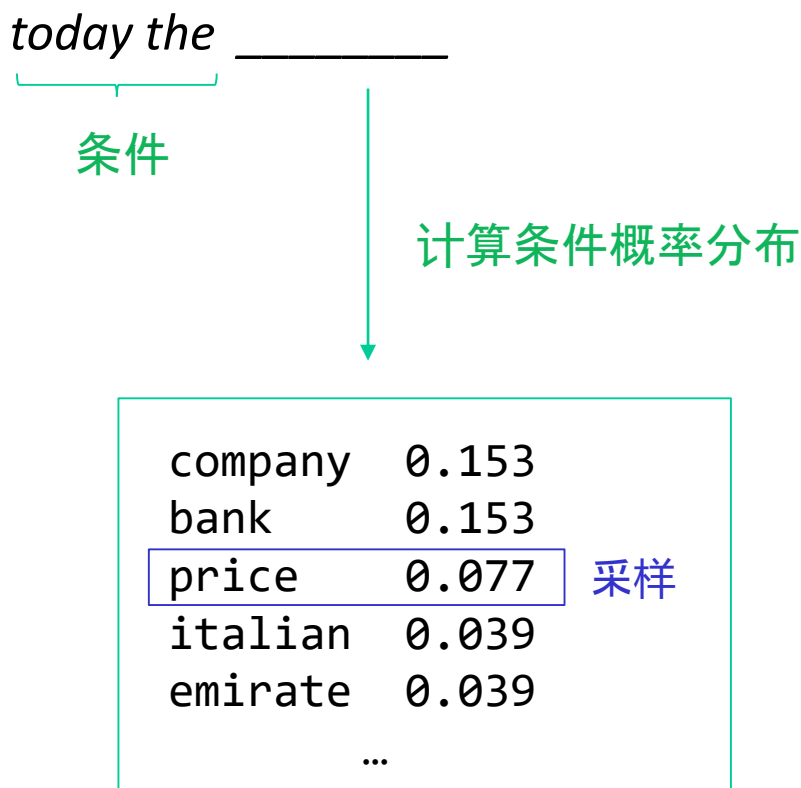
这正是语言模型在计算的概率

# 语言模型用于联想输入：

我们每天都在使用语言模型！



## ► 我们还可以使用语言模型**生成文本**



- 我们还可以使用语言模型**生成文本**

*today the price* \_\_\_\_\_

条件

计算条件概率分布

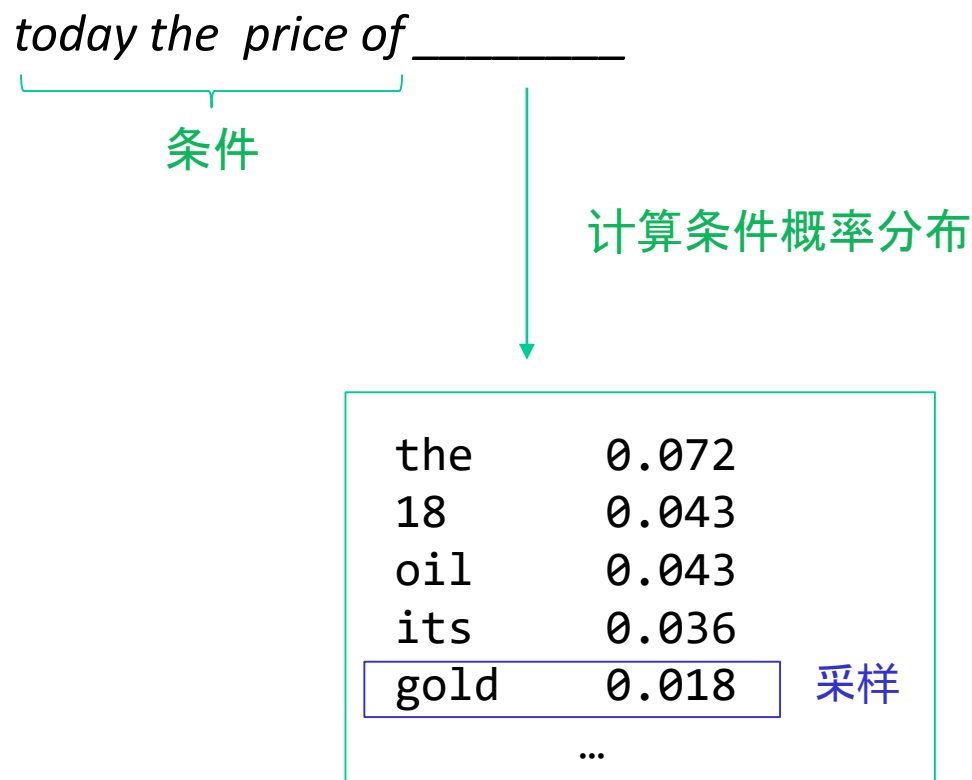
of	0.308
for	0.050
it	0.046
to	0.046
is	0.031

采样

...



- 我们还可以使用语言模型**生成文本**



## ► 我们还可以使用语言模型**生成文本**

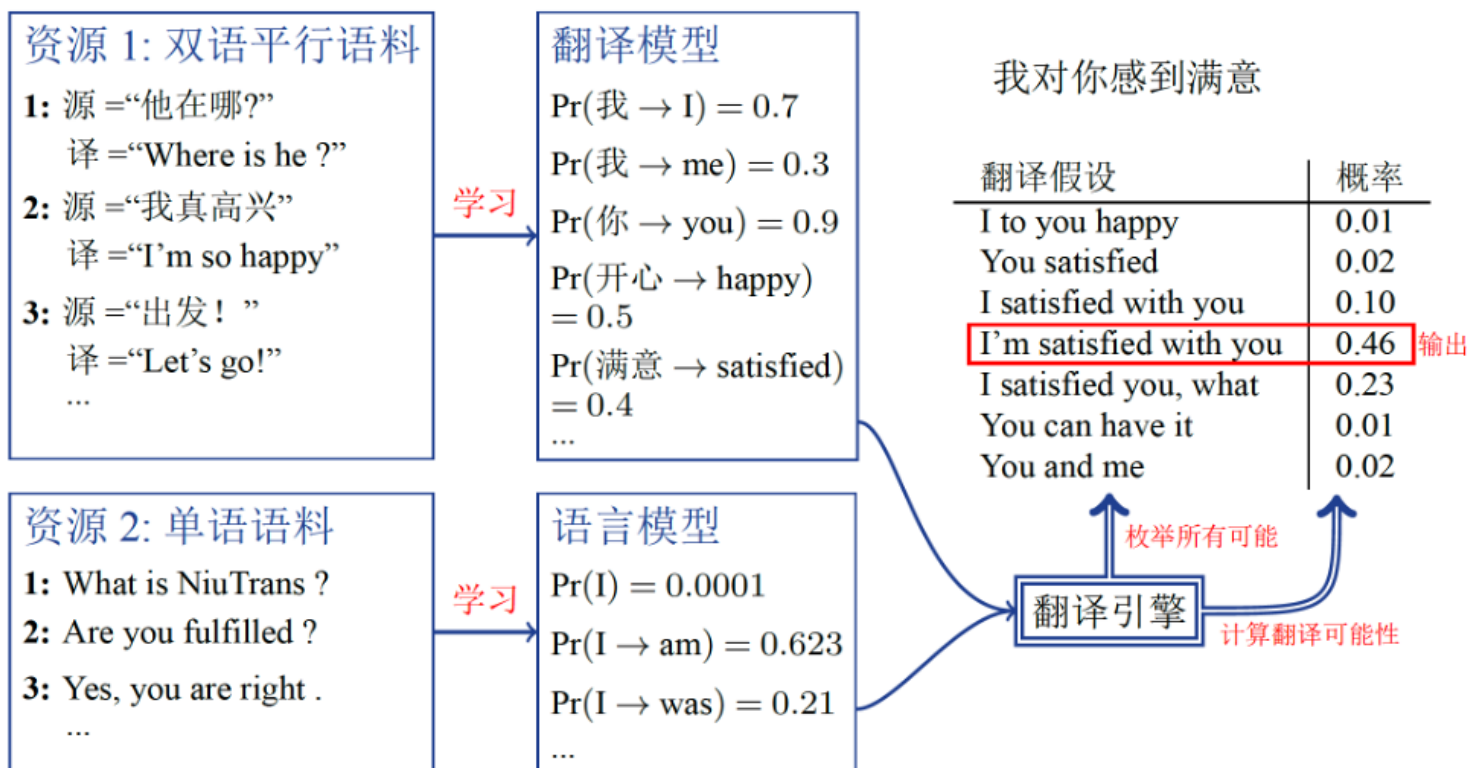
*today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .*

银行在考虑并回绝了国际货币基金组织的需求以重建耗尽的欧洲库存、结束9月30日76CTS每股之后，干预了今天的每吨黄金价格、鞋的生产持续和鞋业。

## ► 这是使用我们今天要讲的n-gram语言模型做的文本生成。

# 语言模型在机器翻译中的应用

语言模型是统计机器翻译的核心成分，与翻译模型、调序模型一起联合解码，生成最终的翻译结果。



# 语言模型的种类

学生打开了他们的\_\_\_\_\_

- ▶ 我们要如何构建一个语言模型呢？

- ▶ 使用基于统计的方法！——n-gram语言模型

- ▶ 使用神经网络！——神经语言模型

 3.2节内容

# 如何评价语言模型的性能？

- ▶ 在分类任务中，评价模型好坏的指标非常明确，就是分类精度。
- ▶ 但是对于语言模型而言，我们关注的重点在于
  - ▶ **语言模型在每个预测的位置上给出的概率值本身**（这对于估计文本概率、机器翻译中帮助挑选句子等应用场景而言比较重要），以及
  - ▶ **不同单词之间概率大小的相对顺序。**（想想前面讲的联想输入那个场景）
- ▶ 这些都超出了“下一个词是否预测正确”这样直接的评价指标所能描述的范畴。
- ▶ 所以，分类精度并不适用。

# 如何评价语言模型的性能？

- ▶ **那我们怎么办？**
- ▶ 有个很蠢但是靠谱的办法：
- ▶ 假设我们有两个不同的语言模型A和B，我们想比较他们的好坏。
  - ▶ 把它们分别放到下游任务里面去试试看：
    - ▶ 机器翻译、联想输入、拼写检查.....
  - ▶ 在每个下游任务上，通过比较A、B两个模型下游任务上能达到的性能指标，来间接评价A和B
- ▶ **但是这非常地耗费精力，并不现实。**

# 如何评价语言模型的性能？

- ▶ 更好的评价指标应该更直接地反映语言模型本身，并且要能自动计算才好。

- ▶ 既然语言模型可以计算一段文本出现的概率：

$$\begin{aligned} P(x^{(1)}, x^{(2)}, \dots, x^{(T)}) &= P(x^{(1)}) \times P(x^{(2)} | x^{(1)}) \times \dots \times P(x^{(T)} | x^{(T-1)}, \dots, x^{(1)}) \\ &= \prod_{t=1}^T P(x^{(t)} | x^{(t-1)}, \dots, x^{(1)}) \end{aligned}$$

- ▶ 如果给出一段**足够长的、真实的、模型没有见过的**语料，那么，模型对这段语料预测的概率越高，则说明这个模型越好。

# 如何评价语言模型的性能？

- ▶ 如果给出一段**足够长的、真实的、模型没有见过的**语料，那么，模型对这段语料预测的概率越高，则说明这个模型越好。
  - ▶ **足够长**——避免模型“运气好”，刚好在要预测的几个位置上表现良好。
  - ▶ **真实语料**——只有在真实语料上模型预测的概率才是越高越好。
  - ▶ **模型没有见过的**——避免某些模型“作弊”，比如在训练的时候背下来整个训练集，然后测试的时候刚好遇到了它背过的东西。

$$\begin{aligned} P(x^{(1)}, x^{(2)}, \dots, x^{(T)}) &= P(x^{(1)}) \times P(x^{(2)} | x^{(1)}) \times \dots \times P(x^{(T)} | x^{(T-1)}, \dots, x^{(1)}) \\ &= \prod_{t=1}^T P(x^{(t)} | x^{(t-1)}, \dots, x^{(1)}) \end{aligned}$$



# 如何评价语言模型的性能？

- ▶ 假设我们给模型看了满足上述条件的这样一个语料（我们叫**测试集**），这个语料是由 $T$ 个单词组成的

$$\mathcal{D}_{test} = [x^{(1)}, x^{(2)}, \dots, x^{(T)}]$$

- ▶ 模型给出该测试集的文本概率

$$P(x^{(1)}, x^{(2)}, \dots, x^{(T)}) = P(x^{(1)}) \times P(x^{(2)}|x^{(1)}) \times \dots \times P(x^{(T)}|x^{(T-1)}, \dots, x^{(1)})$$

$$= \prod_{t=1}^T P(x^{(t)}|x^{(t-1)}, \dots, x^{(1)})$$

- ▶ 接着我们对测试集长度归一化，把这概率平均分摊到每个词上（注意，因为是求相乘的平均，这里我们用几何平均）：

$$\sqrt[T]{P(x^{(1)}, x^{(2)}, \dots, x^{(T)})} = \sqrt[T]{\prod_{t=1}^T P(x^{(t)}|x^{(t-1)}, \dots, x^{(1)})}$$

# 如何评价语言模型的性能？

- ▶ 接着我们对测试集长度归一化，把这概率平均分摊到每个词上（注意，因为是相乘，这里我们用几何平均）：

$$\sqrt[T]{P(x^{(1)}, x^{(2)}, \dots, x^{(T)})} = \sqrt[T]{\prod_{t=1}^T P(x^{(t)} | x^{(t-1)}, \dots, x^{(1)})}$$

- ▶ 这个时候，上式的物理意义是  
“在测试集中，平均每个模型做出预测的位置上，那个实际出现的真实词被模型赋予的概率。”
- ▶ 事实上这时候我们已经得到了我们想要的评价指标了。但是我们还要加上最后这一步：我们对上式取倒数：

$$\text{PPW} = \frac{1}{\sqrt[T]{\prod_{t=1}^T P(x^{(t)} | x^{(t-1)}, \dots, x^{(1)})}}$$

# 混淆度 (perplexity)

$$\text{PPW} = \frac{1}{\sqrt[T]{\prod_{t=1}^T P(x^{(t)} | x^{(t-1)}, \dots, x^{(1)})}}$$

- ▶ 我们得到的上式就是语言模型的评价指标。它的物理意义是

“在测试集中，平均每个模型做出预测的位置上，那个实际出现的真实词被模型赋予的概率的倒数。”

- ▶ 这个量有个名字，它叫做**混淆度 (perplexity)**，因为我们对每个位置做了平均，每个位置实际上对应一个单词。所以也叫做**perplexity per word**，简写成**PPW**。

# 混淆度 (perplexity)

$$\text{PPW} = \frac{1}{\sqrt[T]{\prod_{t=1}^T P(x^{(t)} | x^{(t-1)}, \dots, x^{(1)})}}$$

- ▶ 为啥非得取个倒数呢？为啥这个量叫做混淆度呢？
- ▶ 我们考虑两个极端情况下的例子来体会一下：
  - ▶ 1. 假设有一个完全没受训练的“白板”级语言模型，他在每个位置上，对每个可能的单词都给出一样的概率。即全都是 $\frac{1}{|V|}$ 。
  - ▶ 2. 假设有一个“终极大师”级的完美语言模型，每次都能准确预测出单词。即在每个位置上，那个正确的单词都能被模型预测到1的概率。

# 混淆度 (perplexity)

$$\text{PPW} = \frac{1}{\sqrt[T]{\prod_{t=1}^T P(x^{(t)} | x^{(t-1)}, \dots, x^{(1)})}}$$

- ▶ 为啥非得取个倒数呢？为啥这个量叫做混淆度呢？
- ▶ 我们考虑两个极端情况下的例子来体会一下：
  - ▶ 1. 假设有一个完全没受训练的“白板”级语言模型，他在每个位置上，对每个可能的单词都给出一样的概率。即全都是 $\frac{1}{|V|}$ 。

$$\text{PPW}_{\text{白板}} = |V|$$

- ▶ 2. 假设有一个“终极大师”级的完美语言模型，每次都能准确预测出单词。即在每个位置上，那个正确的单词都能被模型预测到1的概率。

$$\text{PPW}_{\text{终极大师}} = 1$$

# 混淆度 (perplexity)

$$\text{PPW} = \frac{1}{\sqrt[T]{\prod_{t=1}^T P(x^{(t)} | x^{(t-1)}, \dots, x^{(1)})}}$$

- ▶ 为啥非得取个倒数呢？为啥这个量叫做混淆度呢？
- ▶ 所以PPW的取值范围一般在  $1 \sim |V|$  之间，并且是值越低越好。
- ▶ 事实上，我们还能这么理解混淆度：比如，某种语言的词汇量一共是30000个，但是我们的语言模型混淆度达到了30。
  - ▶ 这意味着模型在每次预测的时候，相当于是在包含正确单词的30个词当中随机选择了一个词。也就是说，它相当于一个“白板”语言模型，但是在每次预测的时候，有人人工帮它排除了一堆错误答案，只留下30个供它随机选择。
  - ▶ 在预测下一个词的时候，它相当于只随机混淆了30个词。
  - ▶ 这也是为什么这个量叫做混淆度。

- ▶ 什么是语言模型
  - ▶ 语言模型的应用场景
  - ▶ 语言模型的种类
  - ▶ 如何评价语言模型的性能：混淆度 (perplexity)
- ▶ **N-gram语言模型**
  - ▶ N-gram的定义
  - ▶ 如何确定N-gram语言模型中的参数
  - ▶ 最大似然估计
  - ▶ Unigram、bigram示例
- ▶ N-gram语言模型中的数据稀疏问题及解决办法
  - ▶ 折扣法 (Discounting)
  - ▶ 回退 (Backoff) 及插值 (interpolation)

学生/打开/了/他们/的/\_\_\_\_\_

- ▶ 定义：n-gram是由n个连续单词组成的块
  - ▶ unigrams: “学生”，“打开”，“了”，“他们”，“的”
  - ▶ bigrams: “学生打开”，“打开了”，“了他们”，“他们的”
  - ▶ trigrams: “学生打开了”，“打开了他们”，“了他们的”
  - ▶ 4-grams: “学生打开了他们”，“打开了他们的”
- ▶ 思想：收集不同n-gram出现频率的统计数据，并用这些数据预测下一个单词



# n-gram语言模型

- ▶ 马尔科夫假设:  $x^{(t+1)}$  只依赖于前面  $n - 1$  个词

$$P(x^{(t+1)} | x^{(t)}, \dots, x^{(1)}) = P(x^{(t+1)} | \overbrace{x^{(t)}, \dots, x^{(t-n+2)}}^{n-1 \text{ 个词}}) \quad (\text{假设})$$

某个n-gram的概率



$$= \frac{P(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})}{P(x^{(t)}, \dots, x^{(t-n+2)})}$$

(条件概率的定义)

某个(n-1)-gram的概率



- ▶ 我们又要如何获得这些n-gram和(n-1)-gram的概率呢?
  - ▶ 在海量的文本中统计它们出现的次数

$$\approx \frac{\text{count}(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})}{\text{count}(x^{(t)}, \dots, x^{(t-n+2)})} \quad (\text{统计近似})$$

# n-gram语言模型：例子

- ▶ 例如，我们在学一个6-gram语言模型

学生/打开/了/他们/的\_\_\_\_\_

条件

$$P(w|\text{学生, 打开, 了, 他们, 的}) = \frac{\text{count}(\text{学生, 打开, 了, 他们, 的}, w)}{\text{count}(\text{学生, 打开, 了, 他们, 的})}$$

- ▶ 假设在语料库中，统计发现

- ▶ “学生打开了他们的” 出现了1000次
- ▶ “学生打开了他们的课本” 出现了400次
  - ▶  $P(\text{课本}|\text{学生, 打开, 了, 他们, 的}) = 0.4$
- ▶ “学生打开了他们的试题” 出现了100次
  - ▶  $P(\text{试题}|\text{学生, 打开, 了, 他们, 的}) = 0.1$

所以模型预测  
填入的词最有  
可能是：  
课本

- ▶ 在前面的推导中，我们很自然地将概率预测用“数数”代替了：

$$\begin{aligned} P(x^{(t+1)} | x^{(t)}, \dots, x^{(t-n+2)}) &= \frac{P(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})}{P(x^{(t)}, \dots, x^{(t-n+2)})} \\ &\approx \frac{\text{count}(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})}{\text{count}(x^{(t)}, \dots, x^{(t-n+2)})} \end{aligned}$$

- ▶ 虽然这很直观也很自然，但是问题来了.....

- ▶ 在前面的推导中，我们很自然地将概率预测用“数数”代替了：

$$\begin{aligned} P(x^{(t+1)} | x^{(t)}, \dots, x^{(t-n+2)}) &= \frac{P(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})}{P(x^{(t)}, \dots, x^{(t-n+2)})} \\ &\approx \frac{\text{count}(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})}{\text{count}(x^{(t)}, \dots, x^{(t-n+2)})} \end{aligned}$$

- ▶ 虽然这很直观也很自然，但是问题来了.....
- ▶ 我们估计模型参数，或者说训练模型，的初衷，本是“最大化模型在训练数据上的预测概率”。
- ▶ 但是这么数数似乎和上面这个初衷并没啥联系。这么数出来的概率它对吗？

# 最大似然估计

在统计学中，最大似然估计（英语：Maximum Likelihood Estimation，简作 MLE），也称极大似然估计，是用来估计一个概率模型的参数的一种方法。

给定一个概率分布，已知其概率密度函数（连续分布）或概率质量函数（离散分布）为  $P_\theta$ ，分布参数为  $\theta$ ，我们可以从这个分布中抽出一个具有  $n$  个值的采样集合  $x_1, \dots, x_n$ ，利用  $P_\theta$  计算出其似然函数：

$$\mathcal{L}(\theta \mid x_1, \dots, x_n) = P_\theta(x_1, \dots, x_n).$$

注意：似然函数是变量  $\theta$  和观测数据  $x_1, \dots, x_n$  的函数

- 我们的训练任务，就是调整  $\theta$ ，使得上面这个似然函数最大化。这样估计出来的  $\theta$ ，叫做  $\theta$  的**最大似然估计**。
- 为了把乘积操作转化为求和操作，以及防止似然函数过小，通常我们会计算上面这个概率的对数值。

# 最大似然估计

**数据:** 词序列  $W_1^N = [w_1, \dots, w_N]$

**模型:** 离散概率

$$P(w_k | W_{k-n+1}^{k-1}), \quad w \in \mathcal{V}$$

其中  $\mathcal{V} = \{v_1, \dots, v_M\}$  是词表

**Criterion:**

$$\begin{aligned} \mathcal{L}(\theta) &= \frac{1}{K} \sum_{k=1}^K \log P(w_k | W_{k-n+1}^{k-1}) \\ &= \frac{1}{K} \sum_{v \in \mathcal{V}} \sum_{y \in \mathcal{Y}} \sum_{k: w_k=v, y=W_{k-n+1}^{k-1}} \log P(v|y) \\ &= \frac{1}{K} \sum_{v \in \mathcal{V}} \sum_{y \in \mathcal{Y}} C(y, v) \log P(v|y) \end{aligned}$$

其中  $\mathcal{Y}$  是所有的  $n$ -gram 历史的集合,  $C(y, v)$  是训练集中完整  $n$ -gram  $(y, v)$  的数目。

$$\hat{P}(v|y) = \arg \max_{P(v|y)} \sum_{v \in \mathcal{V}} \sum_{y \in \mathcal{Y}} C(y, v) \log P(v|y), \quad s.t. \sum_{v \in \mathcal{V}} P(v|y) = 1$$

使用拉格朗日乘子，针对参数  $P(v|y)$  的带约束的优化可以表示为

$$Q(P(v|y)) = C(y, v) \log P(v|y) + \lambda \left( \sum_{v \in \mathcal{V}} P(v|y) - 1 \right), v \in \mathcal{V}$$

分别关于参数  $P(v|y)$  与  $\lambda$  对以上式子求偏导数，并令其等于零：

$$\frac{\partial Q}{\partial P(v|y)} = 0, v \in \mathcal{V} \qquad \frac{\partial Q}{\partial \lambda} = 0$$

$$\begin{aligned} \frac{C(y, v)}{\hat{P}(v|y)} + \lambda &= 0, v \in \mathcal{V} \\ -\frac{C(y, v)}{\lambda} &= \hat{P}(v|y), v \in \mathcal{V} \end{aligned} \quad \Rightarrow \quad \begin{aligned} \sum_{v \in \mathcal{V}} \hat{P}(v|y) - 1 &= 0 \\ -\frac{\sum_{v \in \mathcal{V}} C(y, v)}{\lambda} &= 1 \\ -\sum_{v \in \mathcal{V}} C(y, v) &= \lambda \end{aligned}$$

因此, 对  $\forall v \in \mathcal{V}$ , 有

$$\hat{P}(v|y) = \frac{C(y, v)}{C(y)}$$

其中

$$C(y) = \sum_{v \in \mathcal{V}} C(y, v)$$



# N-gram语言模型的参数估计： unigram示例

**Training sentences:**

<S> I HAVE A RED CAR <S>
<S> I BUY A NEW CAR <S>
<S> THEY HAVE A NEW BOOK <S>

**Vocabulary:**

A	BOOK	BUY	CAR	HAVE	I	NEW	RED	THEY
---	------	-----	-----	------	---	-----	-----	------

**Unigram 语言模型:**

$P(A)$	3/15	$P(BOOK)$	1/15	$P(BUY)$	1/15
$P(CAR)$	2/15	$P(HAVE)$	2/15	$P(THEY)$	2/15
$P(BOOK NEW)$	2/15	$P(CAR RED)$	1/15	$P(HAVE THEY)$	1/15

# N-gram语言模型的参数估计: bigram示例

Training sentences:

<S> I HAVE A RED CAR <S>  
<S> I BUY A NEW CAR <S>  
<S> THEY HAVE A NEW BOOK <S>

Bigram 语言模型:

$P(I   <S>)$	2/3	$P(THHEY   <S>)$	1/3	$P(NEW A)$	2/3
$P(RED A)$	1/3	$P(A BUY)$	1/1	$P(A HAVE)$	2/2
$P(HAVE I)$	1/2	$P(BUY I)$	1/2	$P(CAR NEW)$	1/2
$P(BOOK NEW)$	1/2	$P(CAR RED)$	1/1	$P(HAVE THEY)$	1/1

# n-gram语言模型：例子

- ▶ 例如，我们在学一个6-gram语言模型

当/监考员/发出/指令/后/, /学生/打开/了/他们/的/\_\_\_\_

舍弃

条件

$$P(w|\text{学生, 打开, 了, 他们, 的}) = \frac{\text{count}(\text{学生, 打开, 了, 他们, 的}, w)}{\text{count}(\text{学生, 打开, 了, 他们, 的})}$$

- ▶ 假设在语料库中，统计发现
  - ▶ “学生打开了他们的” 出现了1000次
  - ▶ “学生打开了他们的课本” 出现了400次
    - ▶  $P(\text{课本}|\text{学生, 打开, 了, 他们, 的}) = 0.4$
  - ▶ “学生打开了他们的试题” 出现了100次
    - ▶  $P(\text{试题}|\text{学生, 打开, 了, 他们, 的}) = 0.1$

想一想，我们是否应该舍弃有关“监考员”的上下文呢？

# n-gram语言模型：例子

## ► 我们还可以使用语言模型**生成文本**

*today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .*

银行在考虑并回绝了国际货币基金组织的需求以重建耗尽的欧洲库存、结束9月30日76CTS每股之后，干预了今天的每吨黄金价格、鞋的生产持续和鞋业。

符合语法，但是语义不连贯

- 如果我们想要更好地对语言建模，我们需要一次考虑三个以上的单词，但n的增大会加剧稀疏问题，同时会增加模型的大小

- ▶ **什么是语言模型**
  - ▶ 语言模型的应用场景
  - ▶ 语言模型的种类
  - ▶ 如何评价语言模型的性能：混淆度 (perplexity)
- ▶ **N-gram语言模型**
  - ▶ N-gram的定义
  - ▶ 如何确定N-gram语言模型中的参数
  - ▶ 最大似然估计
  - ▶ Unigram、bigram示例
- ▶ **N-gram语言模型中的数据稀疏问题及解决办法**
  - ▶ 折扣法 (Discounting)
  - ▶ 回退 (Backoff) 及插值 (interpolation)

# n-gram语言模型中的数据稀疏问题

数据稀疏问题一直存在：大词表情况下的高阶  $n$ -gram。

数据稀疏的相关问题主要有如下：

- ▶ **零概率 (Unseen)** 的  $n$ -grams 将会导致无穷大的分支度 PPL
- ▶ **低频次 (Very low)** 的  $n$ -grams 将会导致不可靠的估计

两种解决方案：

- ▶ **折扣法 (Discounting)**  
修正原始  $n$ -gram 的频次，使得概率量重新分布，从通常容易观测到的  $n$ -gram 分布  $\rightarrow$  频次较低或者根本没有的  $n$ -gram 上
- ▶ **回退及插值 (Backing-off and interpolation)**  
回退策略递归地回退到较低阶的  $n$ -gram 上，直到得到一个鲁棒的概率估计

# n-gram语言模型中的数据稀疏问题

- 事实上，数据稀疏问题远比我们以为的严重。以下是在Berkeley Restaurant Corpus上统计得到的一部分bigram的次数信息：

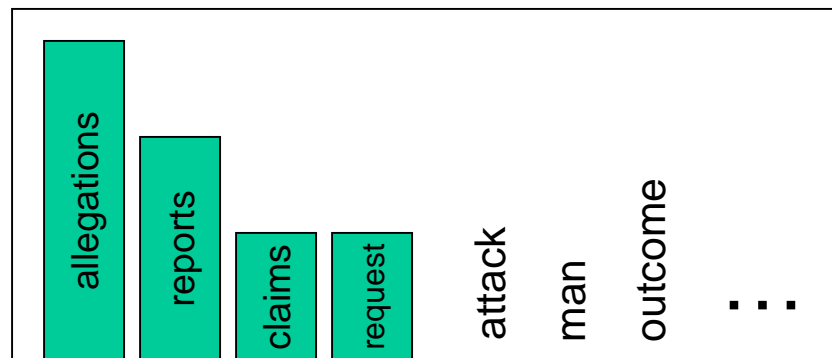
	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

- 即便在最简单的bigram 语言模型中，**90%以上的bigram都是空白的！**

# 数据稀疏问题的解决思路

## ▶ 当我们遇到训练集中零样本的n-gram时:

$P(w \mid \text{denied the})$   
3 allegations  
2 reports  
1 claims  
1 request  
7 total



## ▶ 劫富济贫，将一部分高频n-gram的概率值分配到那些零样本n-gram上

$P(w \mid \text{denied the})$   
2.5 allegations  
1.5 reports  
0.5 claims  
0.5 request  
**2 other**  
7 total





## 具体方法1: Laplace平滑

- ▶ Laplace平滑又被称为add-one平滑。它的做法非常简单。就简单粗暴地将n-gram表中的所有n-gram出现频次全都+1。
- ▶ 这相当于我们给训练集补了一堆语料，在这堆语料里面，每个n-gram都刚好只出现了一次。
- ▶ 比如对于bigram而言，原来的数数方式只要稍做改变，就可以避免数据稀疏问题了：

▶ **MLE estimate:** 
$$P_{MLE}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

▶ **Add-1 estimate:** 
$$P_{Add-1}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

# 具体方法1: Laplace平滑

	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

	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

## 具体方法1: Laplace平滑

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

	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

## 具体方法1: Laplace平滑

- 我们还可以将上面表中的数字乘以 $C(w_{n-1})$ 来计算等效的词频计数:

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V}$$

	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

## 具体方法1: Laplace平滑

	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

	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

- 对数据的影响非常大，尤其是在出现词频高的常见n-gram上。所以这方法虽然简单，但是效果不好。在语言模型中并没有实际应用。

# 回退 (backoff) 及插值 (interpolation)

- ▶ **回退 (backoff) :**

- ▶ 当更高阶的n-gram不存在时，回退到使用低一阶的n-gram算出来的条件概率来替代。
- ▶ 比如trigram 没有那就找bigram, bigram都没有那就找unigram。

- ▶ **插值 (interpolation) :**

- ▶ 将不同阶的n-gram所给出的条件概率混合到一起。

# 回退法举例: “stupid backoff” (Brants *et al.* 2007)

- ▶ 在神经语言模型崛起之前，这是Google在海量数据上算得的巨大规模的n-gram模型上使用的方法。该方法不使用discounting，直接使用相对频率来估计条件概率。
- ▶ 得益于海量的数据，该方法实际效果很不错。

$$S(w_i | w_{i-k+1}^{i-1}) = \begin{cases} \frac{\text{count}(w_{i-k+1}^i)}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^i) > 0 \\ 0.4S(w_i | w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

## 插值方法举例：线性插值

- ▶ 线性插值使用unigram、bigram、trigram等各阶n-gram所给出的条件概率的线性组合来估计最终的条件概率。即：

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) = & \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ & + \lambda_2 P(w_n|w_{n-1}) \\ & + \lambda_3 P(w_n)\end{aligned} \quad \sum_i \lambda_i = 1$$

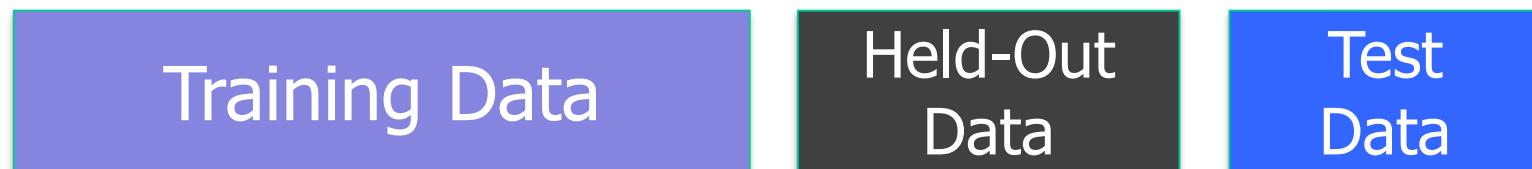
- ▶ 注意，为了得到正确归一化的概率，这里 $\sum_i \lambda_i = 1$ 的约束条件必须满足。同时，对于不同上下文， $\lambda_i$ 也不是固定的，而是当前上下文的函数：

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) = & \lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) \\ & + \lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-1}) \\ & + \lambda_3(w_{n-2}^{n-1})P(w_n)\end{aligned}$$



# 插值方法举例：线性插值

- ▶  $\lambda_i$  参数值的确定：



- ▶  $\lambda_i$  参数值通过引入一个验证集 (held-out data, or validation set, or development set) 来确定。具体步骤如下：
  - ▶ 在训练集上计算各个n-gram，并固定他们的数值。
  - ▶ 在验证集上调整 $\lambda_i$ ，使得模型对验证集的预测概率最大化。

$$\log P(w_1 \dots w_n \mid M(I_1 \dots I_k)) = \sum_i \log P_{M(I_1 \dots I_k)}(w_i \mid w_{i-1})$$

# 插值法举例：Absolute discounting

- ▶ 假设我们要从高频n-gram中抽出一些频度来分给零样本的n-gram们。那么分多少比较好呢？

- ▶ Church and Gale (1991)想了一个很接地气的方法，就是去实际数据集里统计了看看。他们在一个2200万token的数据集里统计了一下。
- ▶ 然后他们惊奇地发现，对于大部分情况而言，平均只要扣掉0.75就可以了！

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

## 插值法举例：Absolute discounting

- ▶ 所以不如简单点，直接扣掉0.75就好了！(d=0.75)!

$$P_{\text{AbsoluteDiscounting}}(w_i | w_{i-1}) = \frac{\overset{\text{discounted bigram}}{c(w_{i-1}, w_i) - d}}{c(w_{i-1})} + \overset{\text{Interpolation weight}}{1 / (\overbrace{w_{i-1}}^{\swarrow})} \overset{\text{unigram}}{P(w)}$$

- ▶ 如果再精调下，对于只出现了一次或者两次的少样本 n-gram，可以另外设个d值，就更好了。
- ▶  $P(w)$ 这儿还能再改下.....

## 更复杂但实用的平滑方法：Kneser-Ney 平滑

- ▶ 考虑一个例子：  
*I can't see without my reading* <sup>glasses</sup> Kong ?
  - ▶ “Kong” 出现的频率比 “glasses” 更大
  - ▶ 但是 “Kong” 大部分时候跟在 “Hong” 之后出现。
- ▶ 如果没有上述的第二条，其实这里填 *Kong* 是合适的。但是正因为有这第二条所述的情况存在，使用  $P(w)$  就显得不是很合适。我们把它替换成  $P_{\text{continuation}}(w)$ 。
- ▶  $P_{\text{continuation}}(w)$  的直观意义就是，“w 作为一个新的 n-gram 出现的概率有多大？”

# 更复杂但实用的平滑方法：Kneser-Ney 平滑

- ▶ 我们这么来计算  $P_{\text{continuation}}(w)$  :
  - ▶ 对于每个词，计算他的bigram的**种类**。  
 $P_{\text{continuation}}(w)$ 应该正比于这个值，即：

$$P_{\text{CONTINUATION}}(w) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

- ▶ 因为是概率，我们还要做归一化。对上面的值除以bigram的**种类总数**： $|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|$
- ▶ 我们就得到了：

$$P_{\text{CONTINUATION}}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|}$$

- ▶ 如此一来，对于像Kong这样基本只出现在Hong之后的单词，它的  $P_{\text{continuation}}(\text{"Kong"})$  就会很小。

# 更复杂但实用的平滑方法：Kneser-Ney 平滑

- ▶ Kneser-Ney平滑可以表示为:

$$P_{KN}(w_i | w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1})P_{CONTINUATION}(w_i)$$

其中 $\lambda$ 是一个插值系数，通过验证集来确定：

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} |\{w : c(w_{i-1}, w) > 0\}|$$

the normalized discount

The number of word types that can follow  $w_{i-1}$   
= # of word types we discounted  
= # of times we applied normalized discount