COMP90042 Web Search and Text Analysis

Lecture 1. Preprocessing

Token: 句子中的每一个词，可重复
Type: the distinct word, 不同的词
Corpus: a collection of document
Tokenisation: the act of transforming a (long) document into a set of meaningful substrings, so that we can compare with other (long) documents. (needs **language identification** before we decide how to tokenise)
Stemming and **Lemmatisation**: Both transform a token into a normalised form. Both operate by applying a series of rewrite operations to remove or replace (parts of) affixes
Lemmatisation needs a lexicon (a list of valid words in the language). The goal is to turn the input token into an element of this list using the rewrite rules. If the token can't be transformed, then the token is left alone. (For example, "lemming" wouldn't be transformed into "lemm")
Stemming simply applies the rewrite rules, even if the output is a garbage token (like lemm). e.g. "this"在lemmatisation后还是"this"，但stemming后是"thi"因为按规则strip掉s后缀
Inflectional Morphology: 曲折形态学 tokens are altered to conform to certain grammatical constraints [noun变plural; verbs: number of the subject (人称), aspect of the action (+ing), the tense of the action (+ed); adjectives: comparatives (+er) and superlatives (+est)] Don't really alter the meaning of the term. <u>Both stemming and lemmatisation</u> attempt to remove this kind of morphology
Derivational Morphology: 衍生形态学 transform terms of one class into a different class. [suffixes: write -> writer, prefixes: health -> unhealthy] Produce terms that differ (perhaps subtly) in meaning, and the two separate forms are usually both listed in the lexicon. lemmatisation doesn't usually remove it, but stemming usually does.
Other Word Normalisation: fixing spelling errors, normalize spelling variations: normalize -> normalise, expanding abbrs
Stop Words: a list of words to be removed from the document (常用于bag-of-word (BOW) representations 即忽略文章句法单词顺序仅看作若干词汇的集合，not appropriate when sequence is important) 选择: all closed-class or function words (the, a, of, for), any high frequency words

Lecture 2. Information Retrieval: the Vector Space Model

qrels: Representation of relevance judgments. qrels are judgments made by humans as to whether a document is relevant to an information need (i.e., topic).
Term-document matrix (TDM): each doc is a bag of words (discarding word order, just recording counts), rows represent docs, column represent terms (typically word types excluding stop words, and often stemmed), matrix cells might be binary indicators, counts <u>or other score</u>

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

also as points in a multi-dimensional term space

Cosine Distance: $\cos(\mathbf{a},\mathbf{b}) = \frac{\mathbf{a}.\mathbf{b}}{|\mathbf{a}| \times |\mathbf{b}|}$ p= $\mathbf{a}.\mathbf{b} = \sum_{i=1}^{|T|} a_i b_i$ $|\mathbf{a}| = \sqrt{\sum_{i=1}^{|T|} a_i^2}$

Vector Space Model: 所有|a|和a可以预先计算好，query时只需要计算b，vectors are sparse with many entries equal to zero, only need to consider non-zero entries in both a and b, use <u>inverted index</u> to exploit sparsity and allow efficient storage and querying

TF-IDF: Scores in TDM typically combine tf and idf. TF: term frequency, IDF: log(doc总数/包含term的doc数)

Raw TF

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

IDF

* "two" occurs in 1 doc (doc1)
  $idf_{two} = \log_2(3/1) = 1.58$
* "tea" occurs in 2 docs (doc1, doc2)
  $idf_{tea} = \log_2(3/2) = 0.58$

Raw TF and IDF

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

idf	1.58	0.58	0.58	0.58

->

	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

(即a.b中的a)

Normalise TF*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

(即每一项除以|a|

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

（这里normalise时每一项都除以了|a|之后的b就不用除以|a|了）

Query Processing in VSM: Treat the query as a short document Query: tea me

Raw TF:

	two	tea	me	you
query	0	1	1	0

TF/|q|

	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 越大越好

Inverted Index (Term-wise Processing): terms as rows, values as lists of (docID, weight) pairs, aka posting list

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

weights are normalised TF*IDF values (也可是count)

Querying an inverted index: query: tea me 似乎不考虑query的tf*idf

Assuming normalised TF*IDF weights:

```
Set accumulator a_d ← 0 for all documents d
for all terms t in query do
    Load postings list for t
    for all postings ⟨d, w_{t,d}⟩ in list do
        a_d ← 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

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]

Compare using a term–document matrix vs. an inverted index for resolving a ranked query efficiently: For the <u>term–document matrix</u>, we will need to read the TDM value of every document for each query term. For the <u>inverted index</u>, we only need to examine each entry in each postings list once; we incrementing the accumulator weights rather than comparing documents identifiers.

Recall the Okapi BM25 term weighting formula:

$$w_t = \log \frac{N - f_t + 0.5}{f_t + 0.5} \times \frac{(k_1+1) f_{d,t}}{k_1((1-b) + b\frac{L_d}{L_{\text{avg}}}) + f_{d,t}} \times \frac{(k_3+1) f_{q,t}}{k_3 + f_{q,t}}$$

Ld is the length of document d and Lavg is the average document length in the collection.

(a) What are its parameters, and what do they signify?

• k1 controls the weight of the term frequencies: high values mean that the weight for a term in the query is mostly linear in $f_{d,t}$; low values mean that the $f_{d,t}$ are mostly ignored, and only their presence/absence matters.

• k2 isn't in this model. In fact, in practice, k2 = k3 = 0 and k1 = 0.9, b = 0.4 gives good performance.

• k3 controls the term frequency within the query. If terms are repeated in the query, they contribute more to the document ranking. (often set to 0, query terms are only binary)

• b controls the penalty for long documents: when b is close to 0, document length is mostly excluded; when b is large, short documents are strongly preferred to long documents.

(b) How do the components relate to TF (term frequency) and inverse document frequency (IDF)?

• As discussed above, k1 controls the importance of the term frequencies (TF).

• Term weights (IDF) in the BM25 model can be negative, if the term appears in greater than half of the documents in the collection. This is unlikely to be a problem in many non-trivial collections, as such a common word is a stop–word, and we would simply exclude it from the collection.

Lecture 3. Index Compression and Efficient Query Processing

Compression Principles: For example, H(abracadabra) = 2.040373 bits with n = 11, fa = 5, fb = 2, fc = 1, fd = 1, fr = 2. Intuition: Spend less bits on items that occur often.

$$H(T) = -\sum_{s \in \Sigma} \frac{f_s}{n} \log_2 \frac{f_s}{n}$$

Posting list Compression: Minimize storage costs, Fast sequential access, Support GEQ(x)

the	ids:	25	26	29	…	12345	12347
	gaps:	25	1	3	…	1	2

Variable Byte Compression:

Compress number 512312 or 1111101000100111000 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 $31 + 128 = 159$

Encoding

```
function ENCODE(x)
    while x >= 128 do
        WRITE(x mod 128)
        x = x ÷ 128
    end while
    WRITE(x + 128)
end function
```

Decoding

```
function DECODE(bytes)
    x = 0, s = 0
    y = READBYTE(bytes)
    while y < 128 do
        x = x ^ (y << s)
        s = s + 7
        y = READBYTE(bytes)
    end while
    x = x ^ ((y − 128) << s)
    return x
end function
```

List Compression - Fast Searching:

1. Compress postings list in blocks of 128 integers at a time 分组
2. For each block store an uncompressed sample value representing the largest value in the block 每组存一个未压缩的本组最大值
3. Use sample values to efficiently seek to any position in the postings list without decompressing everything

Operation GEQ(x) (Greater or Equal x):

比如[7 8 12] [15 18 23] [24 29 57]存未压缩的[12][23][57]二分克制GEQ(17)在第二组

1. Binary search over uncompressed sample values to find destination block
2. Decompress destination block to determine final offset in postings list

Top-k Retrieval:

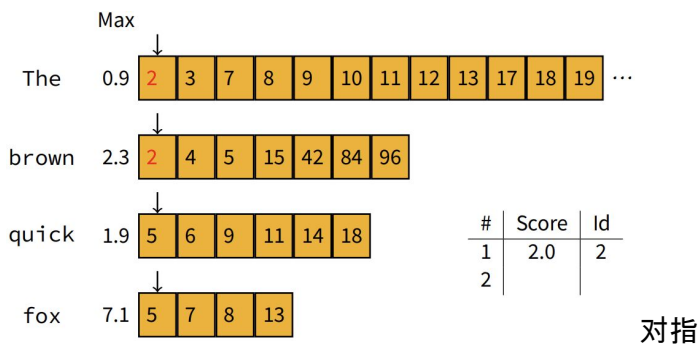
1. Avoid scoring documents we know will not appear in the top-k result list.
2. For a given similarity metric (for example BM25), prestore some information for each term to avoid scoring.
3. Incorporate with block based compression schemes for efficient query processing.
4. Utilise the GEQ(x) operation to avoid decompression of large parts of postings lists.

The WAND Algorithm:

1. Keep track of the top-k highest scored documents.
2. For each unique term in the collection store the <u>maximum contribution</u> (the largest score any document in the collection can have for the query Q only consisting of q, used to overestimate 对于每个term用BM25计算当query只有这个term时，对每一个doc的score取最大的那个) it can have to any document score in the collection.
3. Skip over documents that can not enter the top-k highest results.

Query Q: The quick brown fox with $k = 2$

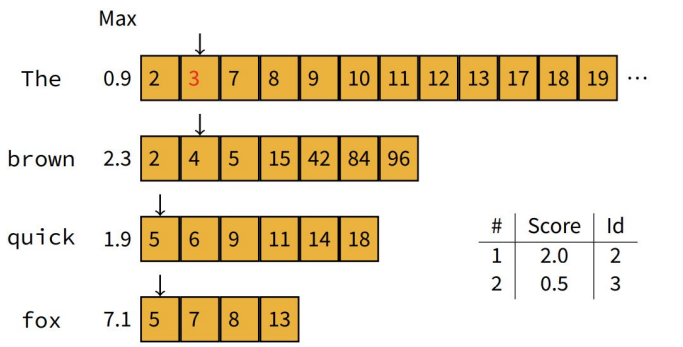
(1) Reorder based on current id and score smallest.



#	Score	Id
1	2.0	2
2		

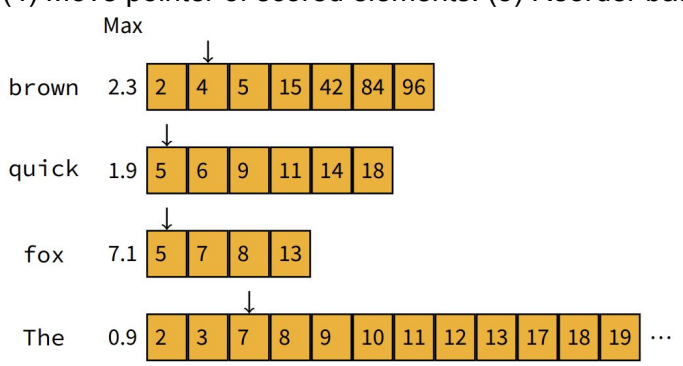
对指向的doc 2计算对Q的实际BM25值

(2) Move pointer of scored elements. (3) Reorder based on current id and score smallest.

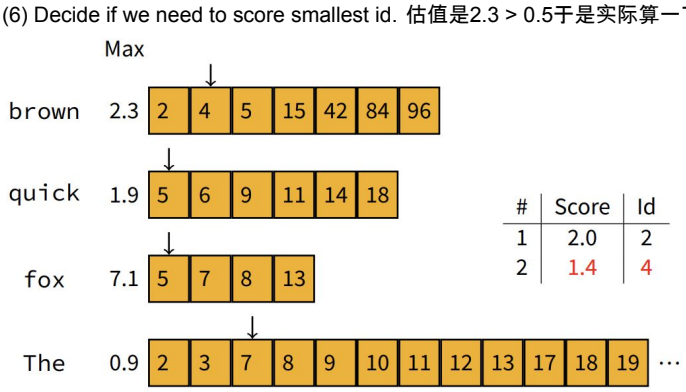


#	Score	Id
1	2.0	2
2	0.5	3

(4) Move pointer of scored elements. (5) Reorder based on current id and score smallest.



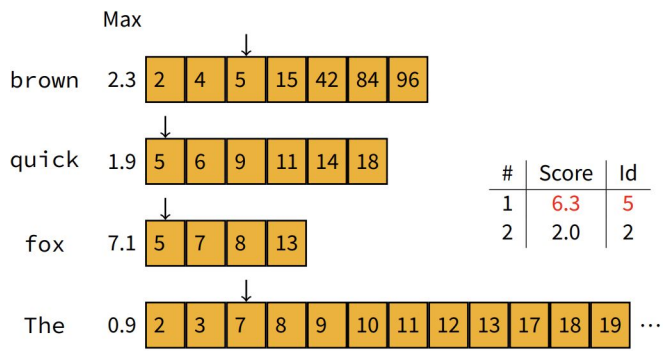
(6) Decide if we need to score smallest id. 估值是2.3 > 0.5于是实际算一下4的值 (7) Replace 3 with 4 on the heap.



#	Score	Id
1	2.0	2
2	1.4	4

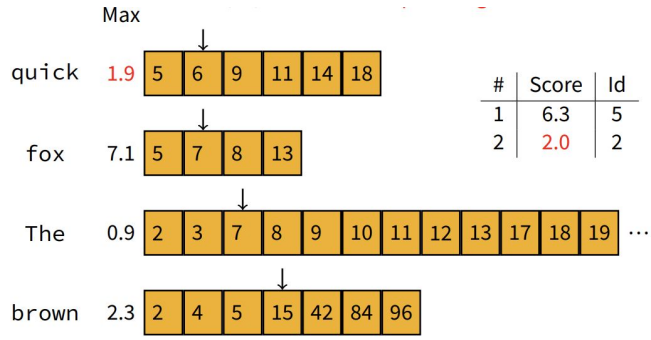
doc 4的实际值是1.4

(8) Sort by current id. Evaluate 5. Add to Heap (即右边那个rank).



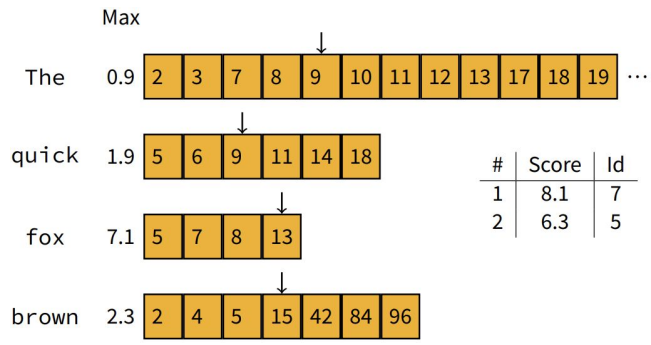
#	Score	Id
1	6.3	5
2	2.0	2

(9) Move pointers and sort. (10) Use max to skip scoring smallest. 6最多是1.9所以skip



#	Score	Id
1	6.3	5
2	2.0	2

7, 8, 9全部skip 对于9 -> 0.9+1.9 = 2.8 < 6.3



#	Score	Id
1	8.1	7
2	6.3	5

已知只有The和quick的doc都要skip，9之后还未解压，假定fox指向的doc13包含The和quick，估计值为0.9+1.9+7.1 > 6.3，所以要计算doc13 Fast forward smaller ids to 13 (GEQ) and sort.

Given Q, k and the postings lists $L[0 \ldots |Q|-1]$ with:

$L[i].max \triangleq$ the **maximum contribution** of the list
$L[i].cur \triangleq$ the current element of the list

```
function WAND(Q,k,L[0 ... |Q| − 1])
    TopDocs = ∅                                   ▷ Min Heap of size k
    Threshold Θ = 0                               ▷ Smallest score in TopDocs
    while Not all lists are processed do
        Sort L based on L[i].cur
        Select pivot list p such that Σ_0^{p−1} L[i].max >= Θ
        Forward all lists L[0 ... |p| − 1] to d_p = L[p].cur
        Compute S^{BM25}_{Q,d_p} and insert into TopDocs if score > Θ
        Θ = min(TopDocs) or 0 if |TopDocs| < k.
    end while
    Return TopDocs
end function
```

Lecture 4. Query Completion / Expansion

Goals of Query Completion: Assist users to formulate search requests, Help with spelling query terms, Guide user towards what a good query might be, Cache results and reduce server load

High Level Algorithm of Query Completion: 1. Retrieve set of candidates “matching” P from set S of possible target queries. 2. Rank candidates by frequency. 3. Possibly re-rank highest ranked candidates with more complex ranking measure (e.g. personalized) 4. Return the top-K highest ranking candidates as suggestions.

Where does the set S of possible completions 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)

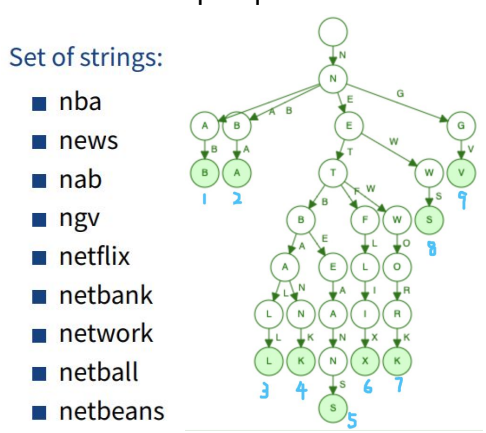
Completion Types (‘Modes’): 1. Prefix match. 2. Substring match. 3. Multi-term prefix match (多词前缀FI wor). 4. Relaxed match (可修复spelling mistakes).

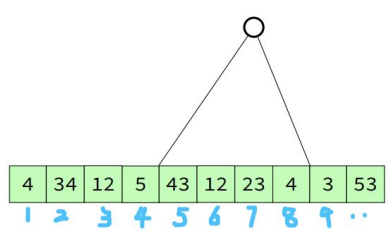
Prefix match - Trie+RMQ based Index:

1. Preprocess data by sorting query log in lexicographical order and counting frequency of unique queries: 统计每个query出现多少次

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	

2. Insert all unique queries and their frequencies into a **<u>trie</u>** (also called a <u>prefix tree</u>).





在叶子节点给每个单词一个index，另有一个array记录每个单词的freq，对于每个query查找所有子树的freq，返回前k个（RMQ）The subtree prefixed by P corresponds to a continuous range.

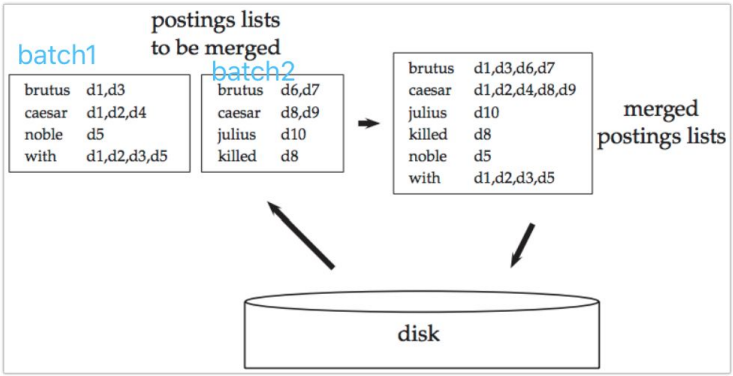
Range Maximum Queries (RMQ): 求区间最大k个值 Array A is size n. There are O(n^2) different ranges A[i, j]. For each range precompute the position of the maximum. Uses O(n^2) space. 预处理所有区间的最大值，查区间最大值O(1)

Extension to K largest numbers: 1. Find position p of largest element on A[i, j]. 2. Recurse to A[i, p − 1] and A[p + 1, j]. 3. Keep going until you have the K largest elements. 4. Runtime O(Klog K).

RMQ Index- Reduce space: Instead of precomputing all O(n^2) ranges A[i, j], for each position A[i], precompute only log n ranges of increasing size: A[i, i + 1],A[i, i + 2],A[i, i + 4],A[i, i + 8]. Any range A[l,r] can be decomposed into two ranges A[l, Y] and A[Z,r] where Y = l + 2^x and Z = r − 2^y such that Z ≥ l, Y ≤ r and, A[l, Y], A[Z,r] overlap. Then, RMQ(A[i, j]) = max(RMQ(A[l, Y]), RMQ(A[Z,r])) Total space cost O(n log n).
比如[18 79 65 87 54 16 35 94 23]要查询18~16这个区间的最大值，可以找max(max(18~87), max(65~16))，两个区间重叠
Query Expansion: User and documents may refer to a concept using different words (poison ↔ toxin, danger ↔ hazard) 同义词替换，错误拼写纠正，可提高recall，dealing with the vocabulary mismatch problem
Relevance feedback: is one way to expand/reformulate the query by incorporating feedback into the query process.
User relevance feedback: comes directly from the user. User provides feedback to the search engine by indicating which results are relevant. 用户直接反馈相关性
Pseudo Relevance (Blind Relevance Feedback): Take top-K results of original query Determine important/informative terms/topics shared by those documents, Expand query by those terms, No explicit user feedback needed 在你query的前k结果中找interesting的词然后加入query再用新query去查找，叫blind是因为user不知道这个过程
Indirect relevance feedback: For a query look at what users click on in the result page, Use clicks as signal of relevance, Learning-2-Rank uses neural models to rerank result pages 记鼠标动作或者在页面停留多久等统计相关性然后学习the feedback would not be from the current user seeking information, but other users who searched for the same query in the past.不立刻生效
Is it possible to perform query expansion without relevance feedback and vice versa 反之亦然? (a) Query expansion can be performed using global resources such as WordNet which do not require relevance feedback (b) Relevance feedback from a user (direct or indirect) can be incorporated without performing query expansion. This might be in the form of re-ranking existing documents or discovering documents that are similar to documents (the “more like this” button) to the ones users consider relevant.

Lecture 5. Index Construction and Advanced Queries

Inverted Index construction:
1. Static scenario: Static collection of documents, Offline construction, Fixed at query time
2. Real world scenario: Document collection grows over time, Documents arrive at a given rate, Documents should be searchable immediately, Might want to delete or update documents
Static construction: Split document collection into batches (X documents per batch) and construct inverted indexes for each batch. Merge batches at the end.
Merge batches: 1. Open all n files containing batches on disk 2. Read one term (or a few) from each file 3. Perform n-way merge, merging terms with same ids 4. Merge equivalent to appending bytes as document ids are increasing 5. Read the next terms 每批doc单独建index, 每次从每个batch中read一个或几个词 -> merge doc id -> 下一个词



From static to incremental indexing: Auxiliary辅助 Index 1. One static large static index on disk 2. As new documents arrive keep them in-memory in second index 3. If second index becomes too big, merge with static index 4. Query both indexes and merge results 2个index一个大的是static一个小的包含新的doc
问题：把小的index合并进大的开销很大，at each merge we look at all existing postings
改进：**Incremental, logarithmic indexing**: (reduce I/O costs) 1. Use a logarithmic number (log N) of indexes. At each level i, store index of size 2^i × n. Query all log N indexes at the same time and merge results 3. Construction cost: N log(N/n) 同一个level的数量够多后merge在一起成为上一个level
Our first level stores at most n postings. The next level stores at most 2n postings, the next level stores at most 4n postings and so on. If the a level fills up we only merge with the next higher level. As the size at the different level increases, merging becomes cheaper as less IOs are required in total.
The main disadvantage of the logarithmic layout is that, at query time a logarithmic number of indexes have to be queried. This introduces additional complexity and runtime overhead as the results of the different indexes have to be merged.
Phrase queries: 1. Seek to identify documents that contain a specific phrase “the who” 2. Combination of individual known terms that occur sequentially in documents 3. Two or more terms possible - “the president of the united states”
Phrase queries Approaches:
1. Inverted Index based: Positional Inverted Index
In addition to document ids and frequencies, store the positions of each term in the document. Perform intersection to find phrases. Index size dramatically increased.
and在doc 6中出现了2次，位置分别是12和43

term t	f_t	Postings list for t (docids,freqs,positions)
and	6	$\langle 1, 6, 7, \ldots \rangle, \langle 1, 2, 3, \ldots \rangle, \langle \langle 5 \rangle, \langle 12, 43 \rangle, \langle 6, 45, 212 \rangle, \ldots \rangle$
big	3	$\langle 2, 5, 42 \rangle, \langle 1, 1, 1 \rangle, \langle \langle 8 \rangle, \langle 43 \rangle, \langle 65 \rangle \rangle$
old	1	$\langle 32 \rangle, \langle 4 \rangle, \langle \langle 6, 34, 56, 59 \rangle \rangle$

Positional Index - Search: Intersect document ids of phrase terms, For the documents containing all terms, intersect position lists, Perform ranking on result set
对于包含phrase中所有term的文章处理

Positional Index - Intersection: To intersect n lists, pick the smallest and intersect with the second smallest. Use result of the intersection to keep intersecting. 从最短的list开始两两做merge，每个list代表一个term的positions，如果在第一个里面有6，53那第二个里就应该是7，54然后在第三个里找8，55...

5	6	13	53	234

2	4	6	12	53	434	545	656

1	3	4	6	9	10	11	12	13	14	18

1	2	3	4	6	8	9	10	13	14	15	17	19

2. String matching indexes (Suffix Arrays)

Alternative search index which does not require tokenization, Widely used in bioinformatics生物信息学 for exact string searches, Gives you more runtime guarantees than an inverted index

Pre-compute an index structure:

Suffix Arrays: Store all suffixes后缀 of the text, Sort them in lexicographical order, Each occurrence of P is a prefix of a suffix 每个pattern都是后缀的前缀

比如BANANA$可以拆成$, A$, NA$, ANA$, NANA$, ANANA$, BANANA$然后字典序排序

i	$SA[i]$	$T[SA[i]..n-1]T[0..SA[i]-1]$
0	18	$abracadabrabarbara
1	17	a$abracadabrabarbar
2	10	abarbara$abracadabr
3	7	abrabarbara$abracad
4	0	abracadabrabarbara$
5	3	acadabrabarbara$abr
6	5	adabrabarbara$abrac
7	15	ara$abracadabrabarb
8	12	arbara$abracadabrab
9	14	bara$abracadabrabar
10	11	barbara$abracadabra
11	8	brabarbara$abracada
12	1	bracadabrabarbara$a
13	4	cadabrabarbara$abra
14	6	dabrabarbara$abraca
15	16	ra$abracadabrabarba
16	9	rabarbara$abracadab
17	2	racadabrabarbara$ab
18	13	rbara$abracadabraba

比如在这里找bar，先二分b的位置，再在结果里二分a

Time complexity: O(m · log n), space O(n log n + |T|)

Compressed Suffix Arrays (CSA) use space equivalent to the compressed size of the text.

More advanced queries: Wildcard/misspelling queries (Sydney vs. Sidney: query S?dney), Regular expression queries (“[Jj]ohn.*”@smith.com???”), Proximity queries (“president” close to “obama”)

Lecture 6. IR Evaluation and re-ranking

Human Judgements: directly measure effectiveness in user studies; for reported satisfaction, completion of tasks, … but too expensive and slow, especially when tuning parameters of the system

Automatic evaluation assumptions: retrieval is ad-hoc临时， query performed only once, and with no prior knowledge of the user or their behavior无用户信息 effectiveness based on relevance, each document is either relevant or irrelevant to information need (often binary, sometimes also multiple grades of relevance)一般只有有关或无关有时有等级, relevance of documents are independent from others (no consideration of redundancy)相关性独立

Test collections: Several reusable test collections constructed for IR evaluation, e.g., for TREC competitions 包含corpus of documents, set of queries, relevance judgements (qrels)

precision @ k: compute precision using only ranks 1..k 前k个返回结果中有多少是对的

average precision: take average over prec@k for each k where rank k item is relevant; measure becomes rank sensitive 只枚举对的那几个k

Mean Average Precision (MAP): AP averaged across multiple queries 对于所有测试集

- 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).

- Many more measures exist!

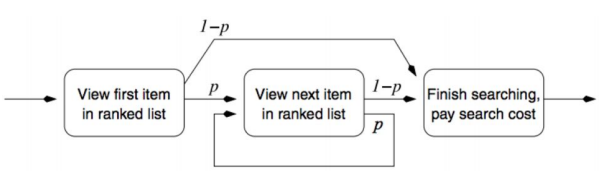
vector中1代表相关(是对的)

Reciprocal rank (RR): 1 / rank of first correct answer

- 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

Take mean over collection (MRR): for above two queries, mean(1, 0.33) = 0.67

Utility based metrics: Example: **Rank-biased precision**: User will pay $1 for each relevant answer but nothing for irrelevant answers. Models utility gained by searcher. User always looks at first result. User looks at second result with probability P. Third result: P^2,P^3,P^4…



doc	score	relevance
a	0.4	0
b	1.2	0
c	2.2	1
d	0.5	1
e	0.1	1
f	0.8	0

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

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

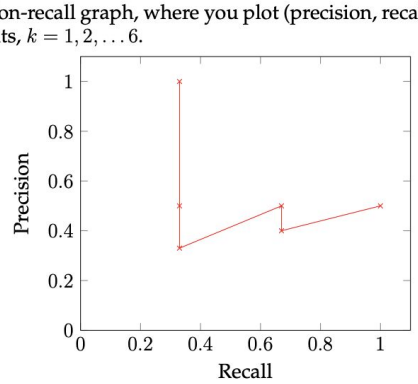
The first step is to form the vector of relevance judgments, ranked by document score <c, b, f, d, a, e> = <1, 0, 0, 1, 0, 1>

rank-biased precision (RBP), with $p = 0.5$

$$
\begin{aligned}
RBP &= (1-p) \times \sum_i r_i p^{i-1} \\
&= (1-0.5) \times \left(0.5^0 + 0.5^3 + 0.5^5\right) \\
&= \frac{1}{2} \times \left(1 + \frac{1}{8} + \frac{1}{32}\right) \\
&= \frac{1}{2} \times \frac{1}{32} \times (32 + 4 + 1) \\
&= \frac{37}{64}
\end{aligned}
$$

1, 4, 6相关 $P@3 = \frac{1}{3}$ $AP = \frac{1}{3} \times (\frac{1}{1} + \frac{2}{4} + \frac{3}{6}) = \frac{1}{3} \times 2 = \frac{2}{3}$

(d) plot the precision-recall graph, where you plot (precision, recall) point for the top k documents, $k = 1, 2, \ldots 6$.



Note this figure could be drawn in a more disconnected way, as it's questionable connecting the points because you can't return a fractional document.

What are the strengths and weaknesses of the methods above for evaluating IR systems?

precision@k is very easy to evaluate and easily understood, however it doesn't differentiate by rank within k, and doesn't incorporate any adjustment for the size of the relevant set.

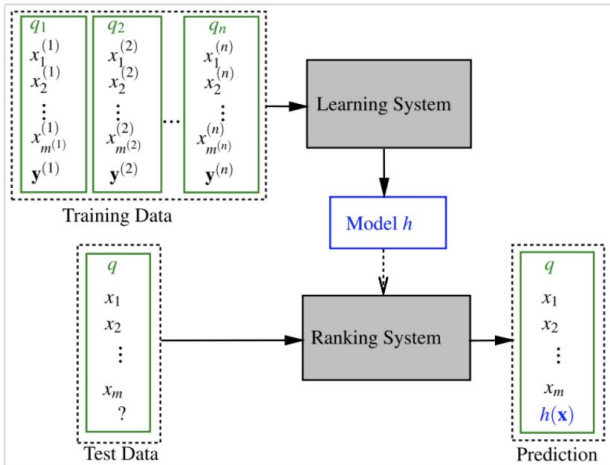
avg precision is less easily understood, but adds differentiation by rank, and adjusts for the size of the relevant set, however this requires knowing the relevant set (we typically don't know this); Averaging over multiple queries (mean average precision) with different number of relevant results also distorts the metric and it is very unintuitive of what the MAP score actually means.

rank biased precision has a similar formulation to AP, but it a little more intuitive. It includes differentiation by rank, and avoids the need to know the relevant set beyond the top ranked documents (as the contribution to RBP of lower ranked documents diminishes quickly, and can be upper bounded). A small issues is the need to know the free parameter, "p", which may differ between users and queries.

Effectiveness in practice: look at query logs and click logs, For a given query which result did users click on, After clicking, did they come back and click on other results, Indirect relevance feedback

Multi-stage retrieval: 1. Use a cheap, fast, simple similarity metric (such as BM25) to retrieve an initial set of relevant documents (top-k retrieval) 2. For those k documents, apply a Machine Learning algorithm which uses more features to re-rank the initial set of k documents.

Use training data consisting of tuples <q,d_i,u,r_i> which represent the query q, the k documents (d_1,…,d_k), user u and Relevance Vector R (r_1,…,r_k),



• Challenges: * Finding the right features representing input features * Defining the objective that we want to optimize that corresponds to ranking documents

Learning to rank deals with the problem of taking the top k results from a heuristic IR system, and reordering these to improve the user acceptability of the results.

User features: What kind of documents has the user been looking for? What kind of links is the user clicking on? How long does the user stay on a URL before returning?

Document features: Various tf/idf features (for example document lengths), Number of slashes in URL, Main topics (see Topic Models!)

Query Features: Number of queries terms, Popularity of the query (query log), Time sensitive? ”World Cup”

Point-wise objective:

* Given a query q, a document d_i, and a user u, find a function $f(q,d_i,u)$ that predicts r_i for document d_i.
* Ask the user: **How relevant is** d_i?
* Relevance judgement might be binary (yes or no) or multi-graded relevance (very relevant, relevant, not relevant)

Point-wise – Algorithm Sketch:

- Train classifier that can predict r_i
- Train model that can compute:

$$P(r_i = \text{relevant}|x_i)$$

- Sort documents by the probability of being relevant
- Multiple classes: Assign classes a value and compute expectation (e.g. -2 highly non relevant, 2 highly relevant)

Pair-wise objective:

* Given a query q, user u, and two documents d_1 and d_2 predict the correct relative order of d_1 and d_2
* Ask the user: Which of these **two** documents is more relevant?

Pair-wise — Sketch: no detail

- Train classifier to predict if $r_u < r_v$ based on pairs of training documents with the same query
- **RankNet** framed as $P(y_{u,v}|x_u, x_v) = \frac{1}{1+e^{\{f(x_u)-f(x_v)\}}}$
  * where f is a scoring function, x_u, x_v vectors representing the two documents, and $y_{u,v}$ is a binary value 1 → u better than v; 0 → v better than u
  * setting $f(x) = \theta.x$ recovers logistic regression with pairwise feature vectors $x_u - x_v$
- To re-rank a test query, sort by value of $f(x)$

Lecture 7. Text Classification

Text classification task examples: Topic classification, Sentiment analysis, Authorship attribution, Automatic fact-checking

Polarity lexicons: 极性词典，一个词的情绪信息

Why is text classification generally a difficult problem? document representation (how do we identify features of the document), feature selection (单个词通常无法表达意思), sparse data problem (bi-gram, tri-gram又会数据稀疏)

Building a Text Classifier: 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

Choosing a classification algorithm: Bias vs. Variance; Feature independence; Feature scaling; Complexity; Speed

Naïve Bayes: probability of the class times probability of features given the class.

<u>Pros</u>: Fast to "train" and classify; robust, low variance; good for low data situations; optimal classifier if independence assumption is correct; extremely simple to implement.

<u>Cons</u>: Independence assumption rarely holds; low accuracy compared to similar methods in most situations; smoothing required for unseen class/feature combinations

Logistic Regression: a classifier, A linear model, but uses softmax "squashing" to get valid probability

<u>Pros</u>: A simple yet low-bias classifier; unlike Naïve Bayes not confounded by diverse, correlated features 不被互相相关的feature干扰

<u>Cons</u>: Slow to train; some feature scaling issues; often needs a lot of data to work well; choosing regularisation a nuisance but important since overfitting is a big problem

Support vector machines: Finds hyperplane which separates the training data with maximum margin

<u>Pros</u>: fast and accurate linear classifier; can do non-linearity with kernel trick; works well with huge feature sets

Cons: Multiclass classification awkward; feature scaling can be tricky; deals poorly with class imbalances; uninterpretable

K-Nearest Neighbour: Classify based on majority class of k-nearest training examples in feature space

Pros: Simple, effective; no training required; inherently multiclass; optimal with infinite data

Cons: Have to select k; issues with unbalanced classes; often slow (need to find those k-neighbours); features must be selected carefully

> k-Nearest Neighbour using **Euclidean distance**: Often this is a bad idea, because Euclidean distance tends to classify documents based upon their length — which is usually not a distinguishing characteristic for classification problems.
>
> k-Nearest Neighbour using **Cosine similarity**: Usually better than the previous, because we're looking at the distribution of terms. However, k-NN suffers from high-dimensionality problems, which means that our feature set based upon the presence of (all) words usually isn't suitable for this model.

Decision tree: Construct a tree where nodes correspond to tests on individual features; Leaves are final class decisions; Based on greedy maximization of mutual information

Pros: in theory, very interpretable; fast to build and test; feature representation/scaling irrelevant; good for small feature sets, handles non-linearly-separable problems

Cons: In practice, often not that interpretable; highly redundant sub-trees; not competitive for large feature sets

> Decision Trees using Information Gain: Decision Trees can be useful for finding meaningful features, however, the feature set is very large, and we might find spurious correlations 虚假相关性. More fundamentally, Information Gain is a poor choice because it tends to prefer rare features; in this case, this would correspond to features that appear only in a handful of documents.

Random forests: Consists of decision trees trained on different subsets of the training and feature space; Final class decision is majority vote of sub-classifiers

Pros: Usually more accurate and more robust than decision trees, a great classifier for small- to moderate- sized feature sets; training easily parallelised

Cons: Same negatives as decision trees: too slow with large feature sets

Neural Networks: An interconnected set of nodes typically arranged in layers; Input layer (features), output layer (class probabilities), and one or more hidden layers

Pros: Extremely powerful, state-of-the-art accuracy on many tasks in natural language processing and vision

Cons: Not an off-the-shelf 现成的 classifier, very difficult to choose good parameters; slow to train; prone to overfitting

HyperParameter tuning: 1. Dataset for tuning (k-fold cross-validation); 2. Specific hyperparameters are classifier specific; 3. For multiple hyperparameters, use grid search (穷举所有超参组合)

	Classified As	
Class	A	B
A	79	13
B	8	10

Accuracy: correct classifications/total classifications = (79 + 10)/(79 + 13 + 8 + 10) = 0.81
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

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

Like precision and recall, defined relative to a specific positive class

But can be used as a general multiclass metric

* Macroaverage: Average F-score across classes
* Microaverage: Calculate F-score using sum of counts

Lecture 8. N-gram language models

Language Models (LM): Assign a probability to a sequence of words 一系列词出现的概率:
Query completion, Spelling correction, Machine translation, Dialogue systems 对话系统
Chain rule:

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

The Markov Assumption: $P(w_i|w_1\ \ ...w_{i-1}) \approx P(w_i|w_{i-n+1}...w_{i-1})$

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

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

Maximum Likelihood estimation: Estimate based on counts in our corpus

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,

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

Trigram example: n-gram的在前面生成n-1个<s>，尾部一个</s>

Corpus:

What is the probability of

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

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

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

P(sent=yes no no yes)

= P(*yes* |<s> <s>) * P(*no* | <s> *yes*) * P(*no* | *yes no*)

* P(*yes* | *no no*) * P(</s> | *no yes*)

= ½ * 1 * ½ * $\frac{2}{5}$ * ½ = 0.1

Several problems: 1. Language has long distance effects — need large n; 2. Resulting probabilities are often very small (Use log); 3. No probabilities for unseen words; 4. Words in new contexts (By default, zero count for any n-gram, 需要smooth)

Smoothing (or discounting): give events you've never seen before some probability, Must be the case that P(everything) = 1 本质上是从概率比较高的那些情况中挪一些给0概率的保持和为1

Many different kinds of smoothing:

1. Laplacian (add-one) smoothing: pretend we've seen each n-gram once more than we did.

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

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

For bigram models,

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

Example: <s> the rat ate the cheese </s>

bigram probability P(ate|rat) under add-one:

$= \frac{C(rat\ ate)+1}{C(rat)+|\boldsymbol{V}|} = \frac{2}{6}$ V = { the, rat, ate, cheese, </s> }

bigram probability P(ate|cheese) under add-one smoothing:

$= \frac{C(cheese\ ate)+1}{C(cheese)+|\boldsymbol{V}|} = \frac{1}{6}$

2. Add-k smoothing: Adding one is often too much, Instead, add a fraction k

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

3. Kneser-Ney smoothing: State-of-the-art method for n-gram language models. A fairly complex method, combining three ideas 三种合并的不考:

* Interpolation (or alternatively, backoff) * Absolute discounting * Continuation counts

Backoff and Interpolation: Smooth using lower-order probabilities (less context)

Backoff: fall back to n-1-gram counts only when ngram counts are zero 只在是0时降

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

P* and α must preserve "sum to 1" property

Interpolation: taking a linear combination of all relevant probabilities 日常考虑全部1~n

$$P_{interp}(w_i|w_{i-2}, w_{i-1}) = \lambda(w_{i-2}, w_{i-1})\, P(w_i|w_{i-2}, w_{i-1}) + (1 - \lambda(w_{i-2}, w_{i-1}))P_{interp}(w_i|w_{i-1})$$

λs can be constant across all contexts * But better if sensitive to n-grams

Absolute discounting:

留存语料库 heldout corpus 一个在训练集中平均出现1次的bigrams在heldout中平均出现1-0.75次，发展出从每个非零计数里减掉一个固定的 discount δ ∈ [0,1]

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

Doesn't effect high counts much 对于本身count高的几乎不受影响

Continuation counts: When backing-off or interpolating, raw counts can be fairly unreliable

* $P(Zealand|Old) = ?$ will interpolate with $P(Zealand)$

在插值时会考虑Zealand出现的次数，但是这个词大多出现在New之后

Instead, use frequency of contexts in which word appears

For many words, closely related to total count

But just 1 for *Zealand*

$continuation_count(w_1) = |\{v: count(v, w_1) > 0\}|$ 不考

Throwing it all together:

$$P_{KN}(w_i|w_{i-2}, w_{i-1}) = \frac{\max(0, C_{KN}(w_{i-2}, w_{i-1}, w_i) - d)}{C_{KN}(w_{i-2}, w_{i-1})}$$

$$+ \lambda(w_{i-2}, w_{i-1})P_{KN}(w_i|w_{i-1})$$

$$\lambda(w_{i-2}, w_{i-1}) = \frac{d}{C_{KN}(w_{i-2}, w_{i-1})} |\{w: C_{KN}(w_{i-2}, w_{i-1}, w) > 0\}|$$ 仅了解，不考

Evaluation: 1. Extrinsic外在: spelling correction, machine translation 实践中去试

2. Intrinsic固有: Perplexity on held-out test set 直接可以计算出来的标准

Perplexity: Inverse probability of entire test set 对于整个测试集，数值越小越好

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

equivalently

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

Lecture 9. Lexical Semantics

Lexical semantics: How the meanings of words connect to one another. Manually constructed resources: lexicons, thesauri, ontologies, etc 人工搞出来的

homonyms: 同一个词有完全不同的意思 1 2

polysemy: Related senses of the same lexical item 词有相关的但不完全相同的意思 1

Bank (noun):

1. A financial institution; a building where a financial institution offers services; a repository; a container for holding money

2. Land sloping down to a body of water

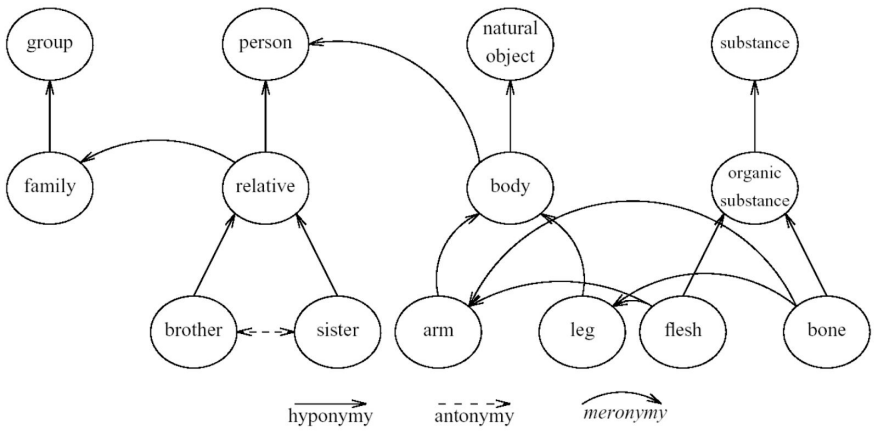
Synonyms: 近义词 **Antonyms**: 反义词

Hypernyms: 上位generic通用 **Hyponyms**: 下位specific

One word is a hypernym of a second word when it is a more general instance ("higher up" in the hierarchy) of the latter, for example, reptile爬虫 is the hypernym of snake (in its animal sense).

Holoynms: whole **Meronyms**: part

One word is a meronym of a second word when it is a part of the whole defined by the latter, for example, scales (the skin structure鳞) is a meronym of reptile.



WordNet: A database of lexical relations; The nodes of WordNet are not words, but meanings; There are represented by sets of synonyms, or synsets:

```
>>> nltk.corpus.wordnet.synsets('bank')

[Synset('bank.n.01'), Synset('depository_financial_institution.n.01'), Synset('bank.n.03'), Synset('bank.n.04'),
Synset('bank.n.05'), Synset('bank.n.06'), Synset('bank.n.07'), Synset('savings_bank.n.02'), Synset('bank.n.09'),
Synset('bank.n.10'), Synset('bank.v.01'), Synset('bank.v.02'), Synset('bank.v.03'), Synset('bank.v.04'),
Synset('bank.v.05'), Synset('deposit.v.02'), Synset('bank.v.07'), Synset('trust.v.01')]

>>> nltk.corpus.wordnet.synsets('bank')[0].definition()

u'sloping land (especially the slope beside a body of water)'

>>> nltk.corpus.wordnet.synsets('bank')[1].lemma_names()

[u'depository_financial_institution', u'bank', u'banking_concern', u'banking_company']
```

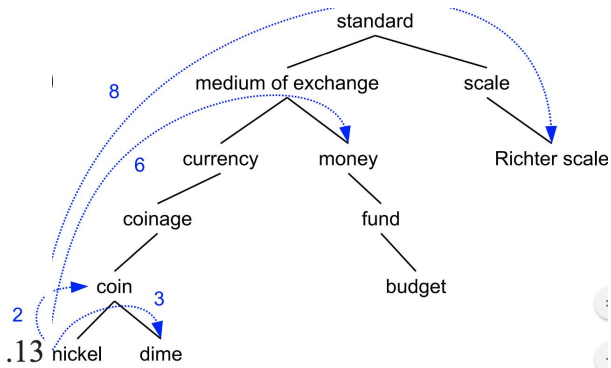
- Connections between nodes are lexical relations
- Including all the major ones mentioned earlier

```
>>> print nltk.corpus.wordnet.lemmas('sister')[0].antonyms()

[Lemma('brother.n.01.brother')]

>>> nltk.corpus.wordnet.synsets('relative')[0].hypernyms()

[Synset('person.n.01')]

>>> nltk.corpus.wordnet.synsets('body')[0].part_meronyms()

[Synset('arm.n.01'), Synset('articulatory_system.n.01'), Synset('body_substance.n.01'), Synset('cavity.n.04'),
Synset('circulatory_system.n.01'), Synset('crotch.n.02'), Synset('digestive_system.n.01'),
Synset('endocrine_system.n.01'), Synset('head.n.01'), Synset('leg.n.01'), Synset('lymphatic_system.n.01'),
Synset('musculoskeletal_system.n.01'), Synset('neck.n.01'), Synset('nervous_system.n.01'),
Synset('pressure_point.n.01'), Synset('respiratory_system.n.01'), Synset('sensory_system.n.02'),
Synset('torso.n.01'), Synset('vascular_system.n.01')]
```

Word similarity with paths: Given WordNet, find path-similarity based on path length in hypernym/hyponym tree.

$\text{simpath}(c_1,c_2) = 1/\text{pathlen}(c_1,c_2)$

pathlen = 包括c1,c2路过的总词数



simpath(*nickel,coin*) = 1/2 = .5

simpath(*nickel,currency*) = 1/4 = .25

simpath(*nickel,money*) = 1/6 = .17

simpath(*nickel,Richter scale*) = 1/8 = .13

Wup_similarity: edges vary widely in actual semantic distance 靠近根的词义相差大

include depth information (Wu & Palmer), find lowest common subsumer (LCS)

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

根standard深度是1

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

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

		entity abstraction... communication message...		
entity abstraction... communication message...	entity abstraction... psychological... cognition...	statement pleading charge... accusation...	entity abstraction... group... collection...	entity abstraction... measure system of meas... information meas...

information

entity physical... process... processing data process... operation computer op...	entity abstraction... psychological... cognition... process... basic cog... memory...	entity abstraction... psychological... event act...

retrieval

- The "message" sense of *information* lies at depth 5; the "data retrieval" sense of *retrieval* is at depth 8; the lowest node in the hierarchy that they share is `entity` (the root node) so the similarity is:

$$\begin{aligned}\text{sim}(\text{information}, \text{retrieval}) &= \frac{2 \times 1}{5+8} \\ &= \frac{2}{13} \approx 0.154\end{aligned}$$

- What about the "message" sense of *information* with the other senses of *retrieval*?
  - The "memory" sense of *retrieval* is at depth 8, and the lowest node shared is `abstraction, abstract entity` (at depth 2); this means that the similarity is $\frac{2\times2}{8+5} \approx 0.308$.
  - The "event" sense of *retrieval* is at depth 6, and the lowest node shared is also `abstraction, abstract entity`, so the similarity is $\frac{2\times2}{6+5} = 0.364$.

depth要加上最底层的information和retrieval

Lin_similarity: include statistics from corpus (Resnik; Lin)

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

$$P(c) = \frac{\sum_{w \in words(c)} count(w)}{N}$$ 在词库中的比例，比如词库里有多少词属于standard

Information content (IC):

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

Lin distance:

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

Word Sense Disambiguation (WSD): is the computational problem of automatically determining which sense (usually, Wordnet synset) of a word is intended for a given token instance with a document. 消歧，当一个词有多个意思时选哪个

1. Supervised WSD: Apply standard machine classifiers; Feature vectors typically words and syntax around target 根据上下文分析 Requires sense-tagged corpora建文库耗时 选上下文范围，但是上下文中的词同样ambiguous
2. Less supervised approaches:
   (1) Lesk: Choose sense whose dictionary gloss from WordNet most overlaps with the context 选意思与上下文重叠最多的意思
   (2) Yarowsky: bootstrap引导 method; Create a small seed training set; Iteratively expand training set with untagged examples; Uses one sense per collocation, one sense per document self-training
   (3) Graph methods in WordNet

Lecture 10. Distributional Semantics

Distributed = represented as a numerical vector (in contrast to symbolic)

Distributional = how the word occurs in the context

The Vector space model: represent meaning as a vector (docid和term)

	…	state	fun	heaven	…
…					
425		0	1	0	
426		3	0	0	
427		0	0	0	
……					

Manipulating the VSM: Weighting the values 把表中的count计算成TF-IDF, Creating low-dimensional dense vectors, Comparing vectors

Dimensionality reduction: Term-document matrices are very sparse 表中0很多，降维

More practical (less features), Remove noise (less overfitting)

Singular value Decomposition:

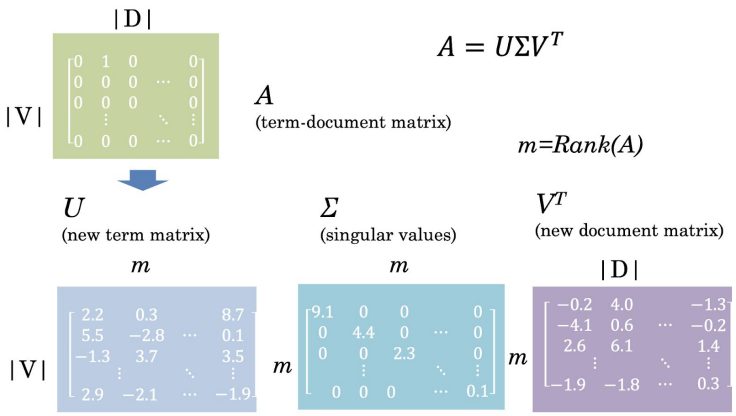
Using the SVD method to build a representation of our matrix which can be used to identify the most important characteristics of documents. By throwing away the less important characteristics, we can have a **smaller representation of the collection**, which will save us (potentially a great deal of) time when evaluating the cosine similarities between the documents and the query.

- In brief, for a (truncated) SVD: $M = U_k \Sigma_k V_k^T$, our document collection is represented as $U_k \Sigma_k$, and then the transformed query can be found as: $q_k = q V_K$ (note the transposition is gone).

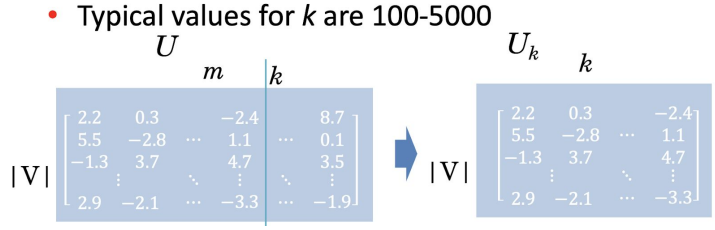
<u>need the query and the documents to have the same number of dimensions</u>

Truncating – latent semantic analysis:

- Truncating *U, Σ,* and *V* to *k* dimensions produces best possible *k* rank approximation of original matrix
- So truncated, U_k (or V_k^T) is a new low dimensional representation of the word (or document)



- Typical values for *k* are 100-5000



Words as context: Lists how often words appear with other words * In some predefined context (usually a window 在一个滑窗中同时出现）the和state同时出现1973次

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

raw freq dominated by common words

Pointwise mutual information: obviously biased towards rare words, doesn't handle zeros

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

越大两个词越可能同时出现

x= state, y = country
p(x,y) = 10/15871304 = 6.3×10^{-7}
p(x) = 12786/15871304 = 8.0×10^{-4}
p(y) = 3617/15871304 = 2.3×10^{-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 tricks: Zero all negative values (PPMI) (Avoid –inf and unreliable negative values); Smooth probabilities (解决Counter bias towards rare events)

What is a <u>word embedding</u> and how does it relate to <u>distributional similarity</u>?

- We're going to have a representation of words (based on their contexts) in a vector space, such that other words "nearby" in the space are similar 用vector表示一个词且相近的词的表示类似
- Using a dimensionality-reduction method like SVD helps keep this to a manageable size, and, if we're lucky, allows us to emphasise the more meaningful contexts (and de-emphasise meaningless contexts, like the). 用SVD丢弃不是很重要的feature
- The row corresponding to the word in the relevant (target/context在这两个表中同一个词那一行) matrix is known as the "embedding".

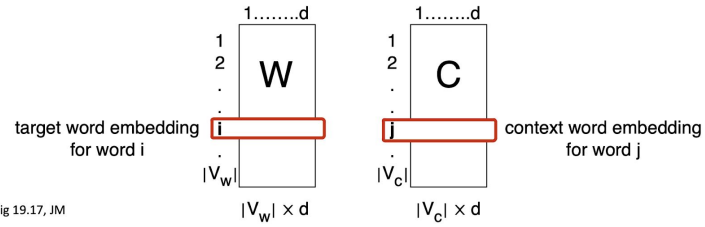


Fig 19.17, JM

What is the difference between a <u>skip-gram</u> model and a <u>CBOW</u> model?

Skip-gram models analyse the probability of the context words given the target word 由一个词分析附近其他词; CBOW models analyse the probability of the target word given the context words 由附近的词分析目标词.

How are the above models trained?

they are based around taking the dot product of the relevant vectors (or average of vectors, in the case of CBOW), and then marginalising.

Lecture 11. Part of speech tagging

POS tag: is a label assigned to a token in a sentence which indicates some grammatical (primarily syntactic) properties of its function in the sentence.

Open Classes: Nouns, Verbs, Adverbs, Adjectives

Closed Classes: Prepositions, Determiners, Pronouns, Conjunctions, Modals

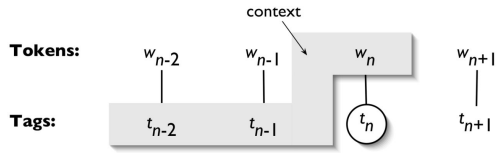
Major English tagsets: Brown, Penn Treebank, CLAWS/BNC, "Universal"

Why automatically pos tag? Important for morphological analysis, nouns are important for information retrieval, adjectives for sentiment analysis, POS tags can offer word sense disambiguation

Automatic Taggers: 1. Rule-based taggers 2. Statistical taggers: Unigram tagger, Classifier-based taggers, Hidden Markov Model (HMM) taggers ()

Rule-based tagging: Typically starts with a list of possible tags for each word; Apply rules to narrow down to a single tag (If DT定语 comes before word, then eliminate VB)

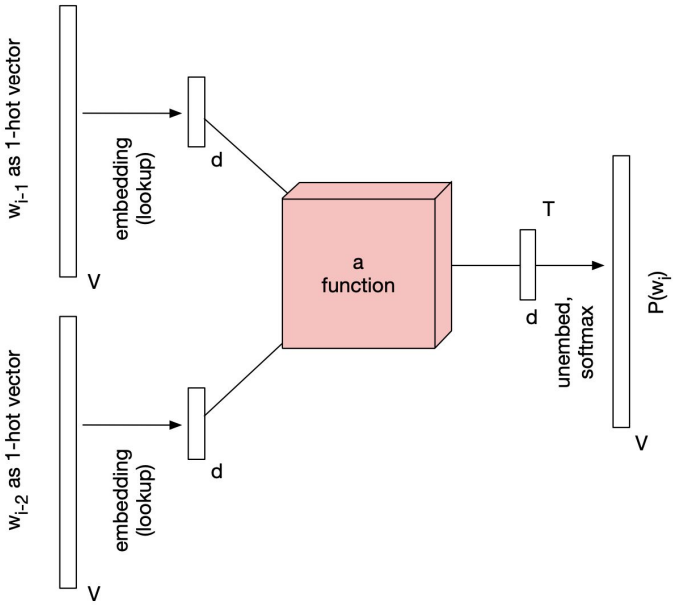
Unigram tagger: Assign most common tag to each word type; Requires a corpus of tagged words; “Model” is just a look-up table; But actually quite good, ~90% accuracy.
N-gram: Assign a POS tag to a token according to the most common tag in the same sequence (based on the sentence in which the token occurs) of n tokens (or tags) in the tagged corpus
Classifier-Based tagging (“MEMM”): Use a standard discriminative classifier (e.g. logistic regression, neural network), with features: 1. Target word 2. Lexical context around the word 3. Already classified tags in sentence
But can suffer from error propagation: wrong predictions from previous steps affect the next ones 可能错一个之后全错



Sequential: Learn a Hidden Markov Model (or other model) based on the observed tag sequences in a tagged corpus.
Unknown words: 用只见过一次的词去猜或use sub-word representations to capture morphology

Lecture 12. Neural sequence models

Feed forward neural net LM: 参数从输入层向输出层单向传播



Ngram LMs 优缺点: cheap to train (just compute counts), but too many parameters, problems with sparsity and scaling to larger contexts, don’t adequately capture properties of words (grammatical and semantic similarity), e.g., film vs movie
NNLMs more robust (Neural network优点): force words through low-dimensional embeddings, automatically capture word properties, leading to more robust estimates, flexible: minor change to adapt to other tasks (tagging)

Neural networks: Based on vector embeddings and compositional functions over these vectors:

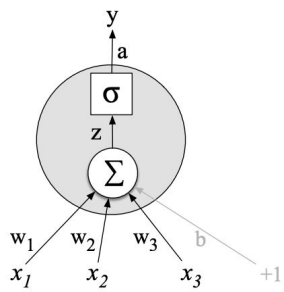
1. Word embeddings capture grammatical and semantic similarity “cows” ~ “sheep”, “eats” ~ “chews” etc.
2. Vector composition can allow for combinations of features to be learned (e.g., humans consume meat)
3. Limit size of vector representation to keep model capacity under control.

Neural network components:

Typically have several hidden units, i.e.,

$$h_i = \tanh\left(\sum_j w_{ij}x_j + b_i\right)$$

* each with own weights (w_i) and bias term (b_i)



<- a single unit

Coupling the Output layer: Add another layer, which takes h as input, and maps into |V| sized vector, Softmax ensures probabilities >0 & sum to 1 字典大小的每个词出现的概率

$$\left[\frac{\exp(v_1)}{\sum_i \exp(v_i)}, \frac{\exp(v_2)}{\sum_i \exp(v_i)}, \cdots \frac{\exp(v_m)}{\sum_i \exp(v_i)}\right]$$

Deep structures: Each layer typically fully-connected to next lower layer, i.e., each unit is connected to all input elements

1. map from 1-hot words, ***w***, to word embeddings, ***e*** (lookup)
2. transform ***e*** to hidden state $\boldsymbol{h_1}$ (with non-linearity)
3. transform $\boldsymbol{h_1}$ to hidden state $\boldsymbol{h_2}$ (with non-linearity)
4. *… repeat …*
5. transform $\boldsymbol{h_n}$, to output classification space ***y*** (with softmax)

Learning from Data: learn the parameters (sets of weights, bias, embeddings) from data. tools like tensorflow, pytorch, dynet use autodiff to compute gradients automatically梯度下降

MEMM (Maximum Entropy Markov Model) tagger: 需要的input

* recent words w_{i-2}, w_{i-1}, w_i
* recent tags t_{i-2}, t_{i-1}

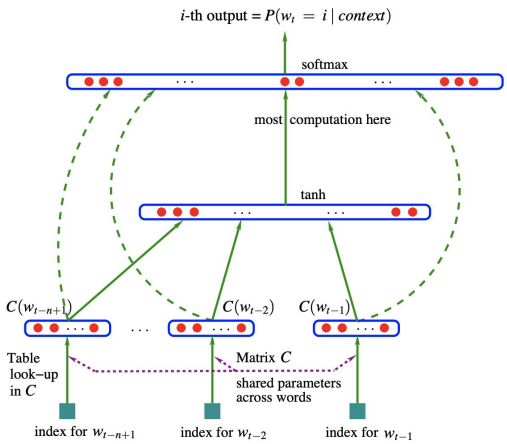
输出的output: current tag t_i

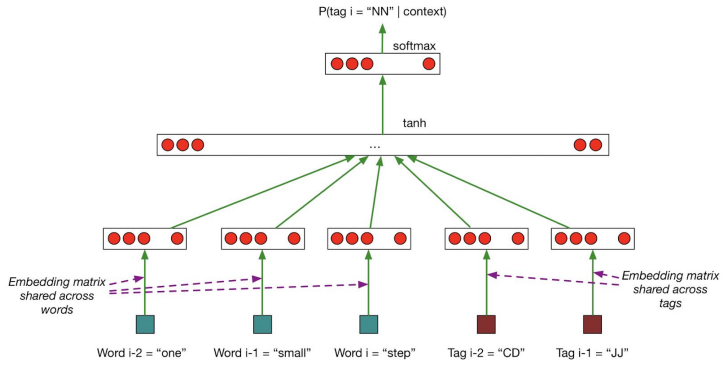
FF-NN for Tagging: 需要的5个input: 3 x word embeddings and 2 x tag embeddings

输出的output: vector of size |T|, using softmax 每个tag的概率

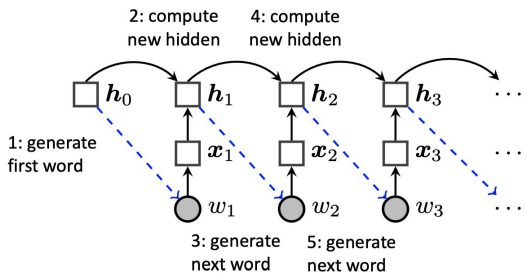
- Train to minimise $-\sum_i \log P(t_i | w_{i-2}, w_{i-1}, w_i, t_{i-2}, t_{i-1})$

FF-NN-LM (Feed forward Neural Net Language Model):





Recurrent NNLMS (RNNs):



每次取一个词然后会改变h，下一个词用新h

Recurrent means that model is structured such that it can be repeatedly applied for each item in a sequence. The recurrence in a RNN is over the hidden states, such that each as new input is fed into the model, this is used to formulate a new hidden state – as a non-linear transformation of the last hidden state, and the new input. In this way the approach can (in theory) represent long-distance phenomena in the sentence, over variable distances.

A FFLM instead assumes a fixed sized context, which can be applied using a ”sliding window” over a sequence.

对于普通NN为了考虑前文每次input会输入前3个embeddings，RNN每次只输入一个，在formulate a new hidden state时自动考虑了前文

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

Bidirectional RNNS: Tagging can be benefit from context to left and right * easy: use two RNNs, left-to-right and right-to-left 一个RNN从前文推一个从后文推

What advantage does a RNN language model have over a feed-forward language model?

RNNLM can capture long-distance dependencies, while FFLM cannot. For example, it can balance quotes and brackets over long distances.

Name the key differences and similarities between n-gram language models versus feed-forward neural language models.

1. n-gram language models and FFNNs share the same setup of using a Markov chain.
2. They factorise the probability of a sentence into the probability of each word given the n − 1 previous words.

3. n-gram models can be considered a “feature-based” model where every ngram is a feature, with a corresponding weight.
4. FFNNLM use an embedding and un-embedding step, to limit the model size and force generalisation.
5. Another key difference between n-grams and FFLM is that n-grams work over highly sparse data, while FFLM work over dense representations

Lecture 13. Information Extraction (IE)

Main goal: turn text into structured data such as databases, etc.

Examples: Help decision makers in applications. Stock analysis: Gather information from news and social media → summarise into a structured format → decide whether to buy/sell at current stock price

- Given this:
  - *“Brasilia, the Brazilian capital, was founded in 1960.”*
- Obtain this:
  - capital(Brazil, Brasilia)
  - founded(Brasilia, 1960)

Two steps:

1. Named Entity Recognition (NER): find out entities such as “Brasilia” and “1960”
2. Relation Extraction: use context to find the relation between “Brasilia” and “1960”(“founded”)

Named Entity Recognition (NER): sequence models such as seq. classifiers, HMMs (Hidden Markov Model) or CRFs (Conditional random fields).

Relation Extraction: mostly classifiers, either binary or multi-class.

NER as sequence labelling: NE tags can be ambiguous (“Washington”人名或地名) -> 从上下文推断 -> 直接用 an out-of-the-box(马上可以用的) sequence tagger? -> Not really: entities can span multiple tokens 由多个词组成一个短语

IO tagging:

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

多个词组用I表示，单个词为O

American/I-ORG Airlines/I-ORG 即这两个词组成一个ORG(organization)的词组

Dealing with adjacent entities: IOB tagging:

- **[ORG American Airlines],** a unit of **[ORG AMR Corp.]**, immediately matched the move, spokesman **[PER Tim Wagner]** said.
- 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**

每个词组开头改用B而不是I

NER: features: Character and word shape features (ex: “L’Occitane”), Prefix/suffix, Word shape, POS tags / syntactic chunks, Presence in a gazeteer 查表: lists of entities, such as place names, people’s names and surnames, etc.

Relation extraction - methods:

1. If we have access to a fixed relation database: Rule-based, Supervised, Semi-supervised, Distant supervision
2. If no restrictions on relations: Unsupervised, Sometimes referred as “OpenIE”

Rule-based relation extraction: “Agar is a substance prepared from a mixture of red algae, such as Gelidium, for laboratory or industrial use.” -> [NP red algae] , such as [NP Gelidium] -> hyponym(Gelidium, red algae)　但一些关系很难找到rule

Supervised relation extraction: Assume a corpus with annotated 注释 relations.

1. Find if an entity pair is related or not (For each sentence, gather all possible entity pairs. Annotated pairs are considered positive examples. Non-annotated pairs are taken as negative examples)
2. For pairs predicted as positive, use a multiclass classifier to obtain the relation. (**OpenIE**)
   - **[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 relation extraction: Annotated corpora is very expensive to create. Assume we have a small set of seed tuples. Mine the web for text containing the tuples:

- Given hub(Ryanair, Charleroi)
- Get sentences containing all terms, e.g., “Budget airline **Ryanair**, which uses **Charleroi** as a **hub**, scrapped all weekend flights out of the airport.”
- Use these patterns to new tuples, e.g., hub(Jetstar, Avalon) as these words occur in similar contexts; repeat　自己制造数据集，可能不准确

Assume the existence of seed tuples

Distant supervision: Obtain new tuples from a range of sources; Generate massive training sets, enabling the use of richer features, and no risk of semantic drift 也自己造数据但无风险 Still rely on a fixed set of relations.

Mining new tuples from DBpedia, Freebase

ReVERB: Unsupervised relation extraction: If there is no relation database or the goal is to find new relations, unsupervised approaches must be used. 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)

Main problem: mapping the substring relations into canonical forms

Evaluation: NER: F1-measure at the entity level.

Relation Extraction with known relation set: F1- measure

Relation Extraction with unknown relations: much harder to evaluate 需要人工，数据太大的话取 a small sample, Can only obtain (approximate) precision, not recall.

Temporal expressions: 时间表达，text中有last weekend，可根据metadata确定具体时间并Normalisation
Event extraction: 提取事件，detect how a set of events happened in a timeline 事件时间轴，与temporal expression相关
Template filling: Some events can be represented as templates. Goal is to fill these slots given a text. Models can take the template information into account to ease the learning and extraction process. 把事件填入模板

Lecture 14. Question Answering

Question Answering (QA): is the task of automatically determining the answer (set) for a natural language question
factoid: a brief or trivial item of news or information. 简短的叙述 non-factoid questions require a longer answer, critical analysis, summary, calculation
Two Key Approaches:

1. Retrieval, find answer question via string/text passage in a document (collection)
2. Natural Language Interface to Database ("NLIDB"), automatically construct a query, and answer question relative to fixed KB
   第一种是只有一个wiki要自己处理，第二种有一个structured database做资料库

IR-based Factoid QA: TREC-QA

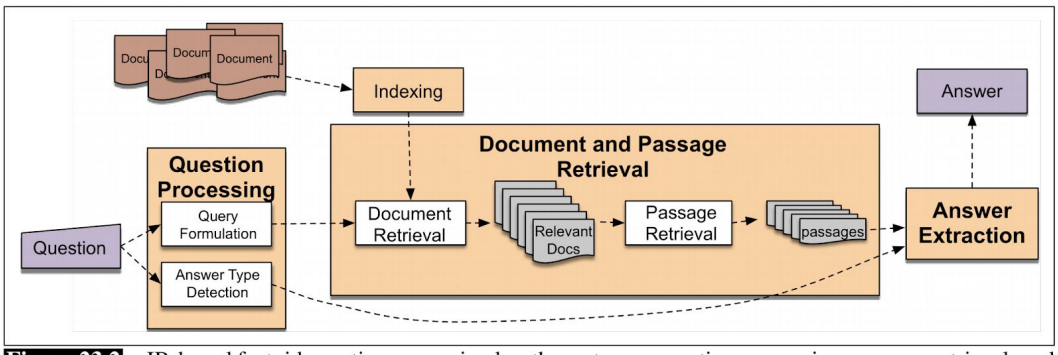


Figure 23.2 IR-based factoid question answering has three stages: question processing, passage retrieval, and answer processing.

Step
1. Use question to make query for IR engine
2. Find document, and passage within document
3. Extract short answer string

Step 1.1 Question Processing:
Find key parts of question that will help retrieval
扔掉wh开头的词
Formulate as tf-idf query, using unigrams or bigrams
Identify entities and prioritise match比如entities是英国和总统，prioritise是现任)
May re-formulate question using templates (query = ["xxxxx is located [in/at]"])
Step 1.2 Answer Type:
Predict expected answer type (here = LOCATION) 帮助之后提取答案，key feature is question headword: who -> HUMAN, What -> ENTITY

Step 2 Retrieval: Find top n documents matching query (standard IR), Next find passages (paragraphs or sentences) in these documents, Re-rank IR outputs to find best passage (e.g., using supervised learning)

Step 3 Answer Extraction: Find a concise answer to the question, as a span in the text; Framed as classification (various features based on match to question, expected entity type match, specific answer patterns)

另一种 **Step 3 Deep learning for answering question from passage**:

dataset: SQUAD - Stanford question answering dataset

DrQA:

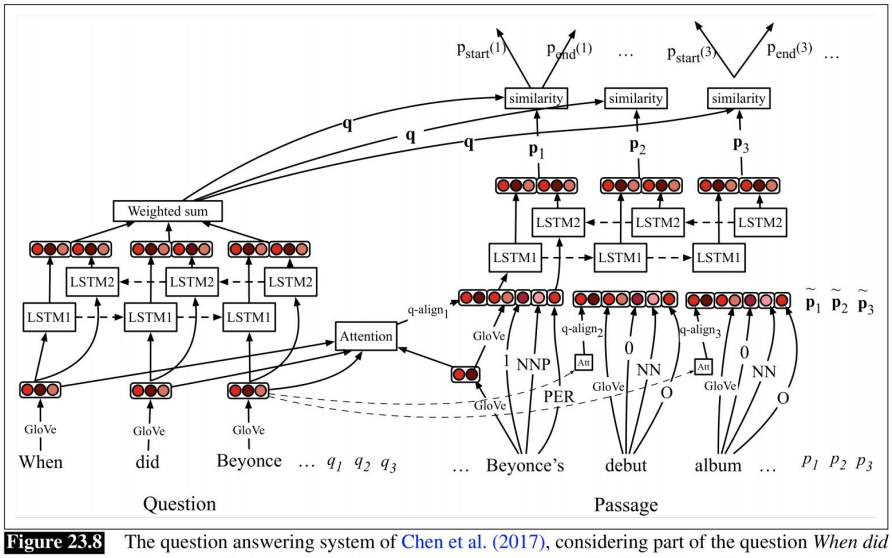


Figure 23.8 The question answering system of Chen et al. (2017), considering part of the question *When did Beyoncé release Dangerously in Love?* and the passage starting *Beyoncé's debut album, Dangerously in Love (2003).*

最顶上输出的是每个word是answer开头和结尾的概率

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

Many large knowledge bases: Sports statistics, Moon rock data, Freebase, DBpedia, Yago, …

Each with own query language SQL, SPARQL etc.

"When was Ada Lovelace born?" → `birth-year (Ada Lovelace, ?x)`

"What is the capital of England?" → `capital-city(?x, England)`

What is semantic parsing, and why might it be desirable for QA? Why might approaches like NER be more desirable?

• As opposed to syntactic parsing - which attempts to define the structural relationship between elements of a sentence - we instead want to define the (meaning–based) relations between those elements.

• For example, in the sentence Donald Trump is president of the United States. we can deduce that Donald Trump is the subject of the verb is, and so on, but in semantic parsing, we might be trying to generate a logical relationship like is(Donald Trump, president(United States)).

• This format allows us to answer questions like "Who is president of the United States?" by generating an equivalent representation like: is(?,president(United States))

Lecture 15. Sequence Tagging: hidden Markov models

为了避免error propagation的基于DP的模型，前面的错误预测不影响后面的结果

Sequence labelling or **Structured prediction**: Define a model that decomposes process into individual word level steps 定义模型时拆成一个一个的词，But that takes into account the whole sequence when learning and predicting 但学习和预测时考虑整个sequence

Hidden Markov Model:

* $P(\boldsymbol{w}|\boldsymbol{t}) = \prod_{i=1}^{n} P(w_i|t_i)$ [Prob. of a word depends only on the tag]
* $P(\boldsymbol{t}) = \prod_{i=1}^{n} P(t_i|t_{i-1})$ [Prob. of a tag depends only on the previous tag]

当前词只与本身的tag和前一个词的tag有关(bigram)

Markov assumption: the likelihood of transitioning into a given state depends only on the current state, and not the previous state(s) (or output(s))

Output independence assumption: the likelihood of a state producing a certain word does not depend on the preceding (or following) states.

emission (O): $P(w_i|t_i)$ and **transition (A)**: $P(t_i|t_{i-1})$

Training uses Maximum Likelihood Estimation (MLE):

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

- What about the first tag?
  * Assume we have a symbol "<s>" that represents the start of your sentence.

$$P(NN| < s >) = \frac{count(< s >,\ NN)}{count(< s >)}$$

- What about the last tag?
  * Assume we have a symbol "</s>" that represents the end of sentence.

The Viterbi algorithm: Dynamic Programming, keep track of scores for each tag for "can" and check what would happen if "can" had a different tag 时间复杂度 $O(T^2N)$

因为independence assumptions才能apply DP，概率都很小实际中可加log

HMMs in practice: 实际中多用tri-gram，需要smooth

* $P(\boldsymbol{t}) = \prod_{i=1}^{n} P(t_i|t_{i-1}, t_{i-2})$ Viterbi now $O(T^3N)$

- Need to deal with sparsity: some tag trigram sequences might not be present in training data
  * Backoff: $P(t_i|t_{i-1}, t_{i-2}) = \lambda_3 \hat{P}(t_i|t_{i-1}, t_{i-2}) + \lambda_2 \hat{P}(t_i|t_{i-1}) + \lambda_1 \hat{P}(t_i)$
  * $\lambda_1 + \lambda_2 + \lambda_3 = 1$
  * Can learn the weights using **deleted interpolation**.

Main drawback: not very flexible in terms of feature representation, compared to MEMMs and CRFs.

Estimate π, A and B for POS tagging, based on the following tagged corpus: π是开头

```
1. silver-JJ wheels-NNS turn-VBP
2. wheels-NNS turn-VBP right-JJ
3. right-JJ wheels-NNS turn-VBP
```

• For π, there are three sentences: two begin with JJ and one with NNS

$\pi[JJ, NNS, VBP] = [\frac{2}{3}, \frac{1}{3}, 0]$ 即共三个<s>其中跟JJ的2个，跟NNS的1个

• For A, we need to observe the distribution for each. 在句末加</s>的话如下图 transition (A)

A	JJ	NNS	VBP	EOS
JJ	0	$\frac{2}{3}$	0	$\frac{1}{3}$
NNS	0	0	1	0
VBP	$\frac{1}{3}$	0	0	$\frac{2}{3}$

即JJ出现3次其中跟NNS的2次，VBP出现三次跟JJ的1次

• For B, we can simply read off the corresponding words for each tag in the corpus. emission (O)

B	right	silver	turn	wheels
(from) JJ	$\frac{2}{3}$	$\frac{1}{3}$	0	0
NNS	0	0	0	1
VBP	0	0	1	0

有3个JJ其中right有2次，siliver有1次

Consider using the following Hidden Markov Model to tag the sentence silver wheels turn:

$\pi[\text{JJ}, \text{NNS}, \text{VBP}] = [0.3, 0.4, 0.3]$

A	JJ	NNS	VBP	B	silver	wheels	turn
JJ	0.4	0.5	0.1	JJ	0.8	0.1	0.1
NNS	0.1	0.4	0.5	NNS	0.3	0.4	0.3
VBP	0.4	0.5	0.1	VBP	0.1	0.3	0.6

Step 1:

α	1:*silver*		2:*wheels*	3:*turn*
JJ:	JJ	$\pi[\text{JJ}]B[\text{JJ}, \text{silver}]$ $0.3 \times 0.8 = 0.24$		
NNS:	NNS	$\pi[\text{NNS}]B[\text{NNS}, \text{silver}]$ $0.4 \times 0.3 = 0.12$		
VBP:	VBP	$\pi[\text{VBP}]B[\text{VBP}, \text{silver}]$ $0.3 \times 0.1 = 0.03$		

Step 2:

α	1:*silver*	2:*wheels*		3:*turn*
JJ:	0.24	JJ → JJ 0.24	$A[\text{JJ,JJ}]B[\text{JJ}, \text{wheels}]$ $\times 0.4 \times 0.1 = \mathbf{0.0096}$	
		NNS → JJ 0.12	$A[\text{NNS,JJ}]B[\text{JJ}, \text{wheels}]$ $\times 0.1 \times 0.1 = 0.0012$	
		VBP → JJ 0.03	$A[\text{VBP,JJ}]B[\text{JJ}, \text{wheels}]$ $\times 0.4 \times 0.1 = 0.0012$	
NNS:	0.12	JJ → NNS 0.24	$A[\text{JJ,NNS}]B[\text{NNS}, \text{wheels}]$ $\times 0.5 \times 0.4 = \mathbf{0.048}$	
		NNS → NNS 0.12	$A[\text{NNS,NNS}]B[\text{NNS}, \text{wheels}]$ $\times 0.4 \times 0.4 = 0.0192$	
		VBP → NNS 0.03	$A[\text{VBP,NNS}]B[\text{NNS}, \text{wheels}]$ $\times 0.5 \times 0.4 = 0.006$	
VBP:	0.03	JJ → VBP 0.24	$A[\text{JJ,VBP}]B[\text{VBP}, \text{wheels}]$ $\times 0.1 \times 0.3 = 0.0072$	
		NNS → VBP 0.12	$A[\text{NNS,VBP}]B[\text{VBP}, \text{wheels}]$ $\times 0.5 \times 0.3 = \mathbf{0.018}$	
		VBP → VBP 0.03	$A[\text{VBP,VBP}]B[\text{VBP}, \text{wheels}]$ $\times 0.1 \times 0.3 = 0.0009$	

Step 3:

α	1:*silver*	2:*wheels*	3:*turn*	
JJ:	0.24	0.0096	JJ → JJ 0.0096	$A[\text{JJ,JJ}]B[\text{JJ}, \text{turn}]$ $\times 0.4 \times 0.1 = 0.000384$
		JJ → JJ	NNS → JJ 0.048	$A[\text{NNS,JJ}]B[\text{JJ}, \text{turn}]$ $\times 0.1 \times 0.1 = 0.00048$
			VBP → JJ 0.018	$A[\text{VBP,JJ}]B[\text{JJ}, \text{turn}]$ $\times 0.4 \times 0.1 = \mathbf{0.00072}$
NNS:	0.12	0.048	JJ → NNS 0.0096	$A[\text{JJ,NNS}]B[\text{NNS}, \text{turn}]$ $\times 0.5 \times 0.3 = 0.00144$
		JJ → NNS	NNS → NNS 0.048	$A[\text{NNS,NNS}]B[\text{NNS}, \text{turn}]$ $\times 0.4 \times 0.3 = \mathbf{0.00576}$
			VBP → NNS 0.018	$A[\text{VBP,NNS}]B[\text{NNS}, \text{turn}]$ $\times 0.5 \times 0.3 = 0.0027$
VBP:	0.03	0.018	JJ → VBP 0.0096	$A[\text{JJ,VBP}]B[\text{VBP}, \text{turn}]$ $\times 0.1 \times 0.6 = 0.000576$
		NNS → VBP	NNS → VBP 0.048	$A[\text{NNS,VBP}]B[\text{VBP}, \text{turn}]$ $\times 0.5 \times 0.6 = \mathbf{0.0144}$
			VBP → VBP 0.018	$A[\text{VBP,VBP}]B[\text{VBP}, \text{turn}]$ $\times 0.1 \times 0.6 = 0.00108$

Lecture 16. Formal Language Theory & Finite State Automata

What are regular grammar and regular language? How are they different?

• A language is a set of acceptable strings and a **grammar** is a generative生成的 description of a language.

• **Regular language** is a formal language形式语言 that can be expressed using a regular expression正则表达. 支持以下操作:

- Symbol drawn from alphabet, Σ
- Empty string, ε
- Concatenation of two regular expressions, RS
- Alternation of two regular expressions, R|S
- Kleene star for 0 or more repeats, R*
- Parenthesis () to define scope of operations 注意ε为空，Σ任何symbol，用()标识这些表达

比如 * *0(0|1)*1* 是0开头1结尾的只包含01的串

* *((what)|(where)|(why)|(which)|(whose)|(whom)) Σ* ?* 任何以wh-开头?结尾的句子

• **Regular grammar**正则文法 is a formal grammar形式文法 defined by a set of productions rules in the form of A → xB, A → x and A → ε, where A and B are non-terminals, x is a terminal and ε is the empty string. 用于检查一个string是不是合法的这种语言

• A language is regular if and only if it can be generated by a regular grammar.

- For example: A simple regular grammar
  - Rules: S → A, A → a A, A → ϵ
  - S is the start symbol
  - It will generate words such as a, aa, aaa, aaa.
  - The set of words generated by this regular grammar is a regular language.
  - This regular language can also be expressed in regular expression (a)*.

Transduction: 转导 input one string, output another 一种translation，比如把words转为tags

Regular languages are closed under union, intersection and concatenation. What does it mean? Why is it important? 经过这些操作以后的RLs还是RLs

Closure properties - 若对某个集合的成员进行一种运算，生成的仍然是这个集合的成员，则该集合被称为在这个运算下闭合

- This means that if L1 and L2 are two regular languages, then L1 ∪ L2, L1 ∩ L2, and the language strings that are the concatenation of L1 and L2 are also regular languages.
- This closure property allows us to apply operations on regular languages to produce a regular language.
- This allows for NLP problems to be factored into small simple parts, such that we can develop regular languages for each part, and combine them into a complex system to handle the NLP problems. This is particularly relevant for transducers and the composition operation, which are used in many NLP pipelines. (Note that FSTs implement "regular relations" rather that regular languages, but the distinction is a bit subtle and is not something we will dive into.)

把短regular languages结合成长的

Finite State Acceptors: 包含字典，状态集(其中q0是起始F是终点，可能有多个终点状态)

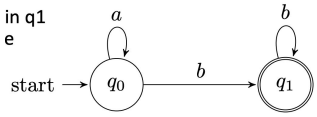
- alphabet of input symbols, Σ
- set of states, Q
- start state, $q_0 \in Q$
- final states, $F \subseteq Q$
- transition function
  symbol and state → next state 还有一些转换function

接受一个string的条件是从q0到一个F存在一条match这个string的path

Example: Input alphabet {a, b} States {q0, q1} Start, final states q0, {q1}

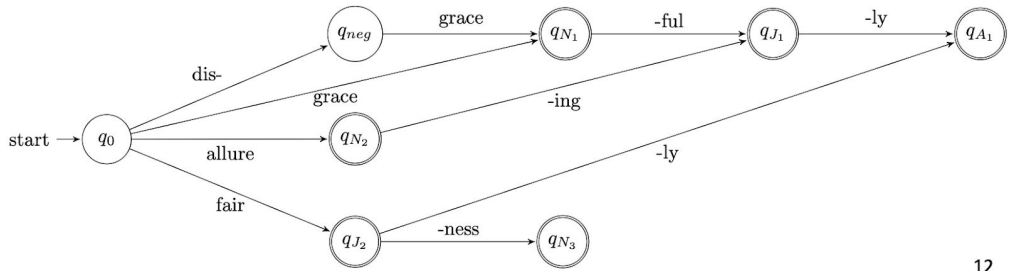
Transition function {(q0,a) → q0, (q0, b) → q1, (q1,b) → q1}

in q1
e



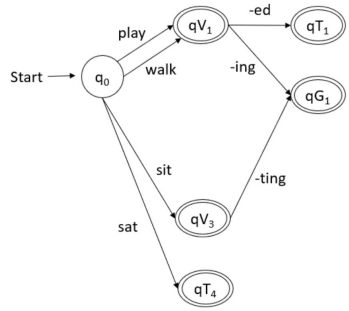
Accepts a*bb* 如在q1状态下见到b则匹配失败

FSA for word morphology: 双线的圈都是F



12

Draw a Finite State <u>Acceptor</u> (FSA) for word morphology to show the possible derivations from root forms using the words: play, played, playing; walk, walked, walking; sit, sat, sitting.



What are Weighted Finite State Acceptors (WFSAs)? When and why are they useful?

• WFSAs are generalizations of FSAs, with each path assigned a score, computed from the transitions, the initial state, and the final state. (给每条边一个score) The <u>total score for any path is equal to the sum of the scores of the initial state, the transitions, and the final state</u>.

• WFSAs can produce a score for each valid word for sub-word decomposition problems or sentence for words-in-sentence problems, while FSAs have no way to express preferences among technically valid words or sentences. 可用于处理错误的词，局部匹配

• For example, WFSAs can assign scores to all strings of characters forming words, so that spelling mistakes, new words, or strange-but-acceptable words can still be handled.

• The same argument holds for sequences of words forming sentences. Clearly some word sequences are gibberish, but being able to provide a numerical score can help in many applications, like how LMs can be used in sentence generation, speech recognition, OCR, translation, etc.

WFSA shortest-path: 总score就是每个t (path)的score和，Use shortest-path algorithm to find π with min. cost，O(V log V + E)，often seek to maximise probability, so set λ, ρ, δ to negative log probabilities and minimise

- Total score of a path $\pi = t_1, \ldots, t_N$ now

  $\lambda(t_0) + \sum_{i=1}^{N} \delta(t_i) + \rho(t_N)$

  each t is an edge, so more formally using from &/or to states and edge label in score calculation

很像Viterbi for HMMs，实际上Hidden Markov Model (HMM) is a transducer-style FSM

N-gram LMs: 实际上也是一种WFSA

- Recall LM calculates score of string as follows

  $P(w_1, w_2, .. w_M) = \prod_{i=1}^{M} P(w_i | w_{i-1})$ *(bigram)*

- Implemented as WSFA
  - Σ = set of word types
  - Q = Σ
  - $\lambda(q_i) = -\log P(w_1 = i | w_0 = \square)$
  - $\rho(q_i) = -\log P(w_{M+1} = \blacksquare | w_M = i)$
  - $\delta(q_i, w, q_j) = \begin{cases} -\log P(w_m = j \mid w_{m-1} = i) & \text{if } w = j \\ \infty & \text{otherwise} \end{cases}$

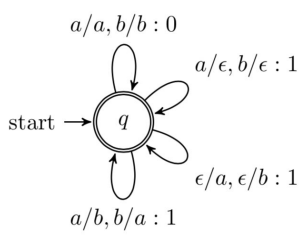
空方块是<s>，黑方块是</s>

Finite State Transducers: 不只只是accept或score，而是把string转换成另一种language

Edit distance automata: edit distance as WFST which takes one string, and outputs the other

用于计算Levenshtein distance的，完全一样cost 0，替换增加删除都cost 1

$$\delta(q, a, a, q) = \delta(q, b, b, q) = 0$$
$$\delta(q, a, b, q) = \delta(q, b, a, q) = 1$$
$$\delta(q, a, \epsilon, q) = \delta(q, b, \epsilon, q) = 1$$
$$\delta(q, \epsilon, a, q) = \delta(q, \epsilon, b, q) = 1.$$



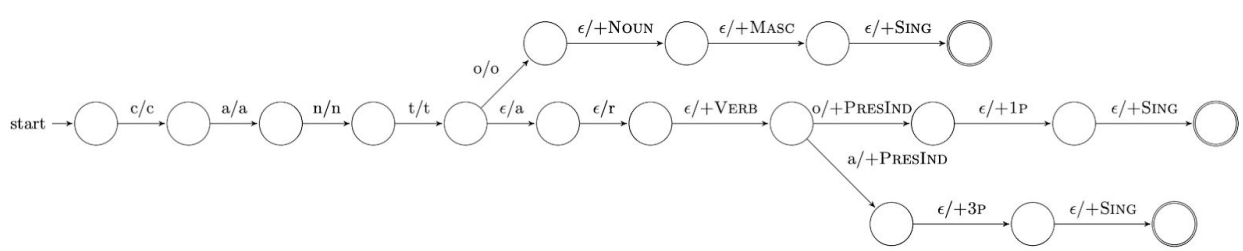
a/ε即原本是a替换成ε

FST for Spanish inflection: 应用于西班牙语的曲折形态转换，canto形态要对应不同人称主语

	cantar	to sing
1P singular	yo canto	I sing
2P singular	tu cantas	you sing
3P singular	ella canta	she sings
1P plural	nostotros cantamos	we sing
2P plural	vosotros cantáis	you sing
3P plural	ellas cantan	they sing

canto → *canto* +Noun +Masc +Sing

canto → *cantar* +Verb +PresInd +1P +Sign



Lecture 17. Context-free Grammars

Terminal: word such as book 实质内容

Non-terminal: syntactic label such as NP or NN 表示不同类型的label，通常用大小写或""区分

Productions (rules): W → X Y Z

Exactly one non-terminal on left-hand side (LHS) 左边的只能有一个

An ordered list of symbols on right-hand side (RHS) — can be Terminals or Non-terminals 右边的可以多个，non或者terminal都行

CFGs (CF grammars) vs regular grammars:

Regular grammars:

describe a <u>smaller</u> class of languages

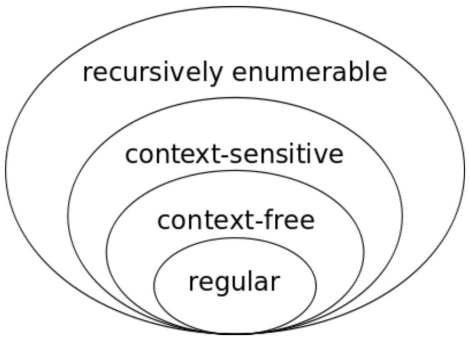
can be parsed using finite state machines (FSA, FST)

CFGs:

can describe hierarchical groupings (a richer class of languages)

requires more complex automata to parse (PDA)

Chomsky hierarchy:



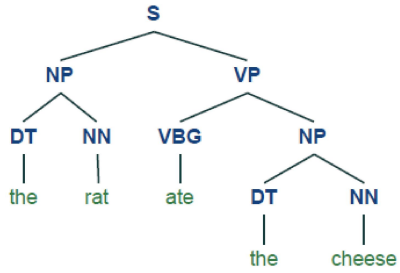
越外圈越需要fancier的算法，内是外的子集

CFG trees: Non-terminals are internal nodes, Terminals are leaves 由productions生成句子

Productions:

S → NP VP
NP → DT NN
VP → VBD NP
DT → *the*
NN → *rat*
NN → *cheese*
VBD → *ate*

由S开始替换



Parsing is the reverse process

Parsing CFGs: Parsing: given string, identify possible structures 由句子生成上面的树 (DP)

Brute force search is intractable

1. Bottom-up, Start with words, work up towards S (**CYK parsing**)
2. Top-down, Start with S, work down towards words, Earley parsing (not covered)

The CYK parsing algorithm:

1. Convert grammar to Chomsky Normal Form (CNF)
2. Fill in a parse table
3. Use table to derive parse
4. Convert result back to original grammar

Convert to Chomsky Normal Form: Change grammar so all rules of form A → B C or A → a

1. Convert rules of form A → B c into pair of rules A → B X, X → c 其中X是一个new label，不能terminal和non-terminal在同一级转换
2. Convert rules A → B C D into A → B Y, Y → C D 不能1个到3个换，从2个过渡
3. CNF disallows unary rules, A → B. 单元转换不可以，会形成环
4. 如果无环转换可传递

CYK algorithm: i是right most word，j是left most word，k是mid point (把区间从哪里分开)

```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
     for all {A | A → words[j] ∈ grammar}
           table[j − 1, j] ← table[j − 1, j] ∪ A
     for i ← from j − 2 downto 0 do
        for k ← i + 1 to j − 1 do
           for all {A | A → BC ∈ grammar and B ∈ table[i,k] and C ∈ table[k, j]}
                 table[i,j] ← table[i,j] ∪ A
```

Basic English Sentence structures:

- Declarative sentences (S → NP VP)
  - *The rat ate the cheese*
- Imperative sentences (S → VP)
  - *Eat the cheese!*
- Yes/no questions (S → VB NP VP)
  - *Did the rat eat the cheese?*
- *Wh*-subject-questions (S → WH VP)
  - *Who ate the cheese?*
- *Wh*-object-questions (S → WH VB NP VP)
  - *What did the rat eat?*

What is chart parsing? Why is it important?

• Parsing in general is the process of identifying the structure(s) of a sentence, according to a grammar of the language.

• In general, the search space is too large to do this efficiently, so we use a dynamic programming method to keep track of partial solutions. The data structure we use for this is a chart, where entries in the chart correspond to partial parses (licensed structures) for various spans (sequences of tokens) within the sentence.

Consider the following simple context–free grammar:

```
S -> NP VP
VP -> V NP | V NP PP
PP -> P NP
V -> "saw" | "walked"
NP -> "John" | "Bob" | Det N | Det N PP
Det -> "a" | "an" | "the" | "my"
N -> "man" | "cat" | "telescope" | "park"
P -> "on" | "by" | "with"
```

What changes need to be made to the grammar to make it suitable for CYK parsing?

– a (single) non-terminal which re-writes as a single terminal

– a (single) non-terminal which re-writes as exactly two non-terminals

```
VP -> V X
X -> NP PP
NP -> Det Y
Y -> N PP
```

Using the CYK strategy and the above grammar in CNF, parse the following sentences:

“an park by Bob walked an park with Bob”

“park by the cat with my telescope”

- This sentence has no parse, because the cell [0,7] doesn’t have an S.

an	*park*	*by*	*Bob*	*walked*	*an*	*park*	*with*	*Bob*
[0,1] Det	[0,2] NP	[0,3] -	[0,4] NP, X	[0,5] -	[0,6] -	[0,7] S	[0,8] -	[0,9] S, S
	[1,2] N	[1,3] -	[1,4] Y	[1,5] -	[1,6] -	[1,7] -	[1,8] -	[1,9] -
		[2,3] P	[2,4] PP	[2,5] -	[2,6] -	[2,7] -	[2,8] -	[2,9] -
			[3,4] NP	[3,5] -	[3,6] -	[3,7] S	[3,8] -	[3,9] S, S
				[4,5] V	[4,6] -	[4,7] VP	[4,8] -	[4,9] VP, VP
					[5,6] Det	[5,7] NP	[5,8] -	[5,9] NP, X
						[6,7] N	[6,8] -	[6,9] Y
							[7,8] P	[7,9] PP
								[8,9] NP

park	*by*	*the*	*cat*	*with*	*my*	*telescope*
[0,1] N	[0,2] -	[0,3] -	[0,4] Y	[0,5] -	[0,6] -	[0,7] Y
	[1,2] P	[1,3] -	[1,4] PP	[1,5] -	[1,6] -	[1,7] PP
		[2,3] Det	[2,4] NP	[2,5] -	[2,6] -	[2,7] NP, X
			[3,4] N	[3,5] -	[3,6] -	[3,7] Y
				[4,5] P	[4,6] -	[4,7] PP
					[5,6] Det	[5,7] NP
						[6,7] N

Lecture 18. Probabilistic Parsing

Basics of Probabilistic CFGs: 同样有terminal和non-terminal和productions

In addition, store a **<u>probability</u>** with each production

must be positive values, between 0 and 1

must **<u>sum to one</u>** for given LHS

What differentiates probabilistic parsing from chart parsing? Why is this important? How does this affect the algorithms used for parsing?

• Parsing in general is the process of identifying the structure(s) of a sentence, according to a grammar of the language.

• In general, the search space is too large to do this efficiently, so we use a dynamic programming method to keep track of partial solutions. The data structure we use for this is a **chart**, where entries in the chart correspond to partial parses (licensed structures) for various spans (sequences of tokens) within the sentence. **Probabilistic parsing** adds real-valued weights (probability) to each production in the grammar, such that parse trees can be assigned a score, namely the product of the probabilities of the productions in the tree. This is important as it allows for discrimination between likely and unlikely parses, rather than just provide a list of all parses, as in standard chart parsing. This affects the algorithms as they need to track the maximum probability analysis for each span, rather than the set of grammar symbols. However the parsing algorithms are very closely related.

What is a probabilistic grammar and what problem does it attempt to solve?

• A probabilistic grammar, as described above, adds real-valued weights (probability) to each production in the grammar. This attempts to provide a “language model”, that is, describe the likely sentences in a language, which is facilitated by their grammatical structure.

CYK for PCFGS:

- Convert grammar to Chomsky Normal Form (CNF)
  - E.g., VP → Verb NP NP [0.05]

    becomes VP → Verb NP+NP [____]
    NP+NP → NP NP [____]

    where NP+NP is a new symbol.

Prob. CYK:

```
function PROBABILISTIC-CKY(words,grammar) returns most probable parse
                                                  and its probability
  for j ← from 1 to LENGTH(words) do
     for all { A | A → words[j] ∈ grammar}
       table[j − 1, j, A] ← P(A → words[j])
     for i ← from j − 2 downto 0 do
        for k ← i + 1 to j − 1 do
              for all { A | A → BC ∈ grammar,
                              and table[i, k, B] > 0 and table[k, j, C] > 0 }
                    if (table[i,j,A] < P(A → BC) × table[i,k,B] × table[k,j,C]) then
                        table[i,j,A] ← P(A → BC) × table[i,k,B] × table[k,j,C]
                        back[i,j,A] ← {k, B, C}
     return BUILD_TREE(back[1, LENGTH(words), S]), table[1, LENGTH(words), S]
```

	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

2

S in the top-right corner of parse table indicates **<u>success</u>**

for each chart cell, **<u>store the split point</u>** and the nonterminal for the left and right children

To get parse(s), follow pointers back for each match

Problems with (P)CFGs:

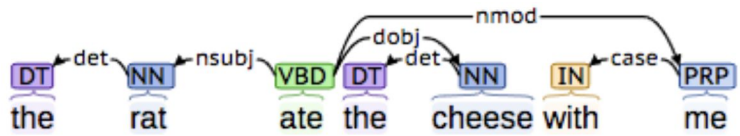
1. poor independence assumptions: inter-dependence is often needed to capture global structure. (NP → PRP used often as subject (first NP), much less often as object (second NP))
2. lack of lexical conditioning: non-terminals representation behaviour of the actual words, but are much too coarse

Lecture 19. Dependency Grammar & Parsing

Phrase-structure grammars assume a constituency tree which identifies the phrases in a sentence 之前那些grammar都是找interchangable的phrase

Dependency grammar: describe binary relations between pairs of words, namely, between heads and dependents依赖者

Links between a head word and its dependent words: syntactic句法 roles或modifier relations



比如me就modifier了ate这个动作

Several types of dependency:

1. predicate谓语: ate(rat, cheese)

• rat is the subject of verb ate (thing doing the eating) 主语
• cheese is the direct object of verb ate (thing being eaten) 宾语

2. modifier which is typically optional (aka adjunct)
• [(with) me] modifies the act of (the rat) eating 附加语

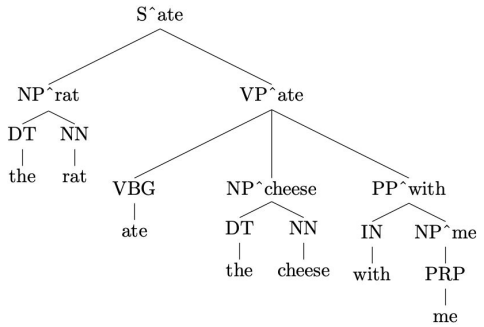
3. specifiers, e.g., <u>the</u> rat, <u>the</u> cheese, with me 定语
• help to specify the referent (which rat?), the head's relation, etc.

Dependency types:

* NSUBJ (nominal subject) — **Daniel** *speaks* Brazilian Portuguese
* DOBJ (direct object) — Trevor *presented* a **lecture** in English
* IOBJ (indirect object) — Morpheus *gave* **Neo** the red pill
* APPOS (appositive) — *Neo*, the main **character**, swallowed the pill

What can we do with dependency trees? use as features for other tasks, e.g., sentiment, relation extraction, QA, various semantic tasks.

Dependency vs head: Close similarity with 'head' in phrase-structure grammars
* the 'head' of an XP is (mostly) an X, i.e., noun in a NP, verb in a VP etc.
* main dependency edges captured in rewrite rules
• S^ate -> NP^rat VP^ate captures dependency rat ← ate



Dependency tree 和 Cf. phrase-structure grammars 区别:

Dependency edges from a tree: each node is a word token, one node is chosen as the root, directed edges link heads and their dependents

Cf. phrase-structure grammars: forms a **<u>hierarchical</u>** tree, word tokens are the **<u>leaves</u>** (而不是node), internal nodes are 'constituent phrases' e.g., NP, VP etc

相同点Both use part-of-speech

Projectivity: A tree is projective if, for all arcs from head to dependent: there is a path from the head to every word that lies between the head and the dependent 在plane上画没有any arcs crossing 没有交叉

Dependency grammar: 不是真正的grammar，any structure is valid，没有define a language

Dependency parsing: Parsing: task of finding the best structure for a given input sentence

Two main approaches:

1. graph-based: uses chart over possible parses, and dynamic programming to solve for the maximum 不考

2. **transition-based**: treats problem as incremental sequence of decisions over next action in a state machine

要维护的数据结构：

buffer = input words yet to be processed

stack = head words currently being processed

支持的操作：

shift = move word from buffer on to top of stack

arc = add arc (left/right) between top two items on stack 并从stack中删dependent

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:

如何知道每一步是shift还是arc:

- Use a scoring function, *score(buffer, stack, transition),* based on the state, i.e.,
  - the next word(s) in the buffer
  - the contents of the stack, particularly the top two items
  - the transition type, one of *{continue, arc-left, arc-right}*

greedy分数最大的

Transition based scoring:

- Form a feature representation for the state
- Example features, ϕ
  - [stack top has tag NN & next in stack has tag DT & transition = arc-left]
  - [stack top has tag NN & next in stack has tag DT & transition = arc-right]
  - [stack top has tag NN & next in stack is "has" & transition = arc-right]
  - [stack top has tag JJ & next in stack has tag DT & transition = shift]
- Have a *weight* for each feature, ***w***
  - such that the parser can choose between the possible transitions (e.g., arc-left, arc-right, shift)

定一些feature计算score

Training a Transition-based Dep Parser: 如何训练weights (Perceptron training)

uses an "oracle" sequence of parser actions, predict next action in sequence, and update when model disagrees with gold action

In what ways is (transition–based, probabilistic) dependency parsing similar to (probabilistic) CYK parsing? In what ways is it different?

– Both methods are attempting to determine the structure of a sentence; both methods are attempting to disambiguate amongst the (perhaps many)

possible structures licensed by the grammar by using a probabilistic grammar to determine the most probable structure.
– Both methods process the tokens in the sentence one–by–one, left–to–right.
– Although POS tags are implicitly used in constructing the “oracle” (training), the depedency parser doesn’t explicitly tag the sentence
– CYK adds numerous fragments to the chart, which don’t end up getting used in the final parse structure, whereas the transition–based dependency parser only adds edges that will be in the final structure.

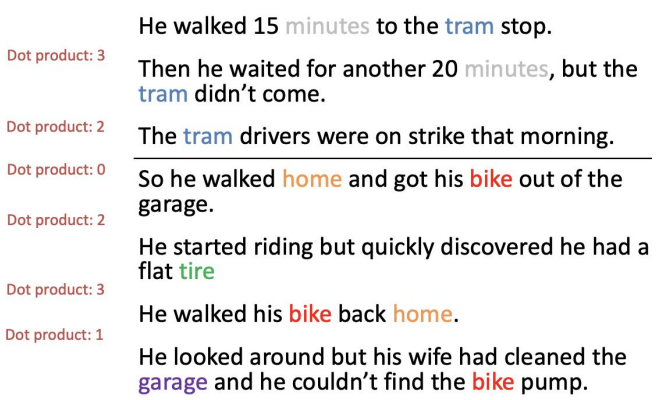
Lecture 20. Discourse

Discourse: a coherent, structured group of sentences **(utterances)**
Three Key discourse tasks: 1. Discourse segmentation把描述同一个话题的几句话group together不同的分开 2. Discourse parsing解析 3. Anaphor resolution找代词代指谁
Discourse segmentation: In Discourse Segmentation, we try to divide up a text into discrete, cohesive units based on sentences.

Assumption: text can be divided into a number of discrete, contiguous sections
Task: classifying whether a boundary exists between any two sentences 两句间是否能拆

1. **An unsupervised approach**: TextTiling algorithm: looking for points of low lexical cohesion
   - For each sentence gap:
     - 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

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

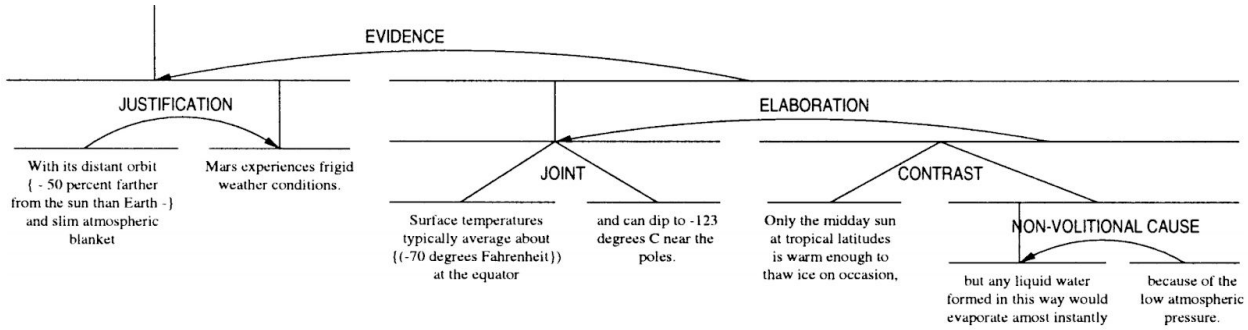


k=2

2. **Supervised discourse segmentation**:
   - Get data from easy sources (documents with paragraphs)
   - Apply a binary classifier to identify boundaries
   - Or use sequential classifiers
   - Integrate a wider range of features, including: *distributional semantics *coreference cues *discourse markers

Discourse parsing: Discourse units (DUs) are related by specific coherence relations. Two related DUs form a new DUs. All DUs in a coherent discourse must be related. A discourse will form a tree, which can be parsed. 把一整个Discourse分成几个Discourse unitis并用tree解析units间的关系

1. **An RST Tree**: RST= Rhetorical修辞 Structure Theory, 23 relations



2. **Parsing using discourse markers**: Some discourse markers (cue phrases) explicitly indicate relations (examples: although, but, for example, in other words, so, because, in conclusion) Can be used to build a simple rule-based parser 但是有些relations没有markers，也有些markers ambiguous可以用多种关系解释
3. **Full-text Discourse Parsing**: Most existing discourse parsers use a greedy approach
   - First, segment into elementary DUs (EDUs)
   - Classify all adjacent EDUs as being same DU or not
   - Combine most probable into a single DU
   - Repeat until entire text is a single DU

Anaphor: an anaphor is a linguistic expression that refers back to one or more elements in the text (generally preceding the anaphor) 代词 These tend to be pronouns (he, she) but can also be determiners (which, the, etc.). Anaphors have an **antecedent**前情 in the discourse

What is anaphora resolution and why is it difficult?

This is the problem of working out which element (generally a noun or noun phrase, but sometimes a whole clause) a given anaphor is actually referring to.

- For example:

  Mary gave John a cat for **his** birthday. (i) **She** is generous. (ii) **He** was surprised. (iii) **He** is fluffy.

  his [birthday] obviously refers to John; (i) (presumably) refers to *Mary*; (ii) (presumably) refers to *John*; and (iii) (presumably) refers to *[the] cat*.

What are some useful heuristics (or features) to help resolve anaphora?

• The most obvious heuristic is the recency近因 heuristic: given multiple possible referents (that are consistent in meaning with the anaphor), the mostly intended one is the one most recently used in the text. 最近的

• A better heuristic is that the most likely referent (consistent in meaning with the anaphor) is the focus of the discourse (the “center”).

Antecedent Restrictions: 1. Pronouns must agree in number with their antecedents 比如单复数him/they 2. Pronouns must agree in gender with their antecedents比如他/她 3. Pronouns whose antecendents are the subject of the same syntactic clause must be reflexive反身代词比如himself

Antecedent Preferences: The antecedent should satisfy the selectional preferences of the verb. 动词偏好，比如the garage不能ride

The Centering Algorithm:

Idea: at any given moment, discourse is focused on a single entity, the "center"

Goal: assign pronoun to antecedents in a manner which avoids "rough" shifts in the "center"

- U_n = utterance n
- $C_f(U_n)$ = list of entities of U_n, ordered by salience
- $C_b(U_n)$ = backward center of U_n, highest ranked entity $C_f(U_{n-1})$ that appears in $C_f(U_n)$
- $C_p(U_n)$ = preferred forward center of U_n, highest ranked entity of $C_f(U_n)$

Cp当前Cf中的highest，Cb上一个Cf中出现的highest且在这个Cf中也出现

1. John saw a Ford in the dealership

   $C_f(U_1)$ = [John, Ford, dealership]
   $C_p(U_1)$ = John
   $C_b(U_1)$ = None

2. He showed it to Bob

   $C_f(U_2)$ = [John, Ford, Bob]
   $C_p(U_2)$ = John
   $C_b(U_2)$ = John

3. He bought it

If he = John then	*If he = Bob then*	***Rough shift** of center:*
$C_f(U_3)$ = [John, Ford]	$C_f(U_3)$ = [Bob, Ford]	*change in Cb between utterances,* and new Cb different to Cp
$C_p(U_3)$ = John	$C_p(U_3)$ = Bob	
$C_b(U_3)$ = John	$C_b(U_3)$ = Ford	

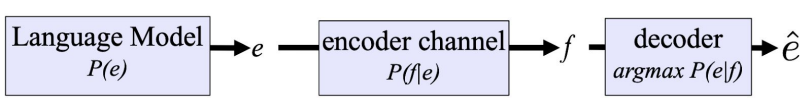
cb变了且cb!=cp叫rough shift

Supervised anaphor resolution:

- Build a binary classifier for anaphor/antecedent pairs
  - Use confidence values to optimize globally
- Convert restrictions and preferences into features
  - Binary features for number/gender compatibility
  - Include features about type of antecedent
- With enough data, can approximate the centering algorithm
- But also easy to include features which indicate tendencies, rather than rules
  - Like repetition, parallelism

Lecture 21. Machine translation: word-based models

Statistical MT: Assume that we started with an English sentence. The sentence was then corrupted by translation into French. We want to recover the original.



Noisy channel MT: $P(e|f) = \frac{P(e)P(f|e)}{P(f)}$ 给定f最好的e是什么

Translation Model (TM)

$\hat{e} = \text{argmax}_e \, P(e)P(f|e)$

Language Model (LM)

- P(f|e) rewards good translations, but permissive of disfluent e
- P(e) rewards e which look like fluent English, and helps put words in the correct order

• **How to learn the LM and TM**

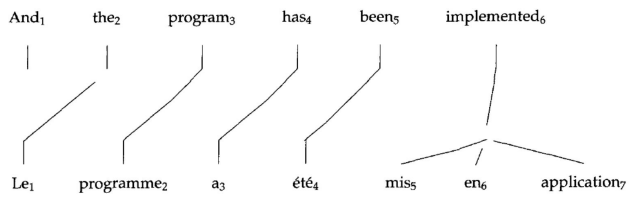
* LM: based on text frequencies in large monolingual corpora 纯英语，之前课中讲的
* TM: based on word co-occurrences in parallel texts 平行文本

What aspects of human language make automatic translation difficult?

The whole gamut of linguistics, from lexical complexity, morphology, syntax, semantics etc. In particular if the two languages have very different word forms, or very different syntax, leading to different word order. These raise difficult learning problems for a translation system, which needs to capture these differences in order to learn translations from bi-texts, and produce these for test examples.

Representing Alignment: 一些词可能没对应，一些可能一词对多词，反过来对齐不对称

E = $e_1 \ldots e_I$ = *And the program has been implemented*
F = $f_1 \ldots f_J$ = *Le programme a ete mis en application*
A = $a_1 \ldots a_J$ = 2, 3, 4, 5, 6, 6, 6.



如果反过来And就要对应0 (null)

IBM model 1: Formulate probabilistic model of translation

$$P(\mathbf{f}, \mathbf{a}|\mathbf{e}) = \frac{\epsilon}{(I+1)^J} \prod_{j=1}^{J} t(f_j|e_{a_j})$$

a是上面那个对齐的index list

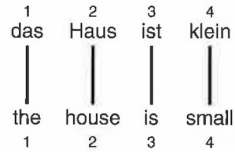
* *t(f|e)* are translation probabilities
* alignments a_j indexes the translation of word *j*

Example IBM:

e = the		e = house		e = is		e = small	
f	**t(f|e)**	**f**	**t(f|e)**	**f**	**t(f|e)**	**f**	**t(f|e)**
das	0.4	Haus	0.35	ist	0.2	klein	0.4
der	0.35	Geschlect	0.05	bin	0.15	gering	0.25
die	0.25	Häuser	0.15	bist	0.10	schmal	0.15
		aufnehmen	0.20	sein	0.30		
		Heim	0.25	sind	0.25		

$$P(\mathbf{f}, \mathbf{a}|\mathbf{e}) = \frac{\epsilon}{5^4} t(\text{das}|\text{the}) t(\text{Haus}|\text{house}) t(\text{ist}|\text{is}) t(\text{klein}|\text{small})$$
$$= 0.00029\epsilon$$

Example adapted from Koehn 09



如何得到alignment: "**expectation maximization**" (EM) algorithm

1. make initial guess of *t* parameters, e.g., uniform
2. initialise counts, *c*, of translation pairs to 0
3. for each sentence pair, (E, F)
   - for each position *j*, and value of $a_j \in \{0, 1, 2, \ldots, l\}$
     - compute *expected alignment*

$$P(a_j|\mathbf{e},\mathbf{f}) = \frac{t(f_j|e_{a_j})}{\sum_{a_j} t(f_j|e_{a_j})}$$

     - update fractional counts
       $c(e_j, f_{aj}) \leftarrow c(e_j, f_{aj}) + P(a_j|\mathbf{e}, \mathbf{f})$
4. update *t* with normalised counts, $t(f|e) = c(e, f) / c(e)$
5. repeat from step 2

Modelling limitations:

* ignores the positions of words in both strings
* limited to word-based phenomena
* asymmetric, can't handle 1:many or many:many
* difficulty learning from sparse data

For the following "bi-text":

Language A	Language B
`green house`	`casa verde`
`the house`	`la casa`

What is the logic behind IBM Model 1 for deriving word alignments?

• The core idea is that we are going to have a translation table which stores the probability of translating a word from the target language into every possible word in the source language (again, this is the wrong direction due to the noisy channel model). 用目标语言对齐初始语言

• The probability of a sentence can then be treated as a uni-gram probability, conditioned on how the tokens in the two sentences are aligned. Or, essentially, the product of all of the corresponding probabilities from the translation table.

Work through the first few iterations of using the Expectation Maximisation algorithm to build a translation table for this collection.

• We need to establish the direction of translation before we begin (although it isn't important in this particular example): let's say, we're trying to translate language B into language A. Consequently, we want the alignments where we're translating A into B.

• We're going to initialise our translation table T with uniform values: every word from A is equally likely to be translated as every word from B:

T	casa	la	verde	Total
green	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	1
house	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	1
the	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	1

- Consequently, each of the two sentences (called I and II below) has four (2^2) possible alignments (where every token of B is accounted for), namely:
  - Ia: `green` aligns with `casa` and `house` aligns with `verde`
  - Ib: `green` aligns with `verde` and `house` aligns with `casa`
  - Ic: `green` aligns with `casa` and `verde` (`house` implicitly aligns to null)
  - Id: `house` aligns with `casa` and `verde`
  - IIa: `the` aligns with `la` and `house` aligns with `casa`
  - IIb: `the` aligns with `casa` and `house` aligns with `la`
  - IIc: `the` aligns with `la` and `casa`
  - IId: `house` aligns with `la` and `casa`

- Now, we're going to calculate the **expected** likelihood of each of these possible alignments, according to the following formula:

$$\hat{P}(F, A|E) = \frac{\epsilon}{(I+1)^J} \prod_{j=1}^{J} t(f_j|e_{a_j})$$

- A close inspection might lead us to say that the $(+1)$ should be excluded, because we're neglecting the null term from the A tokens, but it doesn't actually matter, as we'll see in a moment.
- For Ia, we observe the following:

$$\begin{aligned}\hat{P}(F, A|E) &= \frac{\epsilon}{(I+1)^J} t(\texttt{casa}|\texttt{green}) t(\texttt{verde}|\texttt{house}) \\ &= \frac{\epsilon}{(2+1)^2}(\frac{1}{3})(\frac{1}{3}) = \frac{\epsilon}{9}\frac{1}{9}\end{aligned}$$

• Because our translation table is uniform, every calculation will look the same.

• Now, we're going to make a maximum likelihood estimate of each entry in our translation table. We do this by summing the expected probability of the alignment for each possible translation.

- For `green`:
  - It aligns with `casa` in Ia ($\frac{\epsilon}{9}\frac{1}{9}$) and Ic (same), to give a total of $\frac{\epsilon}{9}\frac{2}{9}$.
  - It aligns with `verde` in Ib ($\frac{\epsilon}{9}\frac{1}{9}$) and Ic (same), to give a total of $\frac{\epsilon}{9}\frac{2}{9}$.
  - It never aligns with `la`, because they don't appear in a sentence together.

T	casa	la	verde	Total
green	$\frac{\epsilon}{9}\frac{2}{9}$	0	$\frac{\epsilon}{9}\frac{2}{9}$	$\frac{\epsilon}{9}\frac{4}{9}$
house	$\frac{\epsilon}{9}\frac{4}{9}$	$\frac{\epsilon}{9}\frac{2}{9}$	$\frac{\epsilon}{9}\frac{2}{9}$	$\frac{\epsilon}{9}\frac{8}{9}$
the	$\frac{\epsilon}{9}\frac{2}{9}$	$\frac{\epsilon}{9}\frac{2}{9}$	0	$\frac{\epsilon}{9}\frac{4}{9}$

- We will now normalise the rows so that they look like probabilities. Doing this causes all of the $\frac{\epsilon}{9}$ terms to vanish; consequently, we will just ignore them for the rest of the steps below.
- After simplifying, here is the new translation table:

T	casa	la	verde	Total
green	$\frac{1}{2}$	0	$\frac{1}{2}$	1
house	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	1
the	$\frac{1}{2}$	$\frac{1}{2}$	0	1

- At this point, it perhaps isn't obvious that this table will give us better alignment estimates, but it does:
- For Ia, we observe the following (ignoring the ϵ term):

$$\begin{aligned}\hat{P}(F, A|E) &= t(\texttt{casa}|\texttt{green}) t(\texttt{verde}|\texttt{house}) \\ &= (\frac{1}{2})(\frac{1}{4}) = \frac{1}{8}\end{aligned}$$

- For Ib:

$$\begin{aligned}\hat{P}(F, A|E) &= t(\texttt{verde}|\texttt{green}) t(\texttt{casa}|\texttt{house}) \\ &= (\frac{1}{2})(\frac{1}{2}) = \frac{1}{4}\end{aligned}$$

- For Ic:

$$\begin{aligned}\hat{P}(F, A|E) &= t(\texttt{casa}|\texttt{green}) t(\texttt{verde}|\texttt{green}) \\ &= (\frac{1}{2})(\frac{1}{2}) = \frac{1}{4}\end{aligned}$$

- For Id:

$$\begin{aligned}\hat{P}(F, A|E) &= t(\texttt{casa}|\texttt{house}) t(\texttt{verde}|\texttt{house}) \\ &= (\frac{1}{2})(\frac{1}{4}) = \frac{1}{8}\end{aligned}$$

- The calculations for II are similar.
- Updating the alignment counts for `green` gives us:
  - It aligns with `casa` in Ia ($\frac{1}{8}$) and Ic ($\frac{1}{4}$), to give a total of $\frac{3}{8}$.
  - It aligns with `verde` in Ib ($\frac{1}{4}$) and Ic (same), to give a total of $\frac{1}{2}$.
  - It never aligns with `la`.

• Our un-normalised counts (neglecting the ε terms) are now:

T	casa	la	verde	Total
green	$\frac{3}{8}$	0	$\frac{1}{2}$	$\frac{7}{8}$
house	$\frac{3}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{5}{4}$
the	$\frac{3}{8}$	$\frac{1}{2}$	0	$\frac{7}{8}$

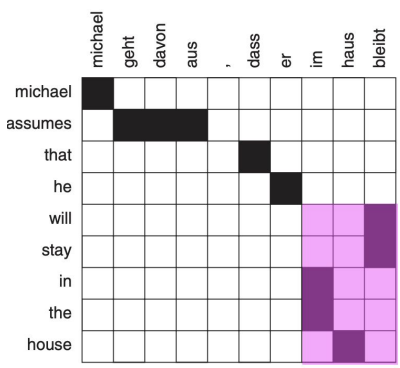
T	casa	la	verde	Total
green	$\frac{3}{7}$	0	$\frac{4}{7}$	1
house	$\frac{3}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	1
the	$\frac{3}{7}$	$\frac{4}{7}$	0	1

- We can see that we have correctly observed that `green` is most likely to be `verde`, `house` to be `casa`, and `the` to be `la`. Summarising the normalised probabilities:
- Further iterations will continue to improve these counts, and to observe that Ib and IIa are the most likely alignments.

Lecture 22. MT: phrase based & Neural Encoderdecoder

Finding & scoring phrase pairs: "Extract" phrase pairs as contiguous chunks in word aligned text; then

* compute counts over the whole corpus
* normalise counts to produce 'probabilities'



$$\phi(\text{im haus bleibt}|\text{will stay in the house}) = \frac{c(\text{will stay in the house; im haus bleibt})}{c(\text{will stay in the house})}$$

Phrase extraction:

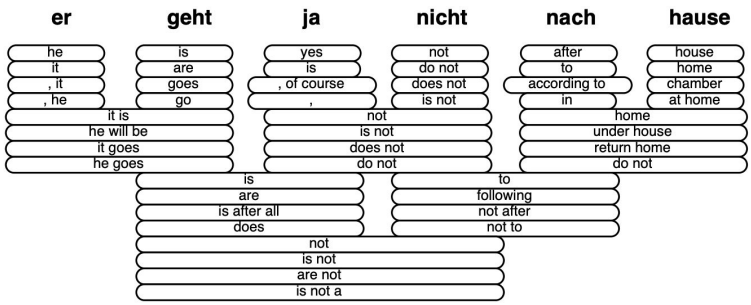
michael – michael
michael assumes – michael geht davon aus ; michael geht davon aus ,
michael assumes that – michael geht davon aus , dass
michael assumes that he – michael geht davon aus , dass er
michael assumes that he will stay in the house
– michael geht davon aus , dass er im haus bleibt
assumes – geht davon aus ; geht davon aus ,
assumes that – geht davon aus , dass
assumes that he – geht davon aus , dass er
assumes that he will stay in the house
– geht davon aus , dass er im haus bleibt
that – dass ; , dass
that he – dass er ; , dass er
that he will stay in the house
– dass er im haus bleibt ; , dass er im haus bleibt
he – er
he will stay in the house – er im haus bleibt
will stay – bleibt
will stay in the house – im haus bleibt
in the – im
in the house – im haus
house – haus

Decoding: A describes the segmentation of F into phrases, and the reordering of their translations to produce E

$$E^*, A^* = \text{argmax}_{E,A}\, \text{score}(E, A, F)$$

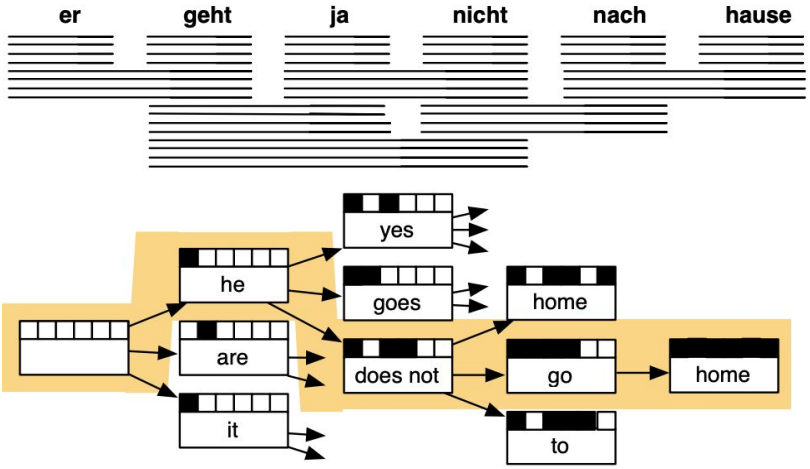
- The *score* function is a product of the
  - translation "probability", *P(F|E)*, split into phrase-pairs
  - language model probability, *P(E)*, over full sentence E
  - distortion model score, $d(start_i, end_{i-1})$, measuring amount of reordering (*minimised*) between adjacent phrase-pairs
- Search problem
  - find translation E* with the best overall score

Search problem: Cover all source words exactly once; visited in any order; and with any segmentation into "phrases"; Choose a translation from phrase-table options



选不重叠的方案

Phrase-based Decoding:



Complexity: NP, Use beam search, a form of approximate search

Phrase-based MT summary:

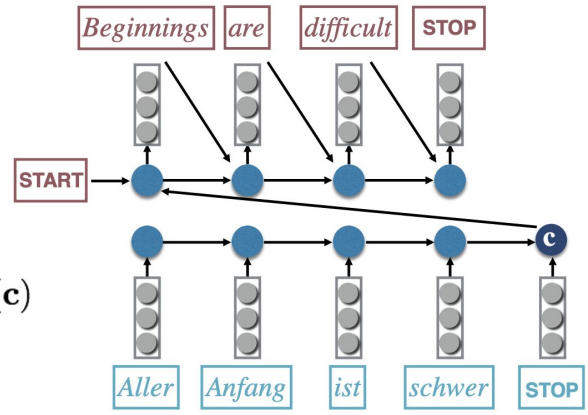
- Start with sentence-aligned parallel text
  1. learn word alignments
  2. extract phrase-pairs from word alignments & normalise counts
  3. learn a language model
- Now decode test sentences using beam-search
- State-of-the-art until 2013 *(see Moses toolkit)*

Neural Machine Translation: Two components: encoder and decoder

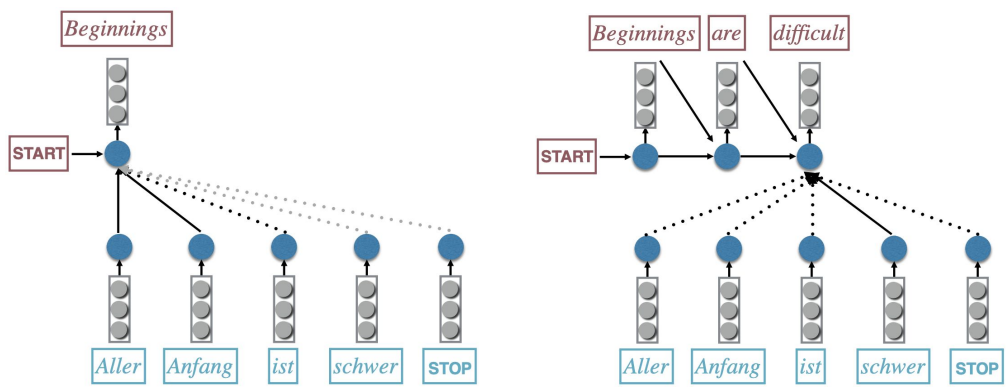
- encoder which represents the source sentence as a vector or matrix of real values
  - akin to word2vec's method for learning word vectors
- decoder which predicts the word sequence in the target
  - framed as a language model

1. **RNN Encoder-Decoders**

$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$
$$\boldsymbol{y} \mid \mathbf{c} \sim \mathrm{RNNLM}(\mathbf{c})$$



2. **RNN Attention Model**



Attention: Framed as a kind of similarity function between encoder state at word i and decoder state at word j

- learn function to take both states, and return scalar (simplest is to take the dot product)
- normalise numbers in a softmax for a given j ("**attention**")
- use attention to reweight encoder states to define *c*

• Context "c" now dynamic, varies during decoding

Evaluation: 两方面fluency和adequacy 找一些有正确翻译的references

MT Evaluation: BLEU: closeness of translation to one or more references

- defined as: $$\text{Bleu} = \text{BP} \times \exp\left(\frac{1}{N}\sum_{n=1}^{N} \log p_n\right)$$

  a weighted average of 1 to 4-gram precisions
  - p_n = num *n*-grams correct / num *n*-grams predicted in output
  - numerator clipped to #occurrences of *n*gram in the reference
- and a brevity penalty to hedge against short outputs
  - bp = min (1, output length / reference length)