

# 现代信息检索

# Modern Information Retrieval

## 第4讲 通配查询与拼写矫正

## Wildcard queries and Spelling Correction

# 本讲内容

- 词典的数据结构：访问效率和支持查找的方式
- 通配查询：包含通配符\*的查询
- 拼写校正：查询中存在错误时的处理

# 提纲

## 词典

通配查询

编辑距离

拼写校正

Soundex

# 倒排索引

对每个词项 $t$ , 保存所有包含 $t$ 的 文档列表

BRUTUS	→	1	2	4	11	31	45	173	174	
CAESAR	→	1	2	4	5	6	16	57	132	...
CALPURNIA	→	2	31	54	101					

⋮

词典(dictionary)  
词典(dictionary)

倒排记录表(posting list)  
倒排记录表(postings)

# 词典

- 词典是指存储词项词汇表的数据结构
- 词项词汇表(Term vocabulary): 指的是具体数据
- 词典(Dictionary): 指的是数据结构

# 采用定长数组的词典结构

- 对每个词项，需要存储：
  - 文档频率 (Document Frequency, DF)
  - 指向倒排记录表的指针
  - ...
- 暂定每条词项的上述信息均采用定长的方式存储
- 假定所有词项的信息采用数组存储

# 采用定长数组的词典结构

词项	文档频率	指向倒排记录表的指针
a	656 265	→
aachen	65	→
...	...	...
zulu	221	→

空间消耗： 20字节    4字节            4字节

# 词项定位(即查词典)

- 在词典中查找给定关键字
- 输入“信息”，如何在词典中快速找到这个词？
- 很多词典应用中的基本问题。
- 以下介绍支持快速查找的词典数据结构。

18

信息

19

数据

20

21

22

挖掘



# 用于词项定位的数据结构

- 主要有两种数据结构：哈希表和树
- 有些IR系统用哈希表，有些系统用树结构
- 采用哈希表或树的准则：
  - 词项数目是否固定或者说词项数目是否持续增长？
  - 词项的相对访问频率如何？固定则用哈希表
  - 词项的数目有多少？

# 哈希表

哈希函数，输入词项，输出正整数(通常是地址)

$f(\text{信息})=18$ ,  $f(\text{数据})=19$ ,  
 $f(\text{挖掘})=19$



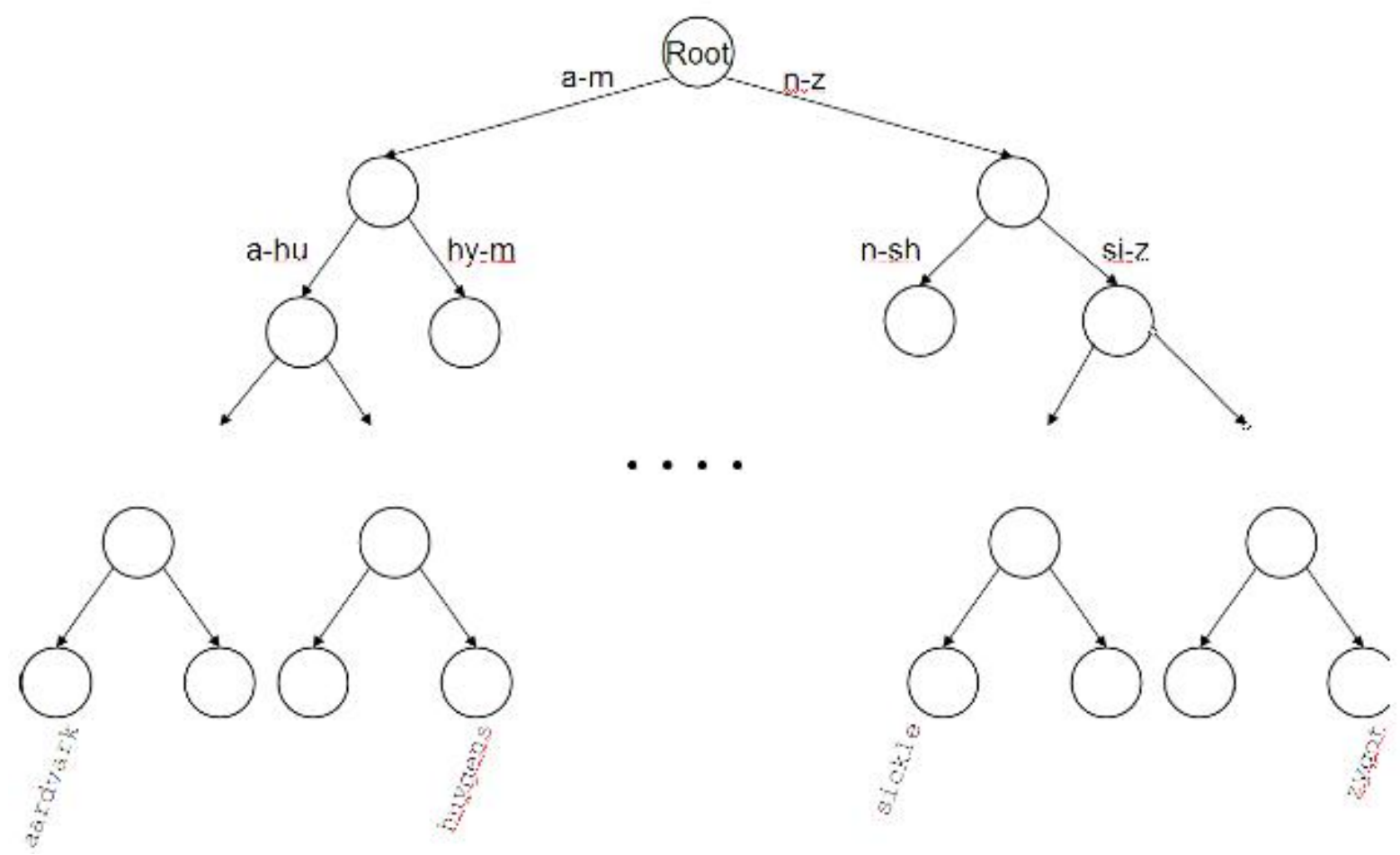
# 哈希表(Hash Table)

- 每个词项通过哈希函数映射成一个整数
- 尽可能避免冲突
- 查询处理时：对查询词项进行哈希，如果有冲突，则解决冲突，最后在定长数组中定位
- 优点：在哈希表中的定位速度快于树中的定位速度
- 查询时间是常数
- 缺点：
  - 无法处理词项的微小变形 (*resume* vs. *résumé*)
  - 不支持前缀搜索 (比如所有以*automat*开头的词项)
  - 如果词汇表不断增大，需要定期对所有词项重新哈希

# 树(Tree)

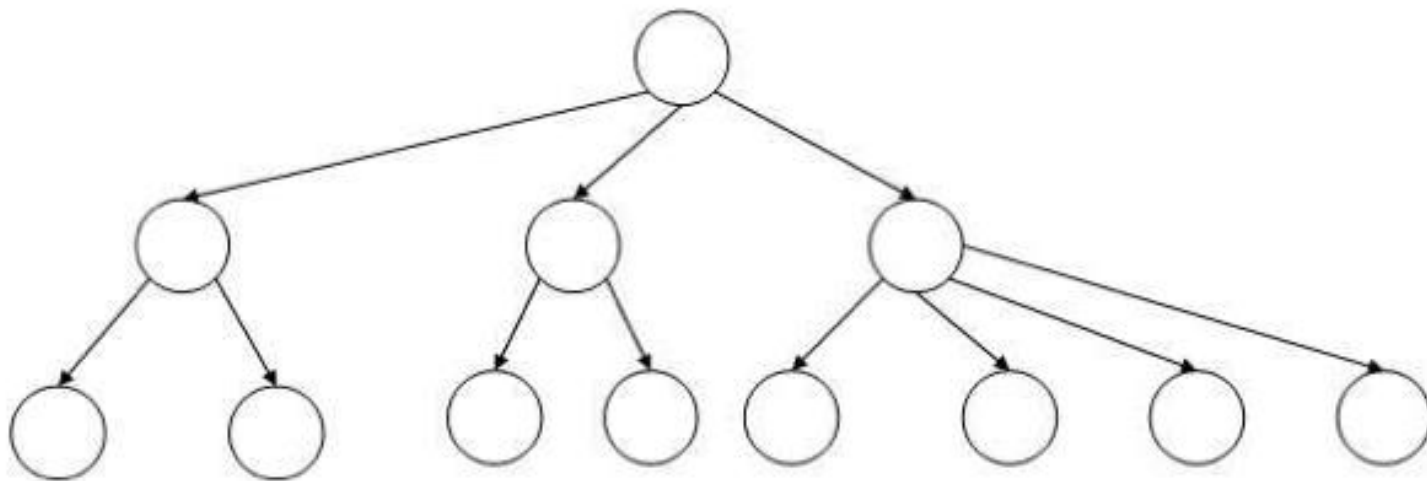
- 树可以支持前缀查找(相当于对词典再建一层索引)
- 最简单的树结构：二叉树
- 搜索速度略低于哈希表方式：  $O(\log M)$ , 其中  $M$  是词汇表大小，即所有词项的数目
- $O(\log M)$  仅仅对平衡树成立
- 使二叉树重新保持平衡开销很大
- B-树 能够减轻上述问题
- B-树定义：每个内部节点的子节点数目在  $[a, b]$  之间，其中  $a, b$  为合适的正整数, e.g.,  $[2, 4]$ .

# 二叉树



# B-树

使用B树，更新的消耗更小



# 提纲

词典

**通配查询**

编辑距离

拼写校正

Soundex

# 通配查询

- 通配查询：包含通配符的查询，例如：

mon\*: 找出所有包含以 *mon*开头的词项的文档  
符合条件的词项有：Monday, monster, monitor等

\*mon: 找出所有包含以*mon*结尾的词项的文档  
符合条件的词项有：summon, common, cinnamon等



# 通配查询的处理

$mon^*$ : 找出所有包含以 *mon* 开头的词项的文档

- 如果采用B-树词典结构，那么实现起来非常容易，只需要返回区间  $mon \leq t < moo$  上的词项  $t$

# 通配查询的处理

\*mon: 找出所有包含以*mon*结尾的词项的文档

- 将所有的词项倒转过来，然后基于它们建一棵附加的树
- 返回区间  $nom \leq t < non$  上的词项  $t$

■ 比如有以下四个词：absurd, Monday, monster, zoo

■ 全部倒转过来：drusba, yadnom, retsnom, ooz

■ 然后基于倒转过来的词项再建一颗B 树

■ 也就是说，通过上述数据结构，可能得到满足通配查询的一系列词项，然后返回任一词项的文档

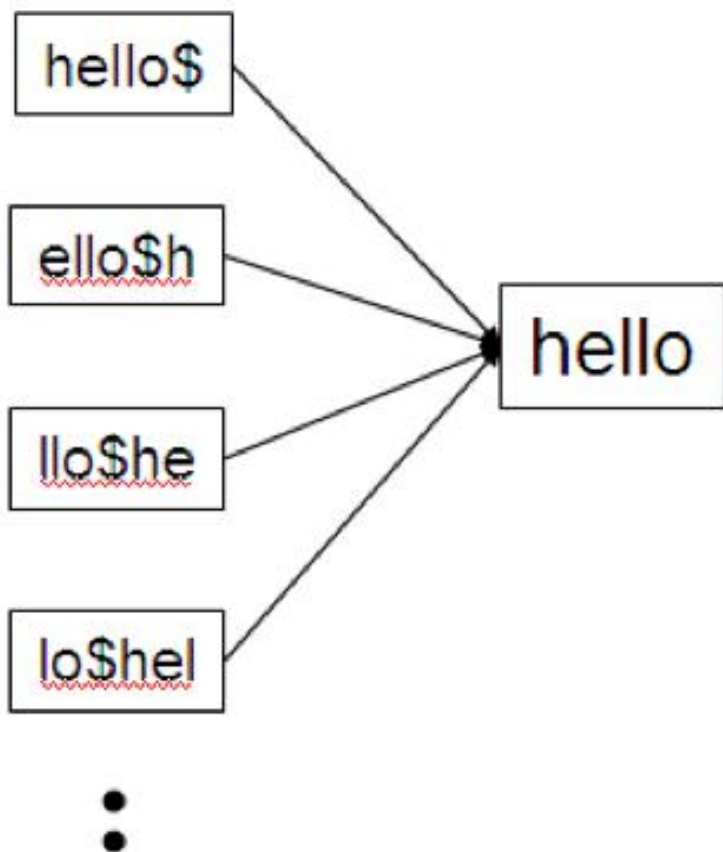
# 词项中间的 \*号处理

- 例子: m\*nchen
- 在B-树中分别查找满足m\*和 \*nchen的词项集合，然后求交集
- 这种做法开销很大
- 另外一种方法: 轮排([permuterm](#)) 索引
- 基本思想:
  - 将每个通配查询旋转，使\*出现在末尾
  - 将每个旋转后的结果存放在词典中，即B-树中

# 轮排索引

- 对于词项hello: 将 *hello\$, ello\$h, llo\$he, lo\$hel, o\$hell* 和 *\$hello* 加入到 B-树中, 其中 \$ 是一个特殊符号(表示结尾)
- 即在词项前面再加一层索引
  - 该索引采用B-树来组织
- 该索引叶节点是词项的各种变形

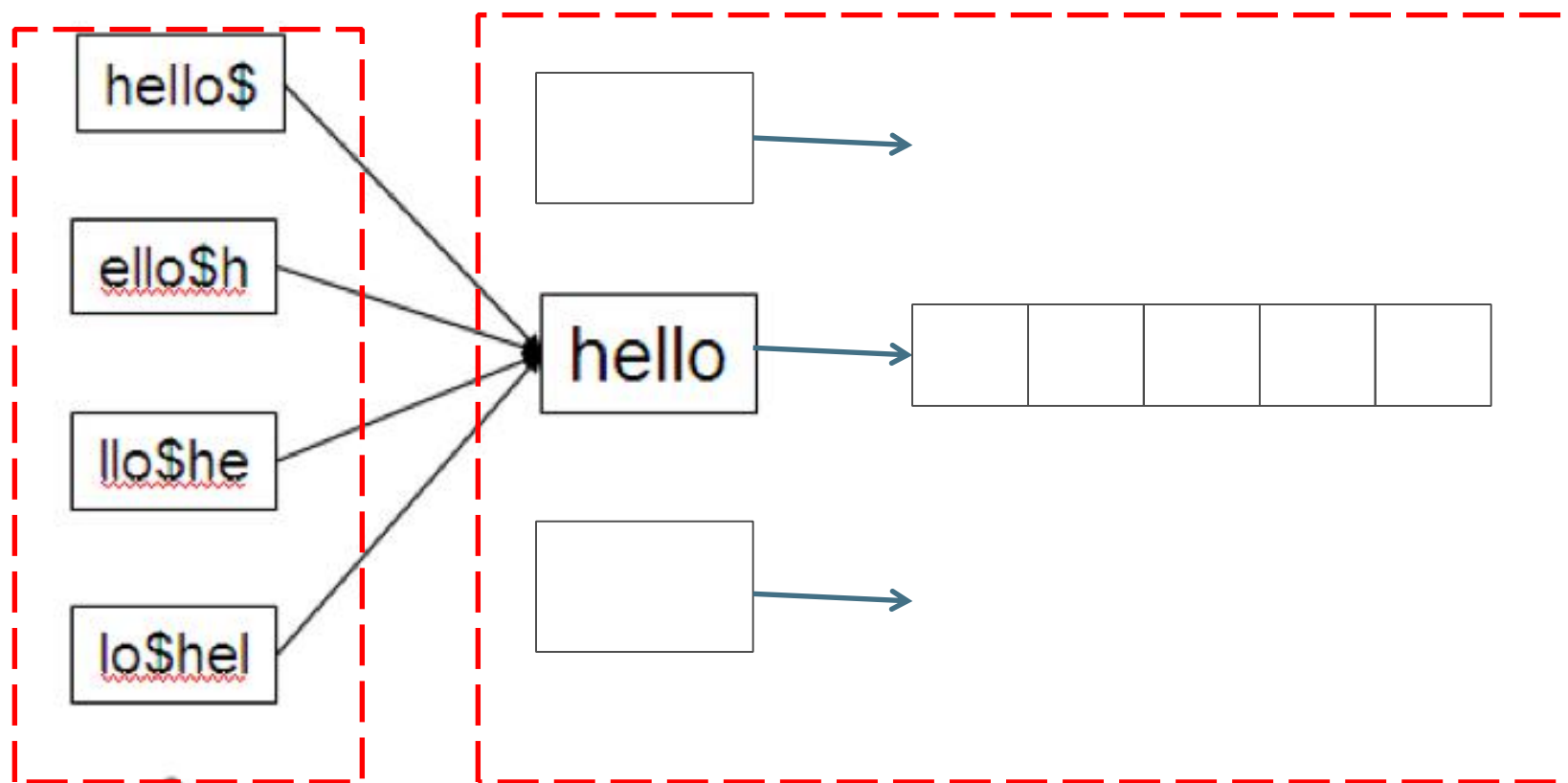
# 轮排结果 → 词项的映射示意图



# 轮排索引 ?

- 对于hello, 已经存储了 *hello\$, ello\$h, llo\$he, lo\$hel, o\$hell*和 *\$hello*
- 查询
  - 对于  $x$ , 查询  $x\$$
  - 对于  $x^*$ , 查询  $\$x^*$
  - 对于  $*x$ , 查询  $x\$^*$
  - 对于  $*x^*$ , 查询  $x^*$ , 比如 $*ello^*$ , 则只需要查到ello开头的串即可(上面是ello\$h), 因为此时ello右部一定包含一个\$, \$结不结尾的串均能满足查询 $*x^*$ ,
  - 对于  $x*y$ , 查询  $y\$x^*$
  - 例子: 假定通配查询为  $hel^*o$ , 那么相当于要查询 $o\$hel^*$
- 轮排索引称为轮排树更恰当
- 但是轮排索引的称呼已经使用非常普遍

# 轮排索引小结



轮排索引（通配查询 词项，采用B树来组织）

传统倒排索引（词项 文档）

# 使用轮排索引的查找过程

- 将查询进行旋转，将通配符旋转到右部
- 同以往一样查找B-树，得到匹配的所有词项，将这些词项对应的倒排记录表取出
- 问题：相对于通常的B-树，轮排索引(轮排树)的空间要大4倍以上 (经验值)



# $k$ -gram 索引

- 比轮排索引空间开销要小
- 枚举一个词项中所有连读的 $k$ 个字符构成 $k$ -gram 。
- 2-gram称为二元组(**bigram**)
- 3-gram称为三元组(**trigram**)
- 例子: *April is the cruelest month*
- 2-gram: *\$a ap pr ri il l\$ \$i is s\$ \$t th he e\$ \$c cr ru ue el le es  
st t\$ \$m mo on nt h\$*
- 同前面一样，\$ 是一个特殊字符，表示单词开始或结束

# $k$ -gram索引

- 构建一个倒排索引，此时词典部分是所有的 $k$ -gram，倒排记录表部分是包含某个 $k$ -gram的所有词项
- 相当于对词项再构建一个倒排索引(二级索引)



## $k$ -gram (bigram, trigram, ...) 索引

- 需要注意的是，这里有两个倒排索引
- 词典-文档的倒排索引基于词项返回文档
- 而 $k$ -gram索引用于查找词项，即基于查询所包含的 $k$ -gram来查找所有的词项

# 利用2-gram索引处理通配符查询

- 例子：查询`mon*`
- 先执行布尔查询: `$m AND mo AND on`
- 该布尔查询会返回所有以前缀`mon`开始的词项...
- ...当然也可能返回许多伪正例(false positives), 比如 `MOON`。同前面的双词索引处理短语查询一样, 满足布尔查询只是满足原始查询的必要条件。因此, 必须要做后续过滤处理
- 剩下的词项将在词项-文档倒排索引中查找文档
- `k`-gram索引 vs. 轮排索引
- `k`-gram索引的空间消耗小
- 轮排索引不需要进行后过滤

# 课堂练习

- Google对通配符查询的支持极其有限
- 比如：在 Google中查询 [gen\* universit\*]
- 意图：想查 University of Geneva, 但是不知道如何拼写，特别是法语中的拼写
- 按照Google自己的说法, 2010-04-29: “\* 操作符只能作为一个整体单词使用，而不能作为单词的一部分使用”
- 问题: 为什么Google对通配查询并不充分支持？

# 原因

- 问题 1: 一条通配符查询往往相当于执行非常多的布尔查询
  - 对于 [gen\* universit\*]: geneva university OR geneva université OR genève university OR genève université OR general universities OR ...
  - 开销非常大
- 问题 2: 用户不愿意敲击更多的键盘
  - 如果允许[pyth\* theo\*]代替 [pythagoras' theorem]的话, 用户会倾向于使用前者
  - 这样会大大加重搜索引擎的负担
  - Google Suggest是一种减轻用户输入负担的好方法

# 提纲

词典

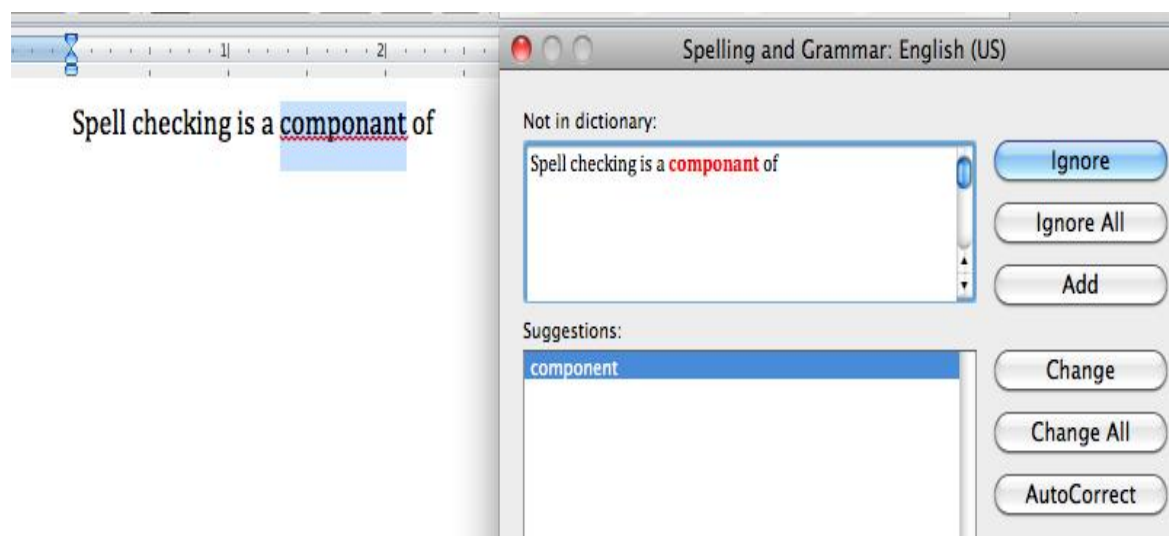
通配查询

拼写校正

Soundex

# 拼写矫正的应用

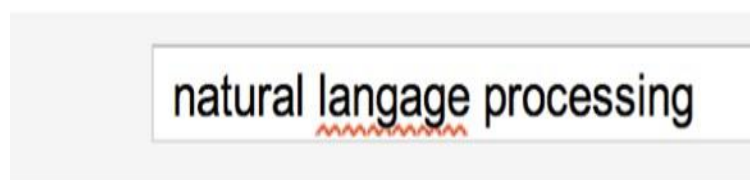
文字处理



手机



Web搜索



Showing results for natural language processing  
 Search instead for natural language processing



# 拼写错误率

拼写错误率取决于具体的应用, 大约在  $\sim 1 - 20\%$  之间

**26%:** Web queries [Wang et al. 2003](#)

**13%:** Retyping, no backspace: [Whitelaw et al. English&German](#)  
输入一篇Wikipedia文章, 不能使用回退/删除

**7%:** 手机输入, 允许纠正

**2%:** 手机输入, 不允许纠正 [Soukoreff & MacKenzie 2003](#)

**1-2%:** 正常键盘输入: [Kane and Wobbrock 2007, Gruden et al. 1983](#)

# 拼写涉及的任务

拼写错误检测

拼写错误矫正:

自动矫正

hte→the

矫正提示: 提示矫正选项

提示列表: 给出矫正候选列表

# 拼写错误种类

## 非词汇 (non-word) 错误

*graffe* → *giraffe* (长颈鹿)

## 真实词汇 (real-word) 错误

排印错误

*three* → *there*

认知错误（同/近音词）

*piece* → *peace*,

*too* → *two*

*your* → *you're*

非词汇错误的纠正一般来说不考虑上下文  
而真实词汇错误的纠正通常需要考虑上下文

# 拼写校正

- 两个主要用途
  - 纠正待索引文档
  - 纠正用户的查询
- 两种拼写校正的方法
  - 词独立(Isolated word)法
    - 只检查每个单词本身的拼写错误
    - 如果某个单词拼写错误后变成另外一个单词，则无法查出，  
e. g., *an asteroid that fell **form** the sky*
  - 上下文敏感(Context-sensitive)法
    - 纠错时要考虑周围的单词
    - 能纠正上例中的错误 *form/from*

# 非词汇拼写错误

## 非词汇拼写错误检测:

**词典**中不存在的词均视为错误

一般来说，词典越大越好

(Web很大，但是充满了拼写错误，因此并不是一个很好的词典)

## 非词汇拼写错误矫正

产生**候选**：与错误书写的单词相似的真实词汇

选择最好的候选词：

最短加权编辑距离

最高噪声通道概率

# 真实词汇 与 非词汇 拼写错误

对每个单词  $w$ , 产生候选集:

- 找到发音相似的候选词

- 找到拼写相似的候选词

- 将  $w$  也包括在候选集里

选择最佳候选词

- 将拼写错误视为（正确单词）经过噪声通道的输出

- 上下文敏感 – 需要考虑周围的文字是否合适，例如：

*Flying form Heathrow to LAX → Flying from Heathrow to LAX*

# 相关术语

我们之前讲过 字符二元组和 $k$ 元组 (*character bigrams and  $k$ -grams*) :

*st, pr, an ...*

同时也有 词项二元组和 $n$ 元组 (*word bigrams and  $n$ -grams*) :

*palo alto, flying from, road repairs*

# 词独立拼写矫正

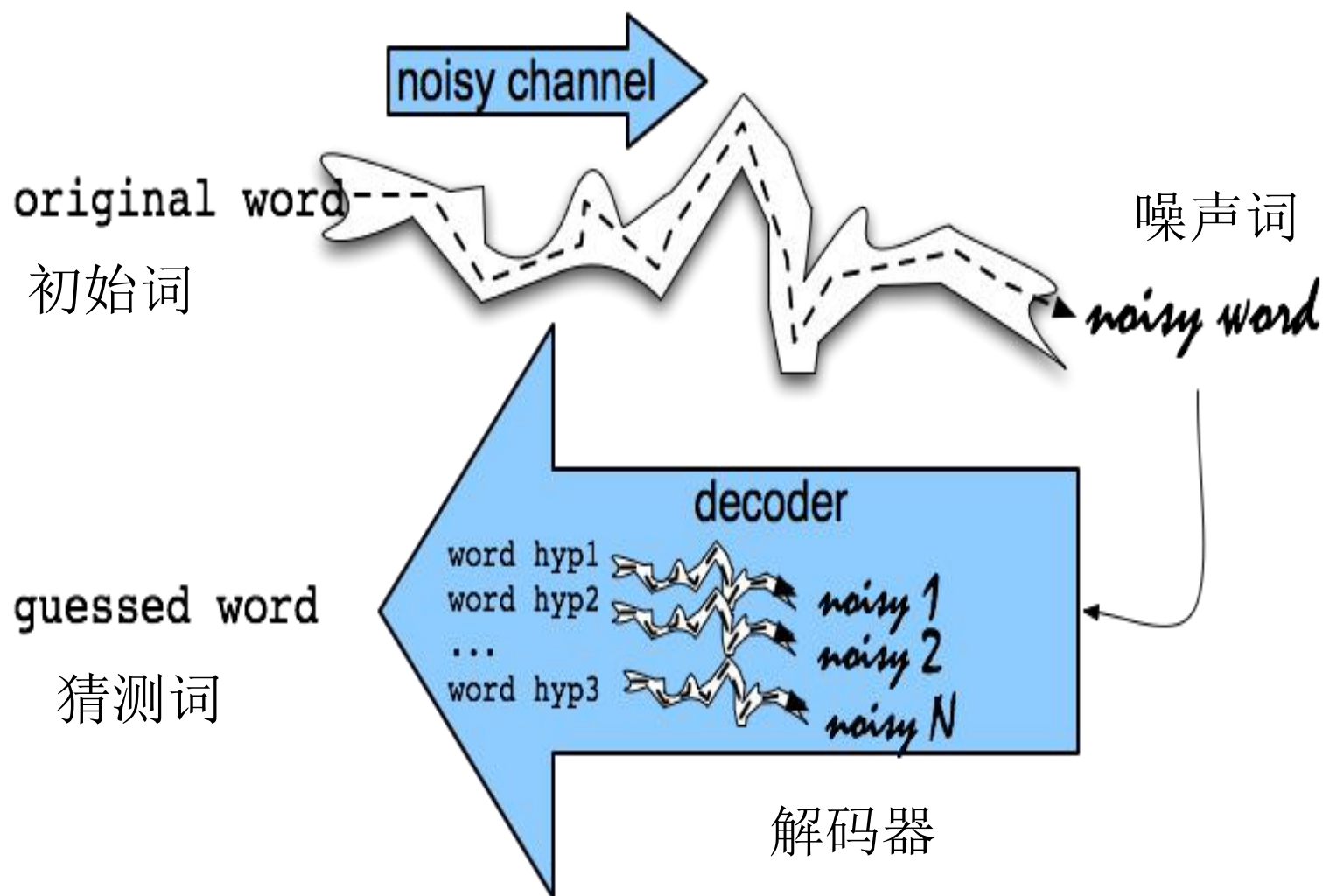
## INDEPENDENT WORD SPELLING CORRECTION

### 拼写噪声通道模型

### The Noisy Channel Model of Spelling



# 噪声通道 (noisy channel)



# 噪声通道 = 贝叶斯定理

观察到一个拼写错误的单词  $x$

找到正确词项  $\hat{w}$

在所有词项中寻找  $\hat{w}$ ,  
使得概率  $P(w/x)$  最大

$$\hat{w} = \operatorname{argmax}_{w \in V} P(w | x)$$

$$= \operatorname{argmax}_{w \in V} \frac{P(x | w) P(w)}{P(x)}$$



$$= \operatorname{argmax}_{w \in V} \underset{\substack{\uparrow \\ \text{Noisy channel model}}}{P(x | w)} \underset{\substack{\nwarrow \\ \text{Prior}}}{P(w)}$$

# 基于噪声通道的拼写矫正模型大约在1990年提出

## IBM

Mays, Eric, Fred J. Damerau and Robert L. Mercer. 1991.  
Context based spelling correction. *Information Processing and Management*, 23(5), 517–522

## AT&T Bell Labs

Kernighan, Mark D., Kenneth W. Church, and William A. Gale.  
1990. [A spelling correction program based on a noisy channel model](#). Proceedings of COLING 1990, 205-210

# 非词汇拼写错误的例子

acress - access?

# 候选词产生：两种情况

## 拼写相近的词

（与拼写错误词）编辑距离(edit distance) 较小的词

flom - from: edit distance=1

## 发音接近的词

# 候选词评价: Damerau-Levenshtein编辑距离

两个字符串的最小编辑距离。算法定义了四种距离：

Insertion （插入） informatin - informati**o**n

Deletion （删除） informati**n** - information

Substitution （替换） info**e**mation - informati**r**ation

Transposition of two adjacent letters

（两个相邻字母的交换） inf**r**omation - informati**o**n

## 与 access 编辑距离为1的词

Error	Candidate Correction	Correct Letter	Error Letter	Type
acress	actress	t	-	deletion
acress	cress	-	a	insertion
acress	caress	ca	ac	transposition
acress	access	c	r	substitution
acress	across	o	e	substitution
acress	acres	-	s	insertion

# 候选词产生

80% 的拼写错误到正确拼写的编辑距离 = 1

几乎所有拼写错误到正确拼写的编辑距离  $\leq 2$

除上述4种操作外，还可允许插入 **空格** 或 **连接符**

thisidea  $\rightarrow$  this idea

inlaw  $\rightarrow$  in-law

可允许词项合并

data base  $\rightarrow$  database

（上述两种操作使得）对于查询这样的短文本，可以将其视为一个完整的词项来检查拼写



# 怎样产生候选词?几种方法

1. 遍历词典，计算每一个词的编辑距离
2. 生成所有编辑距离  $\leq k$  (例如,  $k = 1$  或  $2$ ) 的词，然后与词典取交集
3. 建立一个字符 *k-gram* 索引，从词典中找到共享最多 *k-grams* 的词项（例如，基于Jaccard系数计算）

see IIR sec 3.3.4

4. 使用Levenshtein 有限状态转换机（Levenshtein finite state transducer）快速计算
5. 预先计算一个词项到可能的 正确词项/拼写错误 的映射表

# 一个（拼写矫正）范式 ...

找到最好的拼写矫正计算代价高，因此替代方案是

找到一个包含较高质量的拼写矫正子集

(例如编辑距离小于2)

从中选择最好的候选词

子集也许并不包括实际上的最优候选词

这是IR中一种常见范式，例如：

找到与查询最相关的文档, 找到与问题最符合的答案, 找到与用户最相关的广告 ...

首先找到一个好的候选集（find a good candidate set）

然后从中找到最优的前K个答案（Find the top *K amongst them* and return them as the best）

# 产生候选集: 回到贝叶斯定理

若有拼写错误的单词  $x$

找到其正确拼写  $\hat{w}$

$$\hat{w} = \operatorname{argmax}_{w \in V} P(w | x)$$

$$= \operatorname{argmax}_{w \in V} \frac{P(x | w) P(w)}{P(x)}$$

$$= \operatorname{argmax}_{w \in V} P(x | w) P(w)$$



What's  $P(w)$ ?

# Language Model（语言模型）

若有包含T个词条的大文本语料;

且 $C(w) = \# \text{ occurrences of } w$

$$P(w) = \frac{C(w)}{T}$$

在其它应用中，如果（静态）词典不能满足需要，  
可以采用经过适当过滤的用户输入查询训练语言模型。

# Unigram Prior probability (一元先验概率)

Counts from 404,253,213 words in Corpus of Contemporary English (COCA)

word	Frequency of word	$P(w)$
actress	9,321	.0000230573
gress	220	.0000005442
caress	686	.0000016969
access	37,038	.0000916207
across	120,844	.0002989314
acres	12,874	.0000318463

# 通道模型概率

Channel model probability

**Error model probability(错误模型概率), Edit probability (编辑概率)**

*Kernighan, Church, Gale 1990*

错误词项  $x = x_1, x_2, x_3 \dots x_m$

正确词项  $w = w_1, w_2, w_3, \dots, w_n$

$P(x/w)$  = probability of the edit

(deletion/insertion/substitution/transposition)

# 计算错误概率：混淆“矩阵”

```
del[x,y]:      count(xy typed as x)
ins[x,y]:      count(x typed as xy)
sub[x,y]:      count(y typed as x)
trans[x,y]:    count(xy typed as yx)
```

插入（ins）和删除（del）的概率条件依赖前一个字符

# 替换（sub）操作的混淆矩阵

sub[X, Y] = Substitution of X (incorrect) for Y (correct)																											
X	Y (correct)																										
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0	
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0	
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0	
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0	
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0	
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0	
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0	
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0	
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0	
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	5	0	0	0	0	0	0	0	0	
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3	
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0	
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0	
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2	
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0	
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0	
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0	
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1	
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6	
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0	
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0	
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	3	1	0	0	0	0	0	
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0	
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0	



# 邻近键

混淆矩阵构建也可以考虑键盘的邻近型，例如M容易被误打成N



# 混淆矩阵生成

[Peter Norvig's list of errors](#)

[Peter Norvig's list of counts of single-edit errors](#)

All Peter Norvig's ngrams data links: <http://norvig.com/ngrams/>

# Channel model （噪声通道模型）

$$P(x|w) = \begin{cases} \frac{\text{del}[w_{i-1}, w_i]}{\text{count}[w_{i-1} w_i]}, & \text{if deletion} \\ \frac{\text{ins}[w_{i-1}, x_i]}{\text{count}[w_{i-1}]}, & \text{if insertion} \\ \frac{\text{sub}[x_i, w_i]}{\text{count}[w_i]}, & \text{if substitution} \\ \frac{\text{trans}[w_i, w_{i+1}]}{\text{count}[w_i w_{i+1}]}, & \text{if transposition} \end{cases}$$

## 加一概率平滑 (add-1 probability smoothing)

上述混淆矩阵的例子很难避免某种操作样本数为0

这样会导致零概率问题：概率 $P(x|w)$ 为0

一个简单的解决方案是加一平滑：将所有的计数+1。若有一个包含  $|A|$  个字母的词表, 平滑后的概率:

$$\text{If substitution, } P(x|w) = \frac{\text{sub}[x, w] + 1}{\text{count}[w] + A}$$

# 单词 `acress` 的通道模型

Candidate Correction	Correct Letter	Error Letter	$x/w$	$P(x/w)$
actress	t	—	c   ct	.000117
cress	—	a	a   #	.00000144
caress	ca	ac	ac   ca	.00000164
access	c	r	r   c	.000000209
across	o	e	e   o	.00000093
acres	—	s	es   e	.0000321
acres	—	s	ss   s	.0000342

现代信息检索						
Candidate Correction	Correct Letter	Error Letter	$x/w$	$P(x/w)$	$P(w)$	$10^9 \cdot \frac{P(x/w)^*}{P(w)}$
actress	t	–	c   ct	.000117	.0000231	2.7
ress	–	a	a   #	.00000144	.000000544	.00078
caress	ca	ac	ac   c a	.00000164	.00000170	.0028
access	c	r	r   c	.000000209	.0000916	.019
across	o	e	e   o	.0000093	.000299	2.8
acres	–	s	es   e	.0000321	.0000318	1.0
acres	–	s	ss   s	.0000342	.0000318	$1.0^{62}$

Candidate Correction	Correct Letter	Error Letter	$x/w$	$P(x/w)$	$P(w)$	$10^9$ $*P(x/w)P(w)$
actress	t	–	c   c t	.000117	.00000231	2.7
cress	–	a	a   #	.000000144	.0000000544	.00078
caress	ca	ac	ac   ca	.000000164	.000000170	.0028
access	c	r	r   c	.0000000209	.00000916	.019
<b>across</b>	<b>o</b>	<b>e</b>	<b>e   o</b>	<b>.00000093</b>	<b>.000299</b>	<b>2.8</b>
acres	–	s	es   e	.00000321	.00000318	1.0
acres	–	s	ss	.00000342	.00000318	$1.0^{63}$

# 评价：一些拼写矫正数据集

[Wikipedia's list of common English misspelling](#)

[Aspell filtered version of that list](#)

[Birkbeck spelling error corpus](#)

[Peter Norvig's list of errors \(includes Wikipedia and Birkbeck, for training or testing\)](#)



# SPELLING CORRECTION WITH THE NOISY CHANNEL

噪声通道拼写矫正

**Context-Sensitive Spelling  
Correction**

**上下文敏感的拼写矫正**

# 实际应用中的拼写错误

...leaving in about fifteen **minuets** (小舞蹈节目?) to go to her house. **minutes**

The design **an** construction of the system... **a**

Can they **lave** (洗澡?) him my messages? **leave**

The study was conducted mainly **be** John Black. **by**

25-40% 的拼写错误是真实词汇 **Kukich 1992**

对于这一类拼写错误，需要考虑上下文

# 上下文敏感的拼写矫正

For each word in sentence (phrase, query ...)

Generate *candidate set* (产生候选集)

the word itself (单词本身)

all single-letter edits that are English words (所有经过单字母编辑后是英文单词的单词)

words that are homophones (同音异形异义词)

(all of this can be pre-computed! 这些都可以预先线下计算)

Choose best candidates (选择最佳候选词)

Noisy channel model (采用噪声通道模型)

# 真实词汇拼写矫正的噪声通道

- 给定句子  $x_1, x_2, x_3, \dots, x_n$
- 为每个词  $x_i$  产生一个候选词集合
  - $\text{Candidate}(x_1) = \{x_1, w_1, w'_1, w''_1, \dots\}$
  - $\text{Candidate}(x_2) = \{x_2, w_2, w'_2, w''_2, \dots\}$
  - $\text{Candidate}(x_n) = \{x_n, w_n, w'_n, w''_n, \dots\}$
- 选择序列  $W$  使得  $P(W|x_1, \dots, x_n)$  最大

$$\begin{aligned}\hat{W} &= \operatorname{argmax}_{W \in V} P(W|x) \\ &= \operatorname{argmax}_{W \in V} P(x|W) P(W)\end{aligned}$$

结合上下文词汇：上下文敏感的单词矫正

要决定正确的单词是 **actress** 还是 **across** （两者都是真实词汇）需要考虑上下文

需要一个更好的 **language model** （语言模型）

在自然语言处理/计算语言学课程中会涉及到很多相关的知识

这里仅仅简单介绍语言模型的基础知识

**bigram language model** （二元语言模型）中文本片段  $w_1...w_n$  的概率的计算：  $w_i$  的概率仅仅取决于前一个单词

$$P(w_1...w_n) = P(w_1)P(w_2|w_1)...P(w_n|w_{n-1})$$

# 结合上下文词汇

对于一元词项频率,  $P(w)$  始终非零

因为词典是从文本语料本身产生的

但是  $P(w_k | w_{k-1})$  并非如此, 因此需要 平滑

可以使用加一平滑

但是是一种更好的方法是将一元模型与二元模型插值:

$$P_{li}(w_k | w_{k-1}) = \lambda P_{uni}(w_k) + (1-\lambda)P_{bi}(w_k | w_{k-1})$$

$$P_{bi}(w_k | w_{k-1}) = C(w_{k-1}, w_k) / C(w_{k-1})$$

# 几个需要注意的要点

我们有几种单词(一元, 二元)的概率分布

Keep them straight! (使用直接的概率似然估计)

使用对数概率:

$$\log P(w_1 \dots w_n) = \log P(w_1) + \log P(w_2 | w_1) + \dots + \log P(w_n | w_{n-1})$$

否则会有浮点下溢的问题 (floating point underflow)

查询词可能会在一篇文档中到处出现

首先统计一元词项频率, 同时统计二元词项频率

通常人们会在字符串之前使用一个“\$”字符表示字符串的开头, 但是在拼写矫正中并不需要这样做

因此, 需要注意的是, 一元和二元词项的词频总和并不相等 - 虽然这本身并不是一个问题

# 使用二元语言模型

"a stellar and versatile **actress** whose combination of sass and glamour.."

基于现代美国英语语料，使用加一平滑得到的二元词项概率：

$P(\text{actress} \text{versatile}) = .000021$	$P(\text{whose} \text{actress}) = .0010$
$P(\text{across} \text{versatile}) = .000021$	$P(\text{whose} \text{across}) = .000006$

$P(\text{"versatile actress whose"}) = .000021 * .0010 = 210 \times 10^{-10}$

$P(\text{"versatile across whose"}) = .000021 * .000006 = 1 \times 10^{-10}$



# 使用二元语言模型

"a stellar and versatile **actress** whose combination of sass and glamour..."

基于现代美国英语语料，使用加一平滑得到的二元词项概率：

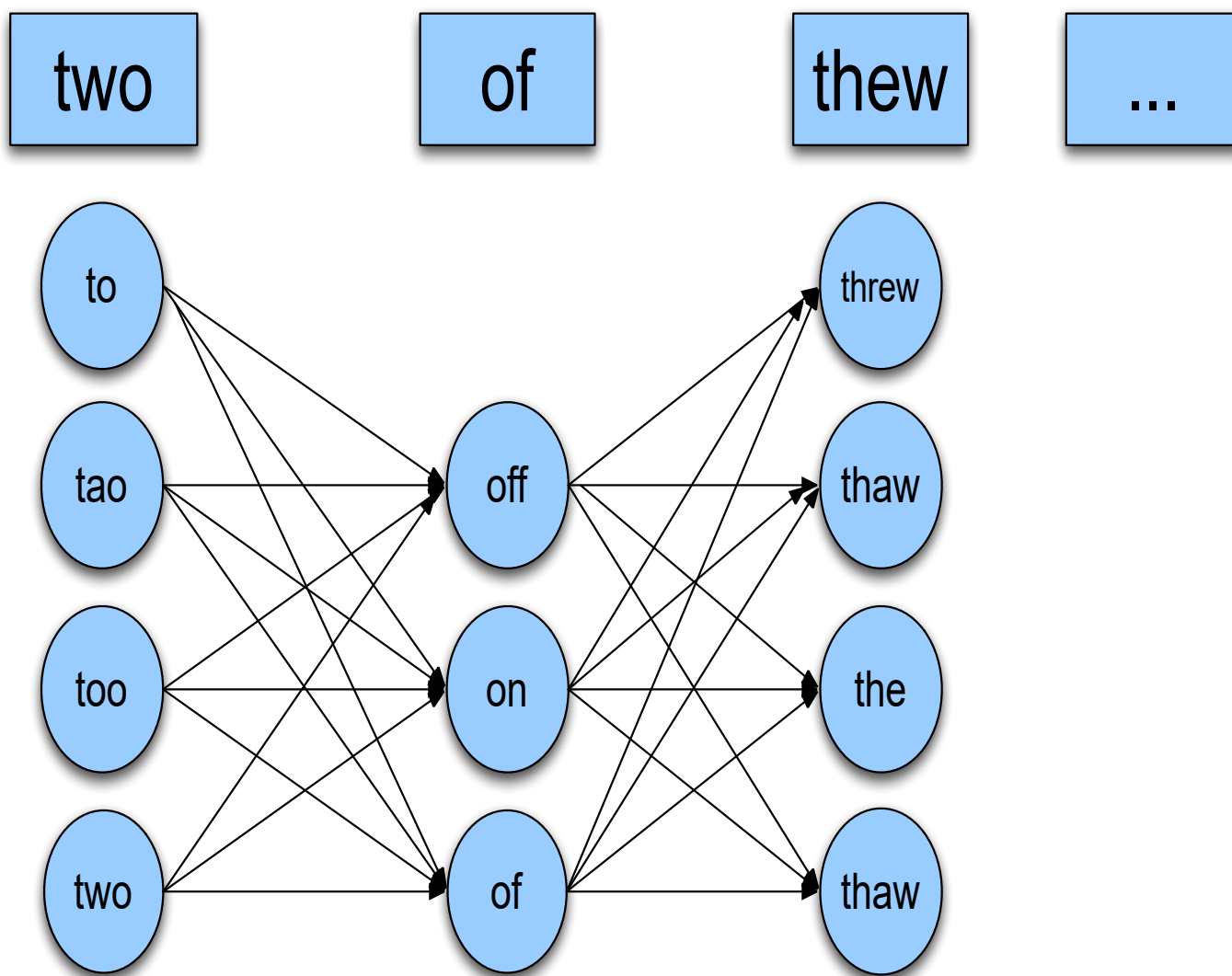
$P(\text{actress}|\text{versatile}) = .000021$     $P(\text{whose}|\text{actress}) = .0010$

$P(\text{across}|\text{versatile}) = .000021$     $P(\text{whose}|\text{across}) = .000006$

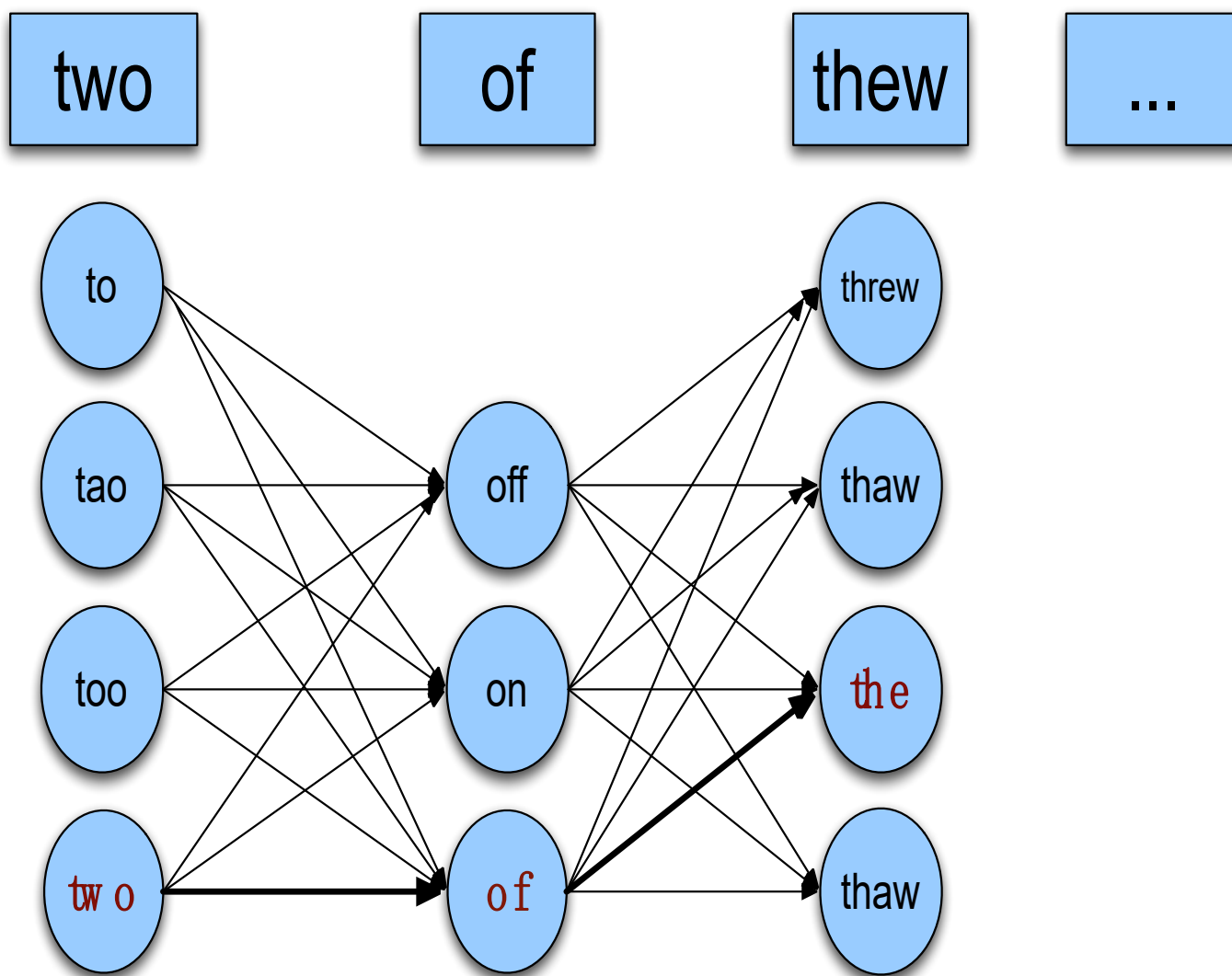
$P(\text{"versatile actress whose"}) = .000021 * .0010 = 210 \times 10^{-10}$

$P(\text{"versatile across whose"}) = .000021 * .000006 = 1 \times 10^{-10}$

# 真实词汇拼写矫正的噪声通道模型



# 真实词汇拼写矫正的噪声通道模型



# 简化：每句话仅考虑一个错误

只考虑仅有一个词不同的句子

$w_1, w''_2, w_3, w_4$       two **off** thew

$w_1, w_2, w'_3, w_4$       two of **the**

$w'''_1, w_2, w_3, w_4$       **too** of thew

...

选择使得 概率 $P(W)$ 最大的序列 $W$

# 怎样计算概率

## 语言模型

- 一元 (Unigram)

- 二元 (Bigram)

- 三元 (trigram) 等等...

## 通道模型

- 与非词汇拼写矫正相同

- 需要额外考虑无拼写错误概率,  $P(w/w)$

# 无错误概率

什么是正确拼写词的通道概率（What is the channel probability for a correctly typed word）？

$P(\text{"the"} | \text{"the"})$

在大语料的支持下，可以准确的估计概率

但是概率值很大程度上取决于应用

.90 (1 error in 10 words)

.95 (1 error in 20 words)

.99 (1 error in 100 words)

# Peter Norvig's “thew (肌肉力量)” example

x	w	x w	P(x w)	P(w)	$10^9 P(x w)P(w)$
thew	the	ew e	0.0000007	0.02	144
thew	thew		0.95	0.000000009	90
thew	thaw	e a	0.001	0.00000007	0.7
thew	threw	h hr	0.0000008	0.0000004	0.03
thew	thwe	ew we	0.0000003	0.000000004	0.0001

# 当前最佳的噪声通道

We never just multiply the prior and the error model（不把先验概率和错误模型相乘）

Independence assumptions  $\rightarrow$  probabilities not commensurate（这是因为：由于独立性假设，这两个概率不相称）

Instead: Weight them（替代方案是为概率赋权重）

$$\hat{w} = \operatorname{argmax}_{w \in V} P(x|w) P(w)^\lambda$$

Learn  $\lambda$  from a development test set（在开发集上学习参数 $\lambda$ ）（由于使用对数概率， $\lambda$ 其实是一个线性系数）



# 通道模型的改进

允许更丰富的编辑操作 (Brill and Moore 2000)

ent → ant

ph → f

le → al

这三个都是同音节替换

将发音融入到通道模型中 (Toutanova and Moore 2002)

将设备融入到通道模型中

（虽然）并非所有的安卓手机都需要相同的错误模型  
但是拼写矫正方案可以在系统级别进行设计（即针对不同的系统/输入法 设计/训练不同的错误模型）