Preprocessing

- Sentence segmentation
- Tokenization
- Word normalization
  - Derivational vs. inflectional morphology
  - Lemmatisation vs. stemming
- Stop words

C1. Preprocessing

1. Sentence segmentation: 断句
   punctuation 标点
   capital 大写
   lexicons 字典
   machine learning

2. Tokenization 断词
   alphabetic strings (\w+)
   Abbreviations
   Hyphens 连字符
   Numbers Dates
   Clitics (n't in can't)
   Internet language
   Multiword units

3. Word normalization 归一化，所有形式归为一种表示
3.1 Derivational vs. inflectional morphology 派生词和词形变化
   affixes & suffixes 前缀后缀
   inflectional suffixes：grammatical roles ：-s, -ed, -ing
   derivational suffixes：change around the way the word: -ly, -ist, -er

3.2 Lemmatisation vs. stemming
   Lemmatisation : remove inflections 找原型，是词
   stemming：remove suffixes 去词缀，不是词

4. stop words：get in the way of retrieval

Information retrieval foundations

- "Information need"
- TF*IDF weighting, components
  - Cosine similarity
- Efficient indexing
- Querying algorithm

C2 Information retrieval foundations

1. "Information need" vs query

Query is a way of representing an information need in order to interact with an engine

information need is a more general concept drives how you interact and drives how evaluate work：TFIDF

2.TF*IDF weighting, components 一个 term 和一个 doc 的关系值

query processing in VSM 检索并排序

1). make the query a pseudo-doc

Example

Corpus:

	two	tea	me	you
doc1	2	2	0	0
doc2	0	2	1	1
doc3	0	0	2	2

Query: **tea me**

	two	tea	me	you
query	0	1	1	0

2). TF-IDF's cos simliarity(may normalize 归一化处理，虽然算夹角，不归一化也一样，但是最后分母是 1，不用特别处理)

Example: TDM

- Corpus represented as TF x IDF matrix

	two	tea	me	you
doc1	3.17	1.16	0	0
doc2	0	1.16	0.58	0.58
doc3	0	0	1.16	1.16

e.g., |doc1| = (3.17^2 + 1.16^2 + 0^2 + 0^2)^(0.5)

- Normalise TF x IDF

	two	tea	me	you
doc1	0.93	0.34	0	0
doc2	0	0.82	0.41	0.41
doc3	0	0	0.71	0.71

normalised matrix has row for doc/|doc1|

  - divide each row by vector magnitude, |row|

Example: query similarity

normalised TDM

	two	tea	me	you
doc1	0.93	0.34	0	0
doc2	0	0.82	0.41	0.41
doc3	0	0	0.71	0.71

query as unit vector

	two	tea	me	you
q	0	0.71	0.71	0

- Compute dot product for each document
  - doc1.q = 0 * 0.93 + 0.71 * 0.34 + 0 * 0.71 + 0 * 0 = 0.24
  - doc2.q = 0 * 0 + 0.71 * 0.82 + 0.71 * 0.41 + 0 * 0.41 = 0.87
  - doc3.q = 0 * 0 + 0.71 * 0 + 0.71 *0.71 + 0 * 0.71 = 0.5
- Rank by cosine score: doc2 > doc3 > doc1

3). rank

3.Efficient indexing 上面的检索方法效率低复杂度ΣdΣt ，长乘宽，矩阵式复杂度，正排索引

Inverted Index 倒排索引

1). row: terms

Values as lists of (docID, weight)

weight: normalized TF*IDF

two	→	1: 0.88	
tea	→	1: 0.47	2: 0.9
me	→	2: 0.32	3: 0.71

2). 新复杂度 Σt Dft 链表式复杂度，倒排索引

4. Querying algorithm

Querying an inverted index

Assuming normalised TF*IDF weights:
Set accumulator $a_d \leftarrow 0$ for all documents d
for all terms t in query **do**
Load postings list for t
for all postings $\langle d, w_{t,d} \rangle$ in list **do**
$a_d \leftarrow a_d + w_{t,d}$
end for
end for
Sort documents by decreasing a_d
return sorted results to user

$w_{t,d}$ denotes normalised $tf_{d,t} \times idf_t$ vector

一个文件对应一个累加器，逐个检索 t，t 对应所有文件的值加到各自对应累加器上去，把累加器最终的结果排序，就是文件的排序

tea	→	1: 0.47	2: 0.9
me	→	2: 0.32	3: 0.71

- a_d = < 0, 0, 0 >
- term ***"tea"***
  - a_d[1] += 0.47
  - a_d[2] += 0.9
  - end of block, a_d = < 0.47, 0.9, 0 >
- term ***"me"***
  - a_d[2] += 0.31
  - a_d[3] += 0.71
  - end of block, a_d = < 0.47, 1.21, 0.71 >
- sort to produce doc ranking
  - 2: 1.21; 3: 0.71; 1: 0.47 → [2, 3, 1]

3 个累加器，逐个检索单词，更新累加器，最终结果排序

IR indexing and querying

- Posting list compression
  - Use of gaps between document ids
    - vbyte encoding
    - opt-pfor-delta encoding
- WAND algorithm
- Index construction: static vs incremental
- Phrase search
  - positional index (intersection, extra information etc.)
  - **NOT** suffix array

C3 IR indexing and querying

1. Posting list compression

1).Posting list Compression

posting list: a sequence of int 序列，递增顺序

the int: [1, N] require log2(N) bits

gap: 各 id 之间的差值代替 id 本身，可以减少 bits，只有第一个 gap 就是 id 不用改

the	ids:	25	26	29	…	12345	12347
	gaps:	25	1	3	…	1	2
house	ids:	5123	5234	5454	5591	…	
	gaps:	5123	1	220	137	…	
aeronaut	ids:	251235	251239	251239			
	gaps:	251235	4	34			

2). Variable Byte Compression

def: n bytes (7+1 bits) represent int

Examples

Number	Encoding	
824	00111000	10000110
5	10000101	

Storage Cost

Number Range	Number of Bytes
0 – 127	1
128 – 16383	2
16384 – 2097151	3

用低 7 位代表数值，高 1 位代表是否是最后一个 byte，1 是，0 否，高 1 位在编码中不写出来

algorithm:

Encoding

```
1: function ENCODE(x)
2:     while x >= 128 do
3:         WRITE(x mod 128)
4:         x = x ÷ 128
5:     end while
6:     WRITE(x + 128)
7: end function
```

Decoding

```
1: function DECODE(bytes)
2:     x = 0, s = 0
3:     y =READBYTE(bytes)
4:     while y < 128 do
5:         x = x ^ (y << s)
6:         s = s + 7
7:         y =READBYTE(bytes)
8:     end while
9:     x = x ^ ((y − 128) << s)
10:    return x
11: end function
```

编码：不断除以 128，余数当低位编码，拿商继续除以 128，余数比较高位的编码，循环往复

解码：不断读字节中的低 7 位，并右移，与 x 异或，覆盖 x 对应的位置

Compress number 512312 or 11111010001001110000 in binary.

- How many bytes? 11111|0100010|0111000. 3 bytes!
- How do we compress the number?
  1. Extract the lowest 7 bits.
     $512312 \mod 128 = 56 = 0111000$
  2. Discard lowest 7 bits.
     $512312 \div 128 = 4002$ (or $512312 >> 7$)
  3. Extract the lowest 7 bits.
     $4002 \mod 128 = 34 = 0100010$
  4. Discard lowest 7 bits.
     $4002 \div 128 = 31$ (or $4002 >> 7$)
  5. Number smaller than 128. Write in lowest 7 bits and set top bit to 1. $31 = 11111$ So we write 10011111 which is $31 + 128 = 159$

x = 0, s = 0, y = 0111000

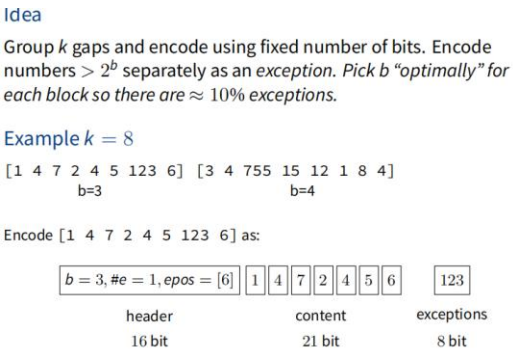
x = 0000000^0111000, s = 7, y = 0100010

x = 0111000^0100010 0000000=0100010 0111000, s = 14, y = 11111

x = 0100010 0111000^11111 0000000 0000000 = 11111 0100010 0111000, s = 21, y = 10000000

3). OptPForDelta Compression

基于 gap 的



k = 8 假设索引块有 8 个 gap，设定 b = 3，即大于 b^3 的数为 exception，选择 b 一般使 exception 大约占 10%，其他索引给正常小空间，异常值单独分配大空间

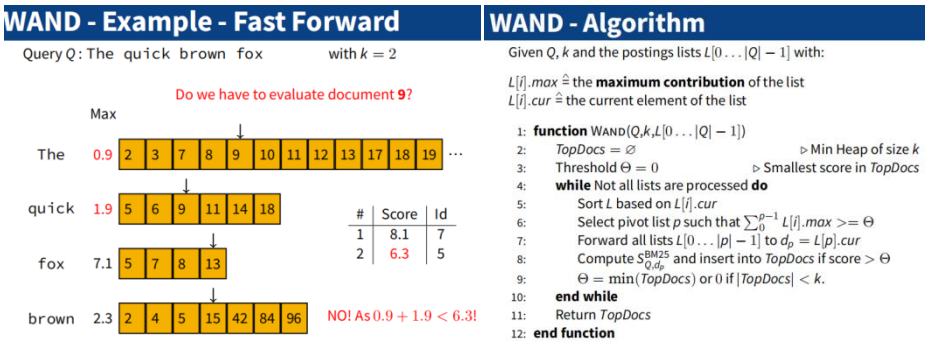
2. Top-k - The WAND Algorithm

1). 要素：避免遍历非 top-k 的 doc，依据 similarity metric (BM25), block based compression, GEQ(x)

2). 策略：

列出倒排索引，求出各 term 的最大分数，按照 doc id 和分数排序，从小到大。

遍历各 doc id，如果 term 的最大分数和小于 top-k，跳过这个 id；如果不是，求 doc 对应的分数和再与 top-k 比较，比完再排序。



3. Index construction

Inverted Index construction 倒排索引创建

(1) Static scenario: 静态情景（设想，非真实）

1. Static collection of documents
2. Offline construction
3. Fixed at query time (no delete or append)

(2) Real world scenario: 真实情景

1. Document collection grows over time 文件增加
2. Documents arrive at a given rate (1000 docs/second) 速率

3. Documents should be searchable immediately 马上就能搜索

4. Might want to delete or update documents 文件增删

(3) static construction 静态地创建 block based

Constraints: 条件

1. Static collection of documents (e.g. 1TB webcrawl) 静态收集，爬取

2. Offline construction (only query after construction) 离线创建

3. Fixed at query time (no modify or append) 搜索时固定

Constraints and Properties: 条件和性质

1. Fixed amount of memory available 内存固定（有限，limited）

2. Dataset will not fit in memory 数据集和内存不匹配，数据集太大，RAM 太小了

3. Final index small compared to dataset 索引需要比数据集小

Algorithm:

Algorithm:

Split document collection into batches (X documents per batch) and construct inverted indexes for each batch. Merge batches at the end.

```
def ConstructStatic(Collection):
    n = 0
    vocab = {} # map term to term_id
    # split collection into fixed size batches
    for batch in Collection:
        # invert each batch
        B = InvertBatch(batch,vocab)
        storeToDisk(B)
        n = n + 1
    # merge all inverted batches
    MergeAllInvertedBatches(B0,B1...,Bn)
```

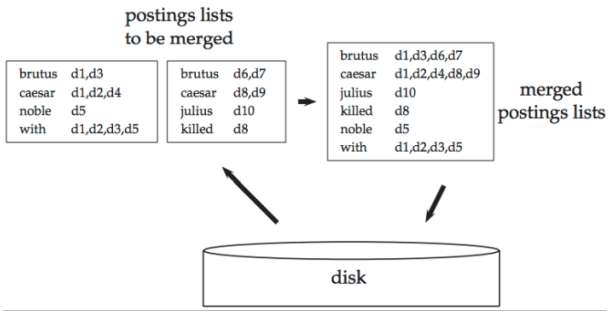
vocab 是 global 的，把所有 doc 分成各个 batch，分别创建倒排索引，存入 disk，

Stored on disk:

Term-ID	Compressed size	Compressed posting
5	52	<vbyte>
7	454	<vbyte>
9	432	<vbyte>
12	5324	<vbyte>

再合并

合并：



1. Open all n files containing batches on disk 打开所有文件

2. Read one term (or a few) from each file 读取一个 term

3. Perform n-way merge, merging terms with same ids 合并所有文件中的同一个 term

4. Merge equivalent to appending bytes as document ids are increasing 存储随着 doc id 一起增加

5. Read the next terms 读下一个 term

(4) From static to incremental indexing 从静态到增量索引（dynamic, receive new doc）

1. One static large static index on disk 原本有一个索引在内存中

2. As new documents arrive keep them in-memory in second index 新文件在第二个索引中

3. If second index becomes too big, merge with static index 如果第二个太大，和第一个合并

4. Query both indexes and merge results 搜索这两个 index，合并结果

problem (cost):

合并的时候每个条目都要检索一遍，而每个 index 看起来就像是均分的，一达到 n，就创建新的，总共 N/n 个 indexs，cost 为 O(N^2/n)

Problem

1. After one year our index has $N = 10$ billion postings
2. At each time we have an inverted index in memory which is merged it has more than $n = 1$ million postings
3. Total merges = N/n. At each merge we look at all existing postings.
   Thus,

$$totalcost = n + 2n + 3n + 4n + \ldots + (N/n)n = O(N^2/n)$$

Example: $N = 10$ billion, $n = 1$ million = $N^2/n = 10B^2/1M = 100$ trillion I/Os

n: disk: 0 memory: n

2n: disk:n memory: n

3n: disk:2n memory: n

要做 N^2/n 次的 IO opreations

idea:

index 大小指数上升，先不断创建上限为低指数的 2^i*n 大小的 index，当这种 index 数目达到某个指数等级，例如 8 个或 16 个 2^2*n 的 index，将他们合并，不光低级的达到数目了要合并，高级也要不断注意合并

Idea

1. Use a logarithmic number ($\log N$) of indexes. At each level i, store index of size $2^i \times n$
2. Query all $\log N$ indexes at the same time and merge results
3. Construction cost: $N \log(N/n)$

incremental construction: Lucene

distributed construction: Elasticsearch （不学）

IR Querying, Evaluation and L2R

- Query completion
  - trie+RMQ algorithm
  - Motivation, Data sources
- Relevance feedback (why, types)
- Evaluation methods
  - precision @ k, (Mean)AveragePrecision, RBP
  - research test collections
- Reranking IR system outputs using learned classifier

C4 IR Querying, Evaluation and L2R

1. Query Completion 查询补全

(1) goal: formulate search results, reduce keystrokes, help spelling, guild what is good query, cache (reduce server load)

strategy: a list of completion for partial Q, refine suggestions, stop when selected or fail, language model

(2) high level algorithm 算法

query pattern P

1) a set of candidates matching P from a set (S) of possible querys 从一些可能的 Q 中找备选

2) rank by freq 排序

3) re-rank by more complex measure

4) return top-k

(3) Completion Targets 补齐的目标（素材）

S come from:

1. Most popular queries (websearch)
2. Items listed on website (ecommerce)
3. Past queries by the user (email search)

Properties:

1. Static (e.g. completion for “twi”)
2. Dynamic (e.g. time-sensitive, “world cup”)
3. Massive or small (email search vs websearch)

(4) Completion Types (Modes) 种类

Modes:

1. Prefix match. 补齐（后半部分，前半完整）
2. Substring match. 子串（不求首尾完整）
3. Multi-term prefix match. 多 t 补齐
4. Relaxed match. 松弛匹配（补齐，不求相同，松弛到相似）

Example: Target “FIFA world cup 2018“:

P	Mode 1	Mode 2	Mode 3	Mode 4
FIFA wo	x	x	x	x
orl		x		
FI wor			x	x
FIFO warld cu				x

(5) Prefix Completion 补齐 --Trie+RMQ based Index 字典树+最值区间查询

Problem: give P, retrive top-k

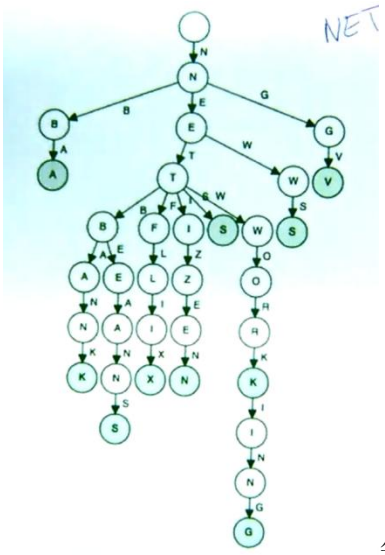
Data: query log consists queries received by search index 查询记录都统计在索引中

Requirement: fast, space efficiency

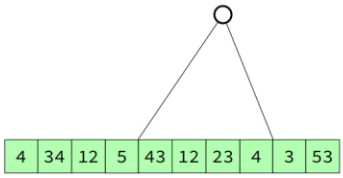
step1: sort, count freq 排序并统计

Before	After
bunnings	< *bunnings*, 47 >
bachelor in paradise	< *big w*, 5 >
bbc news	< *bbc news*, 12 >
bunnings	< *bachelor in paradise*, 2 >
big w	
bbc news	
big w	

step2: insert all queries & freq into a trie 字典树



每一条 string 代表一个编号，储存一个 freq，组成一个连续数组，insert a pattern P：O（|P|）time。每一些子树对应于一个搜索范围，比如



RMQ：

simple algorithm: 不适用

array A, range [l, r], size m

复制 A 到 B，花费 O(m)

对 B 排序，花费 O(m*logm)

return 前 k 个最大数的位置，即对应的 string

问题：时间取决于 m，m 可能很大，另外还需要 O(m)的额外空间

RMQ index: O(1) time

array A, range [i, j], size n

有 O(n^2)个不同的子范围，从 1 加到 n-1 个

针对每一个子范围，预先计算好对应的 max 的位置，用 O(n^2) 空间

找到 A[i, j]的 max 的位置 P，计算 A[i, p-1]和 A[p+1, j]的 max 位置，如此递归直 到找到前 k 个最大

RMQ index- Reduce space:

对于每个 A[i]，不是逐个计算 A[i, i+1] A[i, i+2] A[i, i+3] ...，O(n^2)个

而是计算 A[i, i+1] A[i, i+2] A[i, i+4], A[i, i+8] ...，O(n*logn)个

当查询任意区间 A[l, r]时，分解成 A[l, Y] and A[Z,r] where Y = l + 2^x and Z = r

- 2^y

A[l, Y], A[Z,r]将会重叠，取并集的 max

总空间：O(n*logn)

正好回应了前面的 Requirement: fast, space efficiency

2.Relevance feedback (why, types)

(1) why 要么调整术语的原始查询权重，或者使用这些内容添加到查询词。

(2)types:

user revelance 显式反馈，用户选

pseudorelevance 伪相关反馈，也被称为盲相关反馈，去做一个正常的检索，从 top-k 中的 doc 中选出一些 topic 或者 term，再扩展搜。无需用户 feedback

Indirect relevance 隐式反馈，根据 user 的 click，用 2-rank 神经模型学习

3. Evaluation methods

recall: hard to calculate

precision:

* precision @ k: compute precision using only ranks 1 .. k

* average precision (AP): take average over prec@k for each k where rank k item is relevant; measure becomes rank sensitive

* Mean Average Precision (MAP): AP averaged across multiple queries

Precision: $\frac{\text{number of returned relevant results}}{\text{number of returned results}}$

Recall: $\frac{\text{number of returned relevant results}}{\text{total number of relevant results}}$ (often useless in an IR context)

Precision at k ($P@k$): $\frac{\text{number of returned relevant results in top } k}{k}$

(Recall at k usually not meaningful)

Average Precision: $\frac{1}{R}\sum_{k|d(k)\text{is relevant}} P@k$

where R is the total number of relevant documents for the query (denominator of Recall)

Typically averaged over many queries: MAP (Mean Average Precision)

- Relevance vector

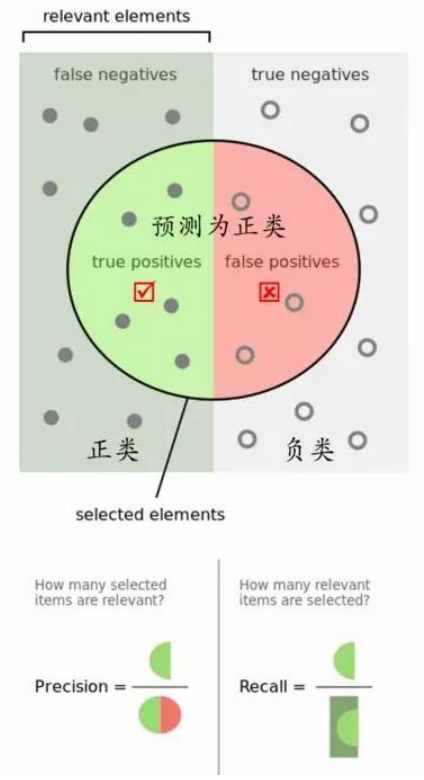
  < 1, 0, 0, 0, 0, 1, 0, 1, 0, 0 >

- Precision
  - P@1 = 1/1P@2 = ½ P@3 = 1/3 P@4 = ¼
    P@5 = 1/5P@6 = 2/6 P@7 = 2/7 P@8 = 3/8
    P@9 = 3/9P@10 = 3/10

- AP (average precision) = 1/3 *(P@1 + P@6 + P@8) = 0.57
  (assuming only 3 docs are relevant, giving 1/3 scale)

- Results then averaged over all queries in test collection
  (mean average precision, MAP).

relevance 返回的结果中，相关结果为 1
precision=返回的相关结果/返回的所有结果
recall=返回的相关结果/所有相关结果
precision@k=top-k 中返回的相关结果/k
AP=(刚好包含一个相关结果的 p@k 全部相加)/所有相关结果
MAP=AP/所有搜索



*Reciprocal rank

Reciprocal rank

- Reciprocal rank = 1 / rank of first correct answer
- Examples:
  - relevance < 1, 0, 0, 0, 0, 1, 0, 1, 0 >
    RR = 1 / 1 = 1
  - relevance < 0, 0, 1, 0, 1, 0, 0, 0, 1 >
    RR = 1 / 3 = 0.33

对第一个相关结果敏感

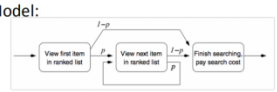
*Rank-biased precision（RBP）

Rank-biased precision

- RBP Formula (r_i is the ith element of the relevance vector of length d)

$$RBP = (1-p) \times \sum_{i=1}^{d} r_i \times p^{i-1}$$

- User Model:



- Patient user: p = 0.95, Impatient user: p=0.50

根据用户的耐心（概率）多次查询，将概率与分数和相乘得到期望值

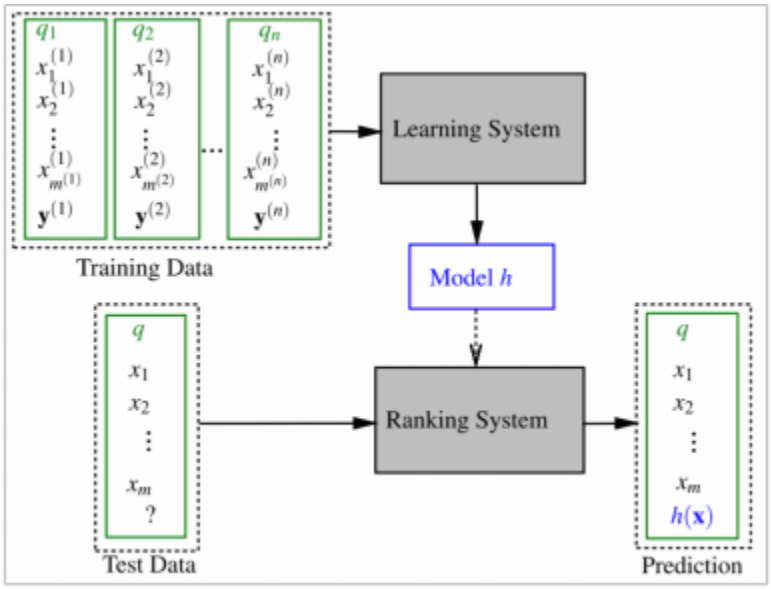
4. Reranking IR system outputs using learned classifier

re-rank doc: learning to rank

Given queries, m (k before) documents documents for each query, click data (or human judgements) use Machine Learning techniques to rank documents

学习人们对于检索结果的点击率重新排序

training data: tuple(q,d,u,r) query,doc,user,reward(rank mark)



Text Classification

- Building a classification system
- Evaluation metrics
- Algorithms
- Text classification tasks
  - including learning-to-rank in IR

C5 Text Classification

1.Building a classification system

1. Identify a task of interest
2. Collect an appropriate corpus
3. Carry out annotation
4. Select features
5. Choose a machine learning algorithm
6. Tune hyperparameters using held-out development data
7. Repeat earlier steps as needed
8. Train final model
9. Evaluate model on held-out test data

2. Evaluation metrics

classification 和 rank 的 evaluation 不同

accuracy = correct/total

most common class baseline accuracy = 最多的 predicted 分类总数/total

precision = positive correct/predicted positive total

recall = positive correct/true positive total

Evaluation: Accuracy

	Classified As	
Class	A	B
A	79	13
B	8	10

Accuracy = correct classifications/total classifications

= (79 + 10)/(79 + 13 + 8 + 10)

= 0.81

0.81 looks good, but most common class baseline accuracy is

= (79 + 13)/(79 + 13 + 8 + 10) = 0.84

B as “positive class”

Precision = correct classifications of B (tp)
/ total classifications as B (tp + fp)
= 10/(10 + 13) = 0.43

Recall = correct classifications of B (tp)
/ total instances of B (tp + fn)
= 10/(10 + 8) = 0.56

F(1)-score: Harmonic mean of precision and recall 调和平均数，倒数平均数

$$H_n = \frac{1}{\frac{1}{n}\sum_{i=1}^{n}\frac{1}{x_i}} = \frac{n}{\sum_{i=1}^{n}\frac{1}{x_i}}$$

F1 = 2 precision*recall/(precision + recall)

3.Algorithms

- Bias vs. Variance

  欠拟合 Underfit || 高偏差 High bias

  正常拟合 Just right || 偏差和方差均较小

  过拟合 Overfit || 高方差 High variance

- Feature independence
- Feature scaling
- Complexity
- Speed

1.Naïve Bayes

pro:

con:

2. Logistic Regression

3. SVM

4.K-Nearest Neighbour

5.Decision tree

6.Random forests

7.Neural Networks

4.Text classification tasks

examples:

* Topic classification
* Sentiment analysis
* Authorship attribution
* Native-language identification
* Automatic fact-checking

(1) Topic classification 图书馆学，信息检索

features:

* Unigram bag of words (BOW), with stop-words removed
* Longer n-grams (bigrams, trigrams) for phrases

BOW 词袋模型就是把所有词放进一个数组里作为一个向量模型用来映射

例句:

Jane wants to go to Shenzhen.

Bob wants to go to Shanghai.

一、词袋模型

将所有词语装进一个袋子里，不考虑其词法和语序的问题，即每个词语都是独立的。例如上面2个例句，就可以构成一个词袋，袋子里包括Jane、wants、to、go、Shenzhen、Bob、Shanghai。假设建立一个数组（或词典）用于映射匹配

```
[Jane, wants, to, go, Shenzhen, Bob, Shanghai]
```

那么上面两个例句就可以用以下两个向量表示，对应的下标与映射数组的下标相匹配，其值为该词语出现的次数

```
[1,1,2,1,1,0,0]
[0,1,2,1,0,1,1]
```

这两个词频向量就是词袋模型，可以很明显的看到语序关系已经完全丢失。

(2)Sentiment Analysis 情感分析/挖掘，商业分析

Features

* N-grams
* Polarity lexicons 极性（倾向性）字典，正面/负面

(3)Authorship attribution 作者归属，司法语言学，抄袭检测

Features

* Frequency of function words
* Character n-grams
* Discourse structure 论文结构

(4)Native-Language Identification 母语（方言）识别，司法语言学，教育应用

Features

* Word N-grams
* Syntactic patterns (POS, parse trees) POS: part-of-speech 词性 语法分析树
* Phonological features 语系特征

(5)Automatic fact-checking 事实审核，假新闻

Features

* N-grams
* Non-text metadata

Lexical semantics

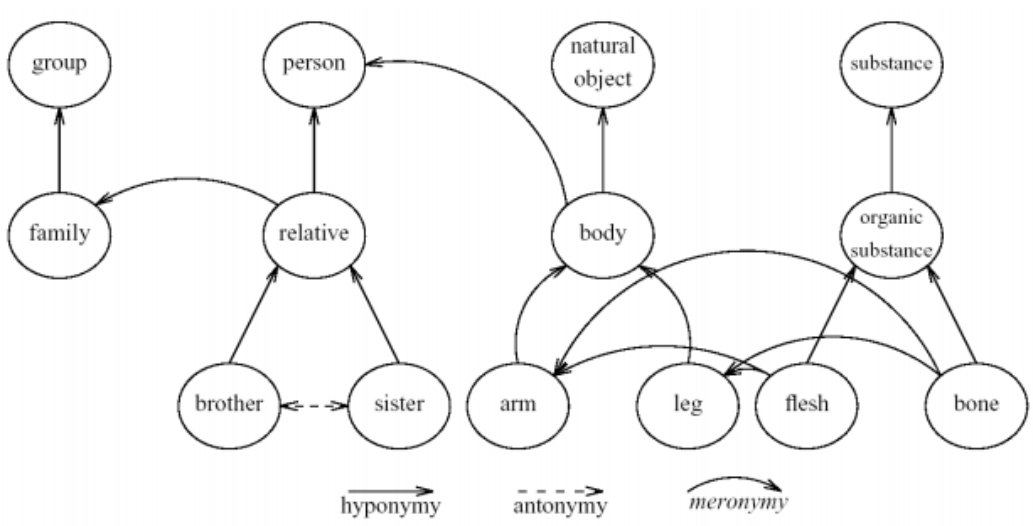
- Lexical relationships (*-nyms*)
- Structure of WordNet
- Similarity metrics
- Approaches to Word Sense Disambiguation

C6 Lexical semantics 字典式语义

1. Lexical relationships (-nyms)
   - Synonyms (same) and antonyms (opposite/complementary)
   - Hypernyms (generic), hyponyms (specific)

• Holonyms (whole) and meronyms (part)

2. Structure of WordNet



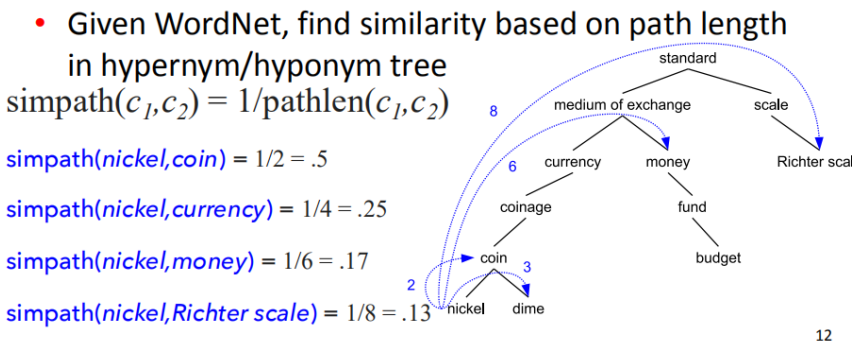
3. Similarity metrics

1. Word similarity with paths

Given WordNet, find similarity based on path length in hypernym/hyponym tree（上位词/下位词，集合与子集）

simpath(c1,c2) = 1/pathlen(c1,c2)

计算距离包含所有走过的词汇，包括自己



2. Beyond path length 不用距离用深度

$$simwup(c_1,c_2) = \frac{2*depth(LCS(c_1,c_2))}{depth(c_1) + depth(c_2)}$$

LCS：lowest common subsumer 最低层的共同类

simwup(nickel,money) = 2*2/(3+ 6) = .44

simwup(nickel,Richter scale) = 2*1/(3 + 6) = .22

nickel：第 6 层

money：第 3 层

medium of exchange：第 2 层

3. Information content 加入信息量

* P(c): prob. that word in corpus is instance of concept *c*

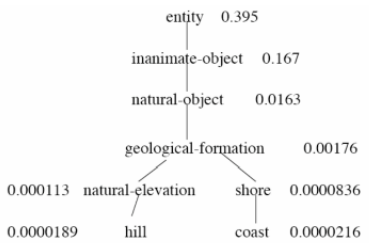
$$P(c) = \frac{\sum_{w \in words(c)} count(w)}{N}$$

* information content (IC)

$$IC(c) = -\log P(c)$$

* Lin distance

$$\text{simlin}(c_1, c_2) = \frac{2 * \text{IC}(\text{LCS}(c_1, c_2))}{\text{IC}(c_1) + \text{IC}(c_2)}$$



P(c)：c 在语料库中的出现率
IC：对 P 取-log
lin dis：对两个词和两个词的 LCS（最低公共分类）那个词分别取 IC 计算

4. Approaches to Word Sense Disambiguation

1). Supervised WSD
standard machine classifiers
Feature vectors
sense-tagged corpora
2).Less supervised WSD
Lesk
Yarowsky: Bootstrap method
Graph methods in WordNet

Distributional semantics

- Matrices for distributional semantics
- Association measures
  * Calculating (P)PMI from a co-occurrence matrix
- Count-based models
  * Basics of singular value decomposition (SVD)
- Predict-based models
  * Skip-gram, CBOW
- Cosine similarity

C7 Distributional semantics 分布式语义学

know a word by the company, Document co-occurrence, Local context reflects a word’s semantic class 通过搭配的其他词了解一个 word，根据共同出现，根据局部搭配归类，通过数学手段，变成向量，这就是分布式，相比于字典式（人工构建，有偏见和误差，昂贵，字典是静态的，

语言是动态的）。数学手段：count-based vs neural prediction-based

word2vec 的 vector 就是一种 distributional semantics。distributional semantics 就是把语义表示成一个向量 represent meaning as a vector，它的理论基础来自于 Harris 的分布假设：语义相似的词出现在相似的语境中（Semantically similar words occur in similar contexts）。具体的计算方法有多种，比如 LSA（Latent Semantic Analysis）、LDA（Latent Dirichlet Allocation）及各种神经网络模型（如 LSTM）等。

1.Matrices for distributional semantics

	…	the	country	hell	…
…					
state		1973	10	1	
fun		54	2	0	
heaven		55	1	3	
……					

Lists how often words appear with other words

2. Association measures

* Calculating (P)PMI from a co-occurrence matrix

$$PMI(x,y) = \log_2 \frac{p(x,y)}{p(x)p(y)}$$

Calculating PMI

	…	the	country	hell	…		Σ
…							
state		1973	10	1			12786
fun		54	2	0			633
heaven		55	1	3			627
…							
Σ		1047519	3617	780			15871304

$p(x,y) = count(x,y)/\Sigma$

$p(x) = \Sigma_x / \Sigma$

$p(y) = \Sigma_y / \Sigma$

x= state, y = country

$p(x,y) = 10/15871304 = 6.3 \times 10^{-7}$

$p(x) = 12786/15871304 = 8.0 \times 10^{-4}$

$p(y) = 3617/15871304 = 2.3 \times 10^{-4}$

$PMI(x,y) = \log_2(6.3 \times 10^{-7})/((8.0 \times 10^{-4})(2.3 \times 10^{-4}))$

$= 1.78$

14

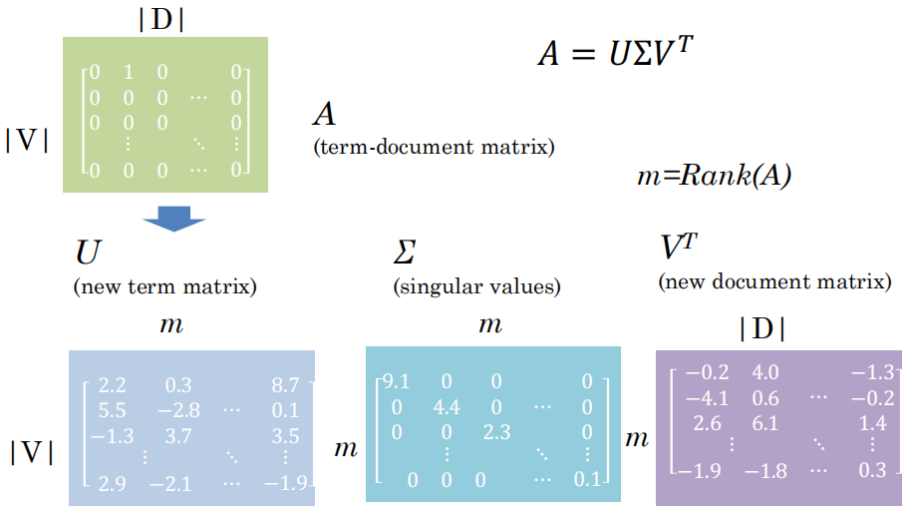
PMI matrix:

	…	the	country	hell	…
…					
state		1.22	1.78	0.63	
fun		0.37	3.79	-inf	
heaven		0.41	2.80	6.60	
……					

无法处理 0 值，受 common word 影响大，对 rare word 很 baised

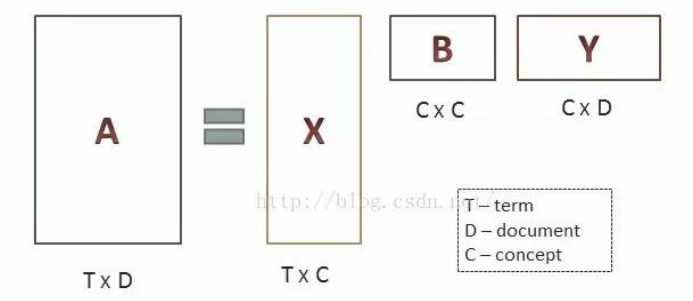
3. Count-based models

* Basics of singular value decomposition (SVD) 奇异值分解：将一个矩阵用其他几个矩阵的乘积来表示。



Dimensionality reduction 降维：Term-document matrices are very sparse，原始的词-文档矩阵过于稀疏，降维可以归类相关词

LSA，潜在语义分析



C 就是潜在语义的分类，也就是需要下降到的维度

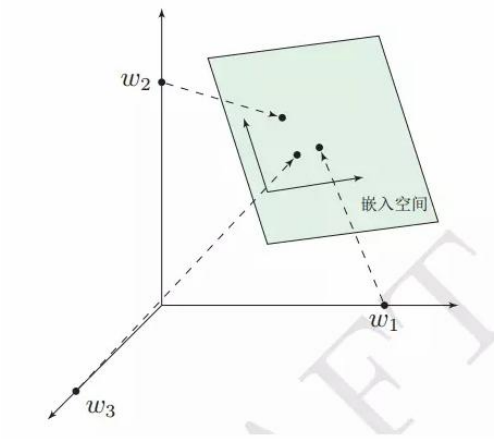
4. neural prediction-based models (word2vec 模型)

* Skip-gram, CBOW:

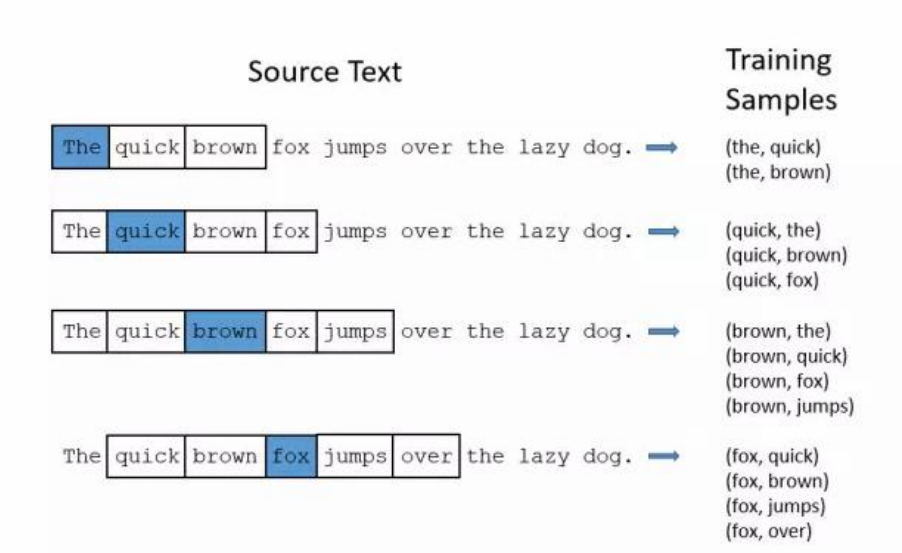
CBOW 模型的训练输入是某一个特征词的上下文相关的词对应的词向量，而输出就是

这特定的一个词的词向量。 Skip-Gram 模型和 CBOW 的思路是反着来的，即输入是特定的一个词的词向量，而输出是特定词对应的上下文词向量。CBOW 对小型数据库比较合适，而 Skip-Gram 在大型语料中表现更好。

word embedding：将高维词（稀疏）向量嵌入到一个低维空间。



1) skip-gram

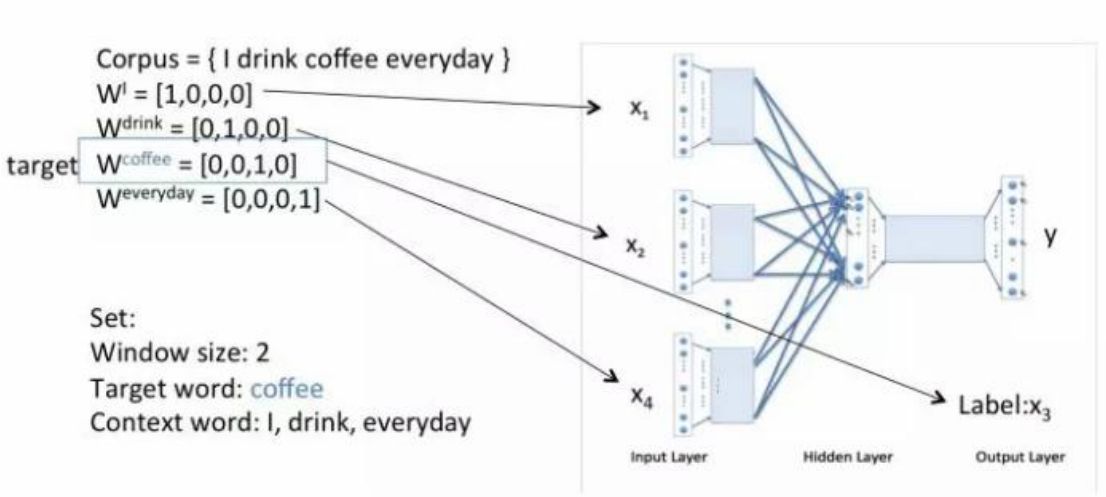


设定我们的窗口大小为 2（window_size=2），也就是说我们仅选输入词前后各两个词和输入词进行组合。下图中，蓝色代表 input word，方框内代表位于窗口内的单词。Training Samples（输入， 输出）

窗口大小又叫 Local context ：words within L positions, L=2 above

2)CBOW

Training Samples 对比上面是反过来的(quick, the) (brown, the)

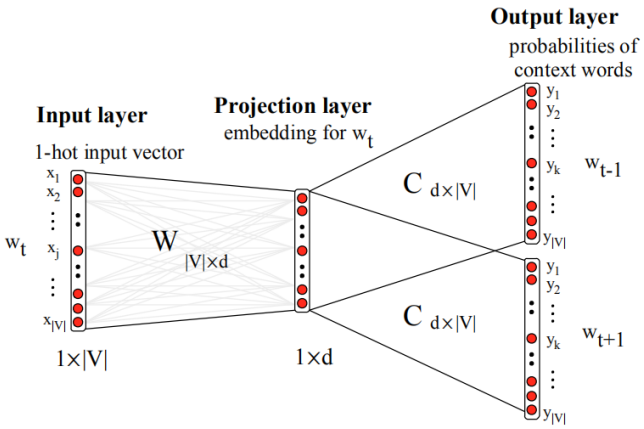


输入维度是|V|，V 是总 vocab。输出维度是要降到的维度 d，隐藏层又叫预测层

输入层：1*|V|（1-hot）

隐藏层 W：|V|*d

输出层 W：d*|V|



隐藏层计算：1*|V| x |V|*d = 1*d

输出层计算：1*d x d*|V| = 1*|V|

5. Cosine similarity

seeing whether or not words are roughly synonymous

Part of speech tagging

- English parts-of-speech
- Tagsets
  - **not:** fine-grained tags of any particular tagset
- Approaches

C8 Part of speech tagging

1. English POS & tagsets

NN noun 名词
VB verb 动词
JJ adjective 形容词
RB adverb 副词
DT determiner 冠词，限定词
CD cardinal number 基数，相对于序数
IN preposition 介词
PRP personal pronoun 人称代词
MD modal 情态动词
CC coordinating conjunction 连词
RP particle 小品词，at,in,on,off,up,out of,into,away
WH wh-pronoun WH 代词
TO to

2. part of speech ambiguity

Many word types belong to multiple classes

Time	flies	like	an	arrow
noun	verb	preposition	determiner	noun

Fruit	flies	like	a	banana
noun	noun	verb	determiner	noun

3.Approaches

• Rule-based taggers

starts with a list of possible tags for each word：From a lexical resource, or a corpus
问题：unambiguous contexts

• Statistical taggers

* Unigram tagger （as a baseline）

Assign most common tag to each word type
Requires a corpus of tagged words

* Classifier-based taggers

用分类器 a standard discriminative classifier：
features：

* Target word 目标词
* Lexical context around the word 周围词
* Already classified tags in sentence 句中已经分类过的 tag

问题：suffer from error propagation 错误的 prediction 影响后面的

* Hidden Markov Model (HMM) taggers

• Like sequential classifiers, use both previous tag and lexical evidence
• Unlike classifiers, treat previous tag(s) evidence and lexical evidence as independent from each other

Information extraction

- Named entity recognition
  - Models
  - Tagging formalisms (BIO)
- Relation extraction:
  - How to frame the problem using binary and multi-class classifiers
- Differences between supervised models and OpenIE.

C9 Information extraction

1. NER 就是从一段自然语言文本中找出相关实体，并标注出其位置以及类型

BIO: (B-begin，I-inside，O-outside) 配上 entity tags

American/**B-ORG** Airlines/**I-ORG** ,/**O** a/**O** unit/**O** of/**O** AMR/**B-ORG** Corp./**I-ORG** ,/**O** immediately/**O** matched/**O** the/**O** move/**O** , /**O** spokesman/**O** Tim/**B-PER** Wagner/**I-PER** said/**O** ./**O**

model：NER as sequence labeling, use features as input to classifier

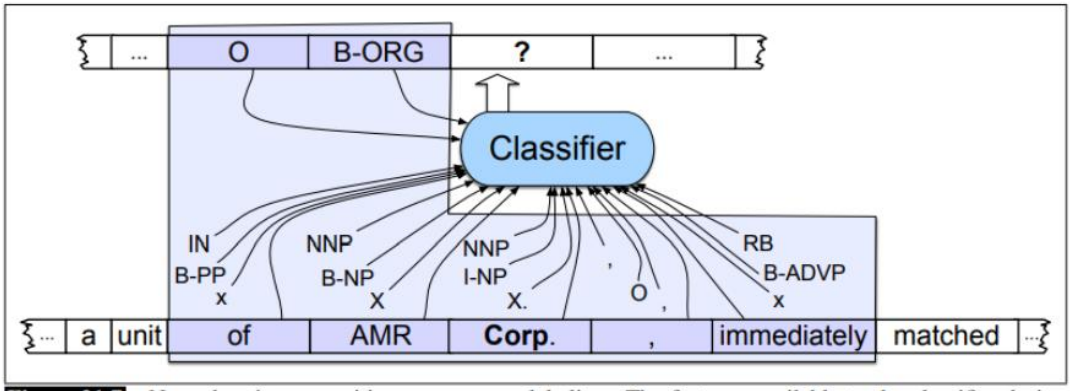


Figure 21.7 Named entity recognition as sequence labeling. The features available to the classifier during training and classification are those in the boxed area.

2.Relation extraction: have entities -> which pairs of entities are in relationships or not

• If we have access to a fixed relation database:

* Rule-based

人工创建 Lexico-syntactic patterns（词汇句法模式）然后一一参考关系

通过手工编写规则匹配文本，实现关系

- 手动编写词汇 句法 模式
- 编写 规则以识别文本中的 模式
  - 例子： founder -of （jobs ，apple ）
  - Text ：Jobs is the new CEO of Apple in 1976
  - rule ： is the new CEO of
  - New text: Mayer is the new CEO of Yahoo !
  - New entity pair: (Mayer ,Yahoo)

* Supervised

先有一个编好关系的语料库

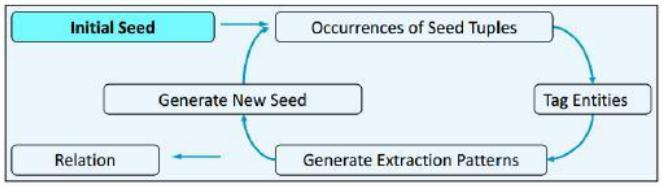
第一步先看两个 entities 是否有关联，binary 分类器，有关 positive，无关 negative

第二遍看是什么关系， multi-class 分类器

- **[ORG American Airlines],** a unit of **[ORG AMR Corp.]**, immediately matched the move, spokesman **[PER Tim Wagner]** said.
- First:
  - (American Airlines, AMR Corp.) -> positive
  - (Tim Wagner, American Airlines) -> positive
  - (Tim Wagner, AMR Corp.) -> negative
- Second:
  - (American Airlines, AMR Corp.) -> subsidiary
  - (Tim Wagner, American Airlines) -> employment

* Semi-supervised

- 为每种关系标注少量**种子实体**对，基于这些实体对在文本语料库中抽取相关**句子集合**，基于这些句子抽取表达关系的**模式**（pattern），以此循环**迭代**，这个过程也被称之为“滚雪球”（snowball）



图：基于bootstrapping的关系抽取流程[]

[Eugene Agichtein and Luis Gravano. Snowball: Extracting Relations from Large Plain-Text Collections. Fifth ACM Conference on Digital Libraries. San Antonio, TX, USA, 2000. Page 3.]

- 基于中文例子的迭代过程
  - Step1：给定关系"出生于"、种子实体对《周杰伦，台湾》和《林丹，福建》
  - Step2：抽取出句子集合：{"周杰伦，出生于台湾省新"，"周杰伦在台湾…"，"林丹小时候在福建学球"}
  - Step3：得到关系"出生于"的描述模式{"，出生于"，"在"，"小时候在"}
  - Step4：基于该模式，抽得句子"林俊杰，出生于新加坡的一个音乐世家"，从而得到实体对《林俊杰，新加坡》
- 代表性系统
  - DIPRE系统 (Brin, 1998)、Snowball系统 (Agichtein, 2000)、KnowItAll系统 (Etzioni et al. 2005)、TextRunner系统 (Banko et al. 2007)

* Distant supervision

远程监督，属于一种弱监督方式，它是利用外界的一个庞大的知识库来指导标注/扩充训练样本集。基本思想是，假设一个含有实体对的句子，如果该实体对能够和 DBpedia/Freebase 知识库中的实体对齐的话，那么该句子同时也包含了该实体对在 Freebase 中的关系，因此可以将该句子看成一个正样本。

Still rely on a fixed set of relations

• If no restrictions on relations:

* Unsupervised ("OpenIE") 如果没有一个 relation 数据库，用包含动词之类的子串来充当一个 relation

- Relations become substrings, usually containing a verb
- "United has a hub in Chicago, which is the headquarters of United Continental Holdings."
  - "has a hub in"(United, Chicago)
  - "is the headquarters of"(Chicago, United Continental Holdings)

问题：将 relation 对应于一个规范的（canonical）form

N-gram Language models

- Derivation
- Smoothing techniques
  - Add-*k*
  - Interpolation vs. backoff
  - Absolute discounting
  - **not:** Kneser-Ney, continuation counts etc.
- Perplexity

C10 N-gram language models

1. Derivation：n-gram language models: Markov models

(1) Probabilities: Joint to conditional 朴素贝叶斯，条件概率

对于一串 m 个 words，第 m 个 word 出现并不是孤立的，是基于 1 到 m-1 条件之下，

是 $P(w_m|w_1 \ \ldots w_{m-1})$ ，这些概率的乘积是整串 words 的概率

$$P(w_1,w_2,\ldots,w_m) = P(w_1)P(w_2|\,w_1)P(w_3|\,w_1,w_2)\ldots P(w_m|w_1\ \ldots w_{m-1})$$

(2) The Markov Assumption

假设一个 word 的出现只和前面 n-1 个 words 有关

When n = 1, a unigram model

$$P(w_1, w_2, .. w_m) = \prod_{i=1}^{m} P(w_i)$$

When n = 2, a bigram model

$$P(w_1, w_2, .. w_m) = \prod_{i=1}^{m} P(w_i|w_{i-1})$$

When n = 3, a trigram model

$$P(w_1, w_2, .. w_m) = \prod_{i=1}^{m} P(w_i|w_{i-2}w_{i-1})$$

(3) Maximum Likelihood estimation 最大似然估计

For unigram models,

$$P(w_i) = \frac{C(w_i)}{M}$$

For bigram models,

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}$$

For n-gram models generally,

$$P(w_i|w_{i-n+1}\ldots w_{i-1}) = \frac{C(w_{i-n+1}\ldots w_i)}{C(w_{i-n+1}\ldots w_{i-1})}$$

就是频数之间的商，Wi-n+1...Wi 这一串 word 出现的频率/Wi-n+1...Wi 的频率，分母少了 Wi

具体地，以Bi-gram为例，我们有这样一个由三句话组成的语料库：

<s> I am Sam <s>
<s> Sam I am <s>
<s> I do not like eggs and ham <s>

容易统计，“I”出现了3次，“I am”出现了2次，因此能计算概率：

$$p(am|I) = \frac{2}{3}$$

同理，还能计算出如下概率：

$$p(I|<s>) = 0.67 \quad p(Sam|am) = 0.5 \quad p(<s>|Sam) = 0.5$$
$$p(do|I) = 0.33 \quad p(not|do) = 1 \quad p(like|not) = 1$$

(4) Book-ending Sequences 书名号

Corpus:

* <s> = sentence start
* </s> = sentence end

<s> <s> *yes no no no no yes* </s>
<s> <s> *no no no yes yes yes no* </s>

对于 n-gram 模型，句首要摆 n-1 个<s>，句尾只用摆一个</s>

(5) problems

long distance effects 有逻辑联系的 word，距离太长，n 需要很大

- *The lecture/s that took place last week was/were on retrieval.*

probability very small 概率值太小，用 log

对于看不到的、没出现过的 word(s)，概率就是 0，用 smoothing

Words in new contexts

- By default, zero count for any *n*-gram we've never seen before, zero probability for the sentence
- Need to smooth the LM!

LM：language model

2. Smoothing (or discounting) 给这些没见过的情况一个概率值

(1) Laplacian (Add-one) smoothing 拉普拉斯平滑

原有的 n-gram 概率值，分子+1，分母加分子的种类数

For unigram models (**V=** the vocabulary),

$$P_{add1}(w_i) = \frac{C(w_i) + 1}{M + |\boldsymbol{V}|}$$

For bigram models,

$$P_{add1}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) + 1}{C(w_{i-1}) + |\boldsymbol{V}|}$$

优点：简单，便于实现

缺点：转移太多的概率

回到之前的莎士比亚问题上，V=30000个词，' the' 出现25545次，' the' 的概率为

$$P(w_i|w_{i-1} = the) = \frac{C(the\ w_i)+\mathbf{1}}{25,545+\mathbf{30,000}}$$

问题：

如果前面的 word 基数太大，会影响概率，这个概率不应该太大（太接近于前面出现过的部分概率）

解决办法：add-k ↓

(2) Add-k smoothing

Instead, add a fraction *k*

$$P_{addk}(w_i|w_{i-1}, w_{i-2}) = \frac{C(w_{i-2}, w_{i-1}, w_i) + k}{C(w_{i-2}, w_{i-1}) + k|\boldsymbol{V}|}$$

(3) Backoff and Interpolation 回退和插值

Interpolation 和 Backoff 的区别是

interpolation 给从高到低所有的 grams 都附一个权重，而 backoff 则基于"如若没有，退而求其次"的思想

backoff:

在高阶模型可靠时，尽可能的使用高阶模型。但是有时候高级模型的计数结果可能为 0，这时我们就转而使用低阶模型来避免稀疏数据的问题。

$$P_{backoff}(w_i|w_{i-n+1}\cdots w_{i-1}) = \begin{cases} P^*(w_i|w_{i-n+1}\cdots w_{i-1}) & ,if \quad C(w_{i-n+1}\cdots w_i) > 0 \\ \alpha(w_{i-n+1}\cdots w_{i-1}) \cdot P_{backoff}(w_i|w_{i-n+2}\cdots w_{i-1}) & ,otherwise \end{cases}$$

其中 α 和 P^* 是归一化因子，以保证 $\sum P_{backoff} = 1$。

interpolation:

在使用插值算法时，我们把不同阶别的 n-Gram 模型线形加权组合后再来使用。

$$P_{interp}(w_n|w_{n-2},w_{n-1}) = \lambda_1 P(w_n) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n|w_{n-2},w_{n-1})$$

其中，$0 \le \lambda_i \le 1$，$\sum_i \lambda_i = 1$。λ_i可以根据试验凭经验设定，也可以通过应用某些算法确定，例如EM算法。

(4) Absolute Discounting

Bigram count in training set	Bigram count in heldout set
0	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

除了计数为 0 和为 1 的 bigram 之外，held-out set 中的平均计数值，都大约相当于 training set 中的计数值减去 0.75。

Absolute discounting 会从每一个计数中减去一个（绝对的）数值 d。这样做的道理就在于，对于出现次数比较多的计数我们其实已经得到了一个相对比较好的估计，那么当我们从这个计数值中减去一个较小的数值 d 后应该影响不大。

$$P_{AbsDiscount}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{C(w_{i-1})} + \lambda(w_{i-1})P(w_i)$$

所以上述方程等号右侧第一项即表示经过 discounting 调整的概率值，而第二项则相对于一个带权重 λ 的 unigram 的插值项(结合了 Interpolation 的公式)。

这里的 d 就应该是 0.75。

3. Perplexity 困惑度，和 p 相反，越小越好

- Inverse probability of entire test set
  - Normalized by number of words (including </s>)
- The lower the better

$$PP(w_1, w_2, .. w_m) = \sqrt[m]{\frac{1}{P(w_1, w_2, .. w_m)}}$$

equivalently

$$PP(w_1, w_2, .. w_m) = 2^{-\frac{\log_2 P(w_1, w_2, .. w_m)}{m}}$$

- Unknown (OOV) words a problem (omit)

RNN models

- Basics of neural network structure
- How to frame LM/tagging as a word-by-word classification task
  - feed-forward classifiers vs recurrent neural networks
- Similarity with seq2seq as used in MT
- **not:** mathematical details of formulation

C11 RNN

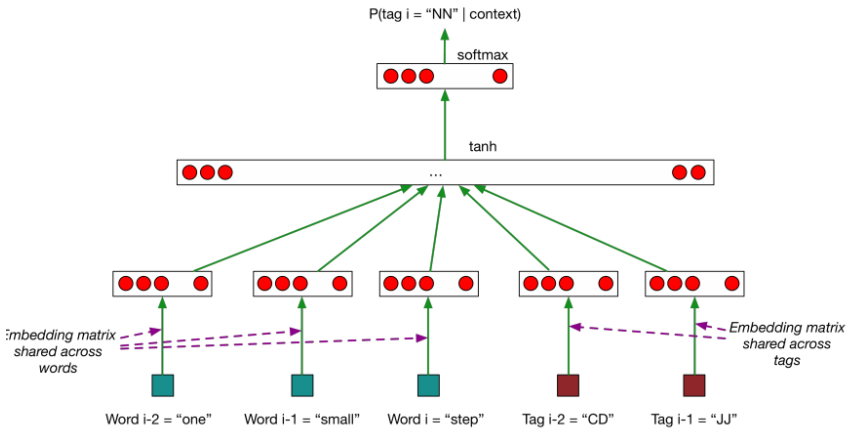
1. Basics of neural network structure

Based on vector embeddings and compositional functions over these vectors.

2. FF-NN vs RNN 前馈神经网络（最简单的神经网络） 循环神经网络

FF-NN：

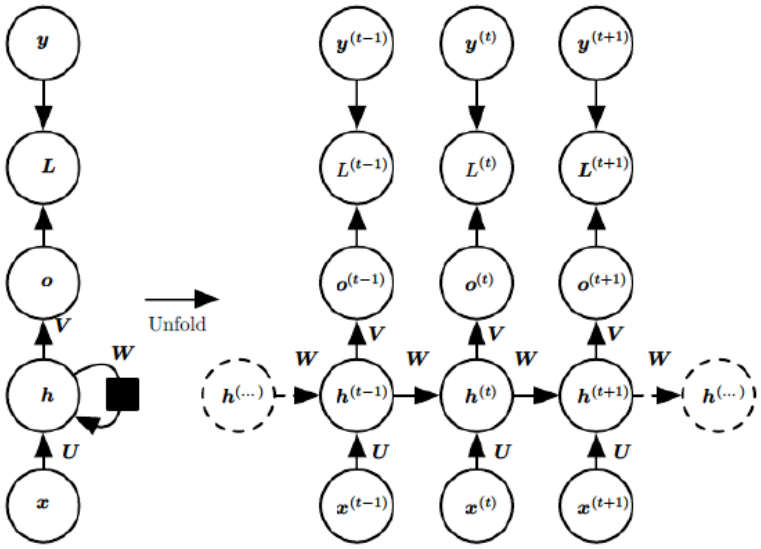
FF-NN for tagging



- MEMM tagger takes as input:
  - recent words w_{i-2}, w_{i-1}, w_i
  - recent tags t_{i-2}, t_{i-1}
- And outputs: current tag t_i
- Frame as neural network with
  - 5 inputs: 3 x word embeddings and 2 x tag embeddings
  - 1 output: vector of size |T|, using softmax

Softmax ensures probabilities >0 & sum to 1 归一化指数函数

RNN：



1）$x^{(t)}$代表在序列索引号t时训练样本的输入。同样的，$x^{(t-1)}$和$x^{(t+1)}$代表在序列索引号$t-1$和$t+1$时训练样本的输入。

2）$h^{(t)}$代表在序列索引号t时模型的隐藏状态。$h^{(t)}$由$x^{(t)}$和$h^{(t-1)}$共同决定。

3）$o^{(t)}$代表在序列索引号t时模型的输出。$o^{(t)}$只由模型当前的隐藏状态$h^{(t)}$决定。

4）$L^{(t)}$代表在序列索引号t时模型的损失函数。

5）$y^{(t)}$代表在序列索引号t时训练样本序列的真实输出。

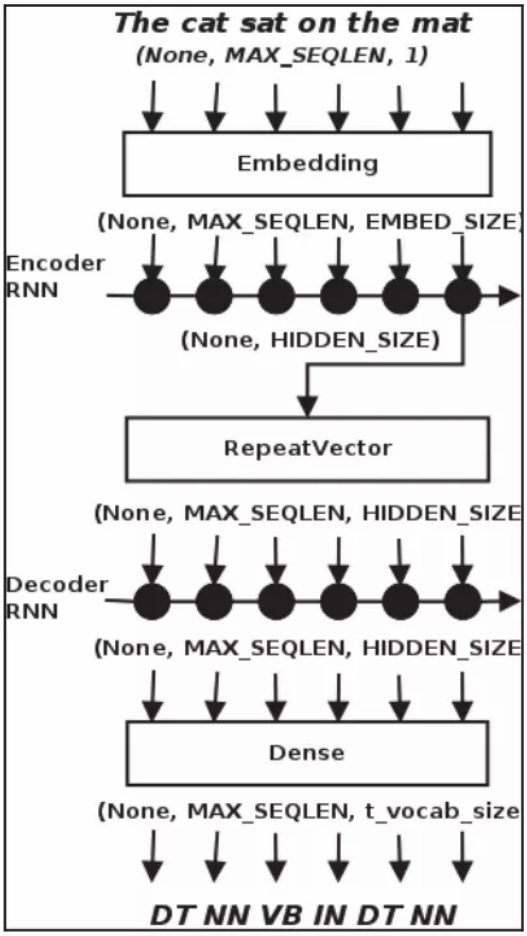
6）U, W, V这三个矩阵是我们的模型的线性关系参数，它在整个RNN网络中是共享的，这点和DNN很不相同。 也正因为是共享了，它体现了RNN的模型的"循环反馈"的思想。

$$h^{(t)} = \sigma(z^{(t)}) = \sigma(Ux^{(t)} + Wh^{(t-1)} + b)$$

$$o^{(t)} = Vh^{(t)} + c$$

$$\hat{y}^{(t)} = \sigma(o^{(t)})$$

tagging, e.g., using two RNNs, one for words and one for tags; and tags as outputs



第一个 RNN 的结果组成第二个 RNN 的输入序列

Sequence models for tagging

- Markov Models vs Hidden Markov Model
  - mathematical formulation of HMM, assumptions
- Training on fully observed data, e.g., tagging
- Viterbi algorithm

C12 Sequence models for tagging

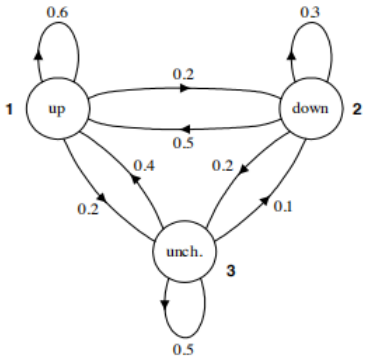
1. Markov Models vs Hidden Markov Model

   * mathematical formulation of HMM, assumptions

   （1）Markov Model

   马尔科夫链就是一串有概率的状态转移链

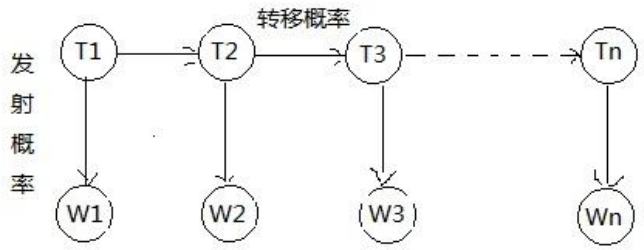
   假设当前的状态只与它的前一个状态相关



（2）Hidden Markov Model（在 tagging 中，t 是 tag，w 是 word，tag 被隐藏，word 被观察，每个 word 对于 tag 有一个观察概率）

隐含马尔科夫模型是马尔科夫模型的拓展，即任意时刻的状态是不可观测的。所以我们无法直接观测一个马尔科夫链，以及根据根据马尔科夫链计算后续的转移概率。

但是隐含马尔科夫模型在任意时刻会输出一个状态 ot，且这个状态只与 st 相关，这个被称为独立输出假设。



已知：词序列：

$$w_1 w_2 \ldots w_n$$

寻找词性序列：

$$t_1 t_2 \ldots t_n$$

使得条件概率最大：

$$P(t_1 t_2 \ldots t_n | w_1 w_2 \ldots w_n)$$

解：（忽略词序列概率，因为没用）

$$= \frac{P(t_1 t_2 \ldots t_n) P(w_1 w_2 \ldots w_n | t_1 t_2 \ldots t_n)}{P(w_1 w_2 \ldots w_n)}$$ 贝叶斯公式

$$\approx \underbrace{\prod_{i=1}^{n} P(w_i | t_i)}_{\text{独立输出假设}} \underbrace{\boxed{\prod_{i=1}^{n} P(t_i | t_{i-1})}}_{\text{发射概率，生成概率}}$$ 银马尔科夫假设 转移概率

这里 P(W1W2W3...)都是给定好了，是 1

发射概率：emission probabilities 一般给好了一个矩阵 Emission (observation) Matrix

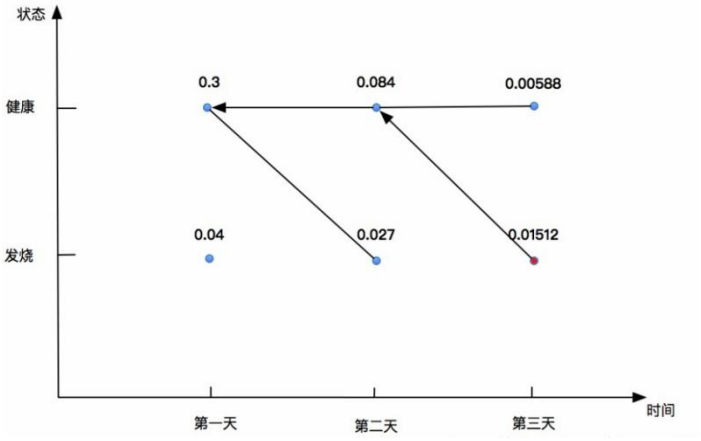
转移概率：transition probabilities 一般给好了一个矩阵 Transition Matrix

把两个概率相乘就行了，取最大值

2.Training on fully observed data

* $P(like|VB) = \frac{count(VB,like)}{count(VB)}$
* $P(NN|DT) = \frac{count(DT,NN)}{count(DT)}$

3. Viterbi algorithm：每一步都保存了前序的最优结果，根据前面的最优解算出当下所有解，提供一个最优解给下一步



bigram：O(T^2*N)　　T is the size of the tagset and N is the length of the sequence.要看前一个 t 的 tags，找最大 P，算出现在 t 的所有 tags 的 P

trigram：O(T^3*N)

tagging 中就是把 p 用两个矩阵对应值相乘算出来，然后把所有 p 的最优解相乘

Grammars and Languages

- Finite state automata and transducers
  - relationship to n-gram LMs, HMMs
  - Chomsky hierarchy
- Basic syntax of English
  - **not:** detailed nuances of grammar (see Q9 from 2017)
- The context-free grammar formalism
- Parsing
  - CYK algorithm

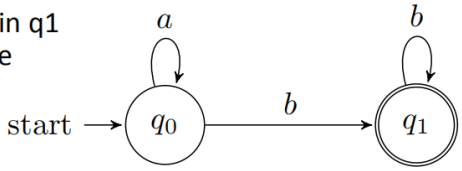
C13 Grammars and Languages

1. Finite state automata and transducers 有限状态机

其在任意时刻都处于有限状态集合中的某一状态。当其获得一个输入字符时，将从当前状态转换到另一个状态，或者仍然保持在当前状态。

Example FSA

- Input alphabet {a, b}
- States {q0, q1}
- Start, final states q0, {q1}
- Transition function {(q0,a) → q0, (q0, b) → q1, (q1,b) → q1}
- Note: seeing a in q1 results in failure
- Accepts *a*bb**



（1）N-gram LMs as WSFA HMMs?

（2）Chomsky hierarchy 乔姆斯基层次结构

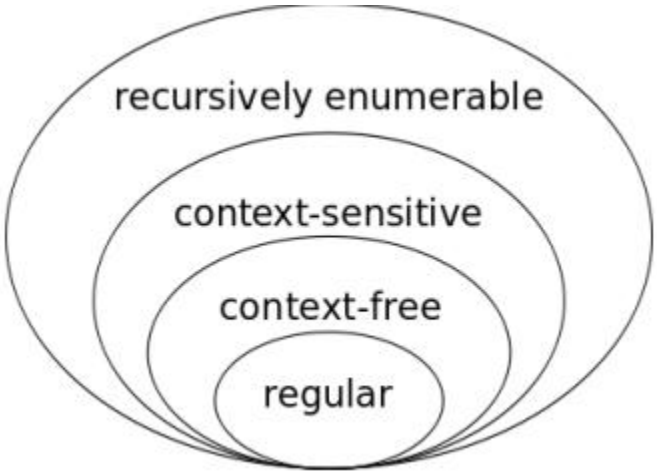


表 1　乔姆斯基层级

乔姆斯基层级	语法	语言	改写规则
0－型	不受限语法	递归可枚举语言	$\varphi_i \rightarrow \varphi_j$
1－型	上下文有关语法	上下文有关语言	$\varphi A_\psi \rightarrow \varphi\omega_\psi$
2－型	上下文无关语法	上下文无关语言	$A \rightarrow \varphi$
3－型	正则语法	正则语言	左线性规则：$A \rightarrow aB$；右线性规则：$A \rightarrow Ba$

低型语言不是高型的，默认情况下，每个高型的语言都是低型的

Grammar	Languages	Automaton	Production rules (constraints)
Type-0	Recursively enumerable	Turing machine	$\alpha \rightarrow \beta$ (no restrictions)
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type-2	Context-free	Non-deterministic pushdown automaton	$A \rightarrow \gamma$
Type-3	Regular	Finite state automaton	$A \rightarrow a$ and $A \rightarrow aB$

有限状态机是一种正则语法，hard to produce

2. Basic syntax of English

3. The context-free grammar formalism

上下文无关语法（Context-Free Grammer）

为了生成句子的语法树，我们可以定义如下的一套上下文无关语法。

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

举例来说，语法的一个语法子集可以表示为下图所示。当给定一个句子时，我们便可以按照从左到右的顺序来解析语法。例如，句子the man sleeps就可以表示为(S (NP (DT the) (NN man)) (VP sleeps))。

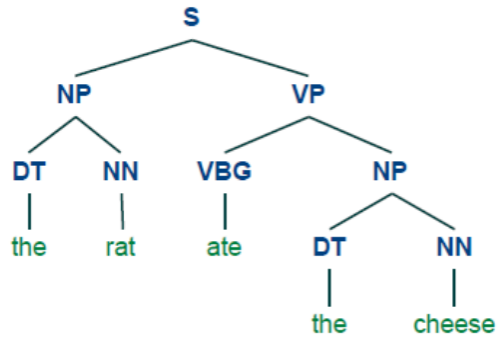
这种上下文无关的语法可以很容易的推导出一个句子的语法结构，但是缺点是推导出的结构可能存在二义性。例如下面两张图中的语法树都可以表示同一个句子。常见的二义性问题有：1）单词的不同词性，如can一般表示"可以"这个情态动词，有时表示罐子；2）介词短语的作用范围，如VP PP PP这样的结构，第二个介词短语可能形容VP，也可能形容第一个PP；3）连续的名字，如NN NN NN。

解析树

CFG trees

- Generation corresponds to a syntactic tree
- Non-terminals are internal nodes
- Terminals are leaves

```
(S (NP (DT the)
       (NN rat))
   (VP (VBG ate)
       (NP (DT the)
           (NN cheese))))
```

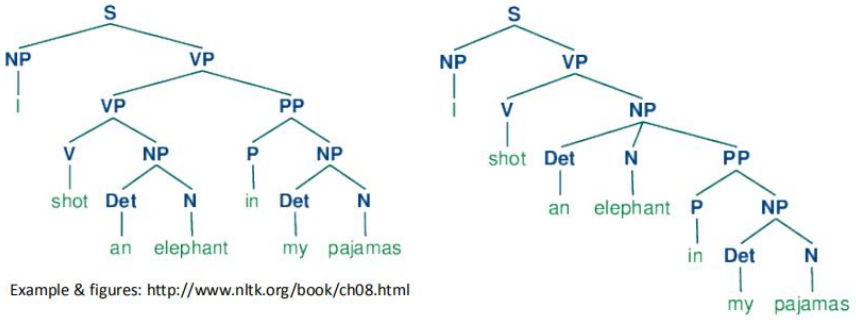


- Parsing is the *reverse* process

两义性

Parse Ambiguity

- Often more than one tree can describe a string
- *"While hunting in Africa, I shot an elephant in my pajamas. How he got into my pajamas, I don't know."*
  Animal Crackers (1930)

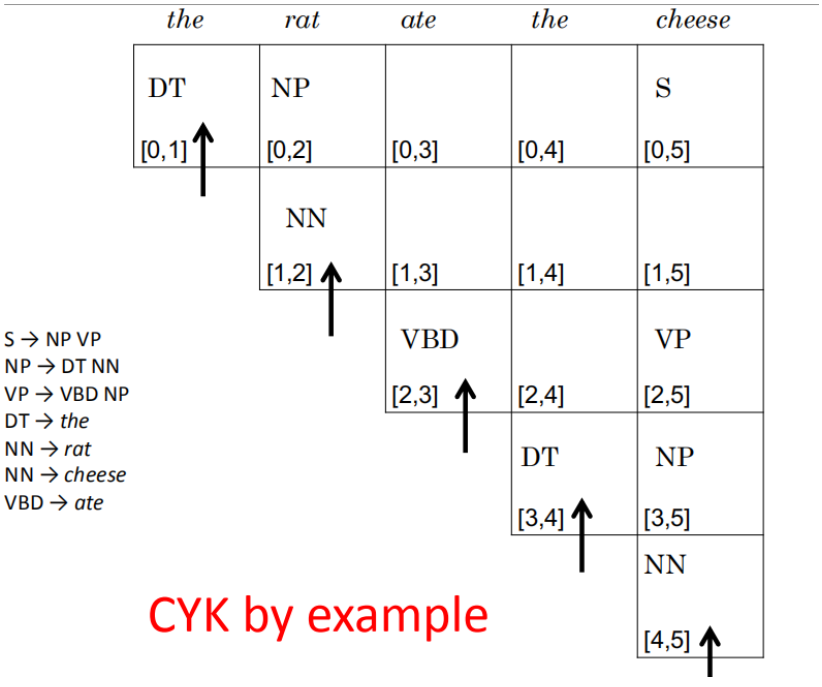


Example & figures: http://www.nltk.org/book/ch08.html

4.Parsing 解析

* CYK algorithm

- Two general strategies
  - Bottom-up
    - Start with words, work up towards S
    - CYK parsing
  - Top-down
    - Start with S, work down towards words
    - Earley parsing (not covered)



CYK by example

• Sentence (S)
• Noun phrase (NP) 名词短语
• Verb phrase (VP) 动词短语
• Prepositional phrase (PP) 介词短语
• Adjective phrase (AdjP) 形容词短语
• Adverbial phrase (AdvP) 副词短语
• Subordinate clause (SBAR) 从句

Prob. CFGs

- Ambiguity in grammars
- Probabilistic context free grammars: rules, generative process, probability of a tree
- PCYK algorithm for parsing
- Comparing to Viterbi and other 'decoding' methods

C14 Prob. CFGs

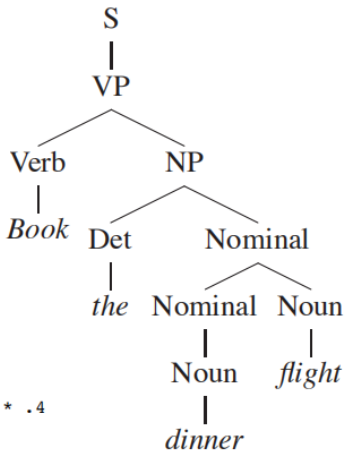
1. Ambiguity in grammars
2. Probabilistic context free grammars: rules, generative process, probability of a tree

A Probabilistic grammar

Grammar		Lexicon
S → *NP VP*	[.80]	*Det* → *that* [.10] | *a* [.30] | *the* [.60]
S → *Aux NP VP*	[.15]	*Noun* → *book* [.10] | flights [.30]
S → *VP*	[.05]	| *meal* [.015] | *money* [.05]
NP → *Pronoun*	[.35]	| *flight* [.40] | *dinner* [.10]
NP → *Proper-Noun*	[.30]	*Verb* → *book* [.30] | *include* [.30]
NP → *Det Nominal*	[.20]	| *prefer* [.40]
NP → *Nominal*	[.15]	*Pronoun* → *I* [.40] | *she* [.05]
Nominal → *Noun*	[.75]	| *me* [.15] | *you* [.40]
Nominal → *Nominal Noun*	[.20]	*Proper-Noun* → *Houston* [.60]
Nominal → *Nominal PP*	[.05]	| *NWA* [.40]
VP → *Verb*	[.35]	*Aux* → *does* [.60] | *can* [40]
VP → *Verb NP*	[.20]	*Preposition* → *from* [.30] | *to* [.30]
VP → *Verb NP PP*	[.10]	| *on* [.20] | *near* [.15]
VP → *Verb PP*	[.15]	| *through* [.05]
VP → *Verb NP NP*	[.05]	
VP → *VP PP*	[.15]	
PP → *Preposition NP*	[1.0]	

How likely is a tree?

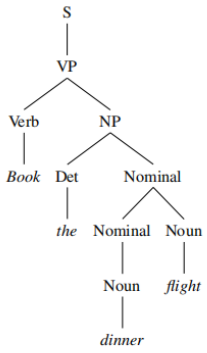
- Given a tree, we can compute its probability
  - Decomposes into probability of each production
- E.g., for tree on right,
  - P(tree) =
    - P(S → VP) ×
    - P(VP → Verb NP) ×
    - P(Verb → *Book*) ×
    - P(NP → Det Nominal) ×
    - P(Det → *the*) ×
    - P(Nominal → Nominal Noun) ×
    - P(Nominal → Noun) ×
    - P(Noun → *dinner*) ×
    - P(Noun → *flight*) = 2.16×10^{-6}

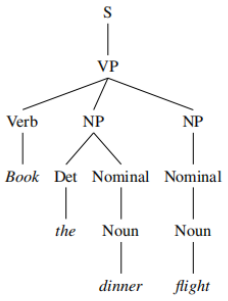



```
I.e., .05 * .2 * .3 * .2 * .6 * .2 * .75 * .1 * .4
```

Resolving parse ambiguity

- Can select between different trees based on P(T)





Source: JM3
Fig 12.2

Mistakes in textbook

```
.05*.05*.20*.15*.75*.75*.30*.60*.10*.40
```

- P = 2.16×10^{-6} P = 3.04×10^{-7}

根据分解概率相乘结果比大小

3.PCYK algorithm for parsing

Illustration

we	eat	sushi	with	chopsticks
NP 1/4		S 1/64		S 1/1024
	V 1	VP 1/16		VP 1/256
		NP 1/8		NP 1/128
			IN 1	PP 1/8
				NP 1/8

S	→ NP VP	1
NP	→ NP PP	½
	→ we	¼
	→ sushi	1/8
	→ chopsticks	1/8
PP	→ IN NP	1
IN	→ with	1
VP	→ V NP	½
	→ VP PP	¼
	→ MD V	¼
V	→ eat	1

Fixed mistake after the lecture

Example & grammar from E18 Chapter 10

25

根据左下角的概率，一路寻找最大的概率值乘积

4.Comparing to Viterbi and other ‘decoding’ methods

Dependency grammar

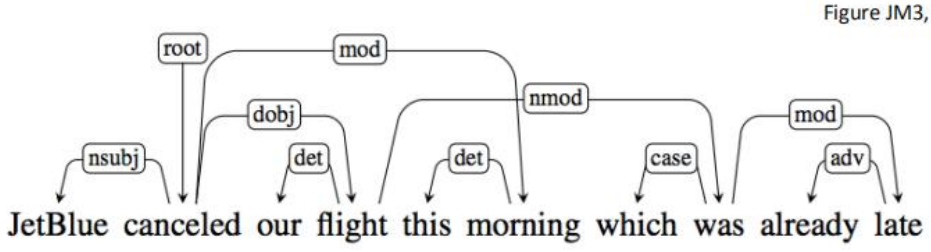
- Notion of dependency between words
- Dependency grammars and dependency parse trees
  - Projectivity vs non-projectivity
  - Transition based parsing algorithm
- **not**: graph based parsing
- **not:** detailed dependency edge inventory

C14 Dependency grammar

1. dependency 词之间的 links
2. Projectivity

A tree is projective if, for all arcs from head to dependent
从头到尾一直都有弧形，不间断，英文基本都是

Figure JM3, Ch 13

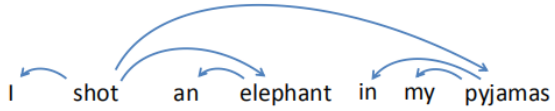


3.Transition based parsing algorithm

Example

Buffer	Stack	Action
I shot an elephant in my pyjamas		Shift
shot an elephant in my pyjamas	I	Shift
an elephant in my pyjamas	I, shot	Arc-left
an elephant in my pyjamas	shot	Shift
elephant in my pyjamas	shot, an	Shift
in my pyjamas	shot, an, elephant	Arc-left
in my pyjamas	shot, elephant	Arc-right
in my pyjamas	shot	Shift
...	...	...
	shot	<done>

Generated parse:



只会形成 projective tree，只用画弧线，不用标种类
依存语法有一个中心词的概念。但是 CFG 没有。
但是在现代的语言理论以及所有的现代统计句法分析中，确实有一个手工定义的“中心词规则”。
名词词组（NP）的中心词是名词/数词/形容词
动词词组（VP）的中心词是动词…
“中心词规则”可以帮助我们从上下文无关文法句法（CFG）中抽取依存句法

Question Answering

- Major approaches
- Information Retrieval QA pipeline
  - Passage retrieval
  - Answer extraction

C15 Question Answering

1. Major approaches
   1) Retrieval, find answer question via string/text passage in a document (collection)
      （1）Question Processing
         Find key parts of question for IR engine
         Predict expected answer type
      （2）Doc and passage retrieval
         Find document, and passage within document：
         Find top n documents

Next find passages (paragraphs or sentences) in these documents

Re-rank IR outputs to find best passage

（3）Answer extraction

Extract short answer string

1.5) Deep learning

2) Natural Language Interface to Database ("NLIDB"), automatically construct a query, and answer question relative to fixed KB (knowledge bases)

Discourse

- Motivation for modelling larger documents
- Discourse segmentation with TextTiling
- Notion of RST parsing of documents (at high level; **not** edge label inventory)
- Anaphora resolution, and the centering algorithm

C16 Discourse

a coherent, structured group of sentences 语篇

1. Motivation for modelling larger documents

- Summarization
- Sentiment analysis 观点分析
- Argumentation 推论
- Authorship attribution 作者
- Essay scoring 打分
- Anaphor resolution 复指，上下文分析

2. Discourse segmentation with TextTiling 分段

TextTiling method:

1>先把文档分成块(tile),每块 k 个句子。

2>计算相邻块之间的相似度,用标准的文档相似度计算公式。

3>用结果绘图。非常相似处会出现波峰,很不相似的地方出现波谷。选择波谷处作为分界线,把块组成段,这些段很可能是有关同一个 Subtopic 的。

* Create two BOW vectors consisting of words from *k* sentences on either side of gap
* Use cosine to get a similarity score (*sim*) for two vectors
* For gap *i*, calculate a depth score, insert boundaries when depth is greater than some threshold

$$depth(\text{gap}_i) = (sim_{i-1} - sim_i) + (sim_{i+1} - sim_i)$$

做词袋，做向量，每 k 个做相似度，再算深度

Text Tiling example (k=2)

He walked 15 minutes to the tram stop.

Dot product: 3

Then he waited for another 20 minutes, but the tram didn't come.

Dot product: 2

The tram drivers were on strike that morning.

Dot product: 0

So he walked home and got his bike out of the garage.

Dot product: 2

He started riding but quickly discovered he had a flat tire

Dot product: 3

He walked his bike back home.

Dot product: 1

He looked around but his wife had cleaned the garage and he couldn't find the bike pump.

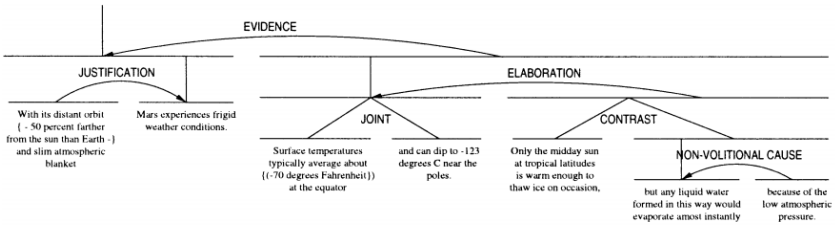
dot product 是点积，gap1 是第一句和二三句之间，为（1,1）·（1,2）=3
gap2 是一二和三四，（2,2，0,0）·（0，1,1,1）=2
depth(gap2) = 3-2+0-2=-1(错，还有分母）

3.Notion of RST parsing of documents

An RST Tree

RST= Rhetorical Structure Theory

23 relations, most with a *nucleus* and *satellite*



4.Anaphora resolution, and the centering algorithm

- Anaphors have a **antecedent** in the discourse, often but not always a noun phrase

Yesterday, Ted was late for work. It all started when his car wouldn't start.

The Centering Algorithm？

Machine translation

- Motivation
- Word alignment with IBM model 1
  - **not:** mathematical derivation of alignment posterior
- Phrase based model; stack decoding algorithm
- Sequence to sequence model
  - **not:** mathematical formulation
- Evaluation