Text Processing

- Example
  - *text* = "I slept and then I dreamed"
  - *tokens* = I, slept, and, then, I, dreamed
  - *types* = I, slept, and, then, dreamed
  - *terms* = sleep, dream (assuming stopword removal).

Example: "The dog and the cat ate dinner and played"

- Tokens = ["The", "dog", "and", "the", "cat", "ate", "dinner", "and", "played"]
- Types = ["The", "dog", "and", "cat", "ate", "dinner", "played"]
- Terms = ["dog", "cat", eat", "dinner", "play"]

- Whitespaces in proper names or collocations:
  - San Francisco => San_Francisco
    - how do we determine it should be a single token?
- Apostrophes are ambiguous:
  - possessive constructions:
    - the book's cover => the book s cover
  - contractions:
    - he's happy => he is happy
    - aren't => are not
  - quotations:
    - 'let it be' => let it be

The Apo
Society
John Ric

- Hyphenations:
  - co-education => co-education
  - state-of-the-art => state of the art? state_of_the_art?
  - lowercase, lower-case, lower case => lower_case
  - Hewlett-Packard => Hewlett_Packard? Hewlett Packard?
- Period
  - Abbreviations: Mr., Dr.
  - Acronyms: U.S.A.
  - File names: a.out

- Numbers
  - 3/12/91
  - Mar. 12, 1991
  - 55 B.C.
  - B-52
  - 100.2.86.144
- Unusual strings that should be recognized as tokens:
  - C++, C#, B-52, C4.5,M*A*S*H.

Usually stop words are:

- Function words: "a", "the", "in", "to", etc...
- Pronouns: "he", "she", "I", etc...
- High frequency words

["Her", "cat", "is", "in", "the", "hat"] → ["cat", "hat"]

- Tokenizing HTML
  - Should text in HTML tags not typically seen by the user be included as tokens?
    - Words appearing in URLs.
    - Words appearing in "meta text" of images.
  - Simplest approach is to exclude all HTML tag information (between "<" and ">") from tokenization.

Usually, Normalization = reducing tokens to same canonical term

- U.S.A. v.s. USA
- Rèsumè v.s. Resume
- colour v.s. color
- anti-matter vs antimatter

Example:

- democrat, democracy, democratic
- organize, organizes, organizing

Stemming is to reduce tokens to “root” form of words. Different stemmers may yield different results.

- **comput**ational, **comput**er, **comput**e
- **compr**essed, **compr**ession, **compr**ess

Common Errors of PorterStemmer

- Can produce unusual stems that are not English words:
  - “computer”, “computational”, “computation” all reduced to same token “comput”
- Commision: words with different meanings are stemmed to the same form
  - organization, organ → organ
- Omission: words with similar meanings result in different stems
  - creation → create
  - create → creat

Lemmatization

- Reduce inflectional/variant forms to base form
- Uses vocabulary and full morphological analysis to find the base word
- Need grammatical rules and it handles irregular words

Example:

- am, are, is → be
- car, cars, car's, cars' → car

Language identity.

$P(W_n|W_{n-1}) = [C(W_{n-1}, W_n) + 1]/[C(W_{n-1}) + V]$

- $C(W_{n-1}, W_n)$ = Frequency count of W_{n-1}, W_n in training data of language.
- $C(W_{n-1})$ = Frequency count of W_{n-1} in training data of language
- V = vocabulary, i.e., total number of unique words (or characters) in the training data
- A tip: use log(P) to avoid very small numbers

Exercise 4

$P(W_n|W_{n-1}) = [C(W_{n-1}, W_n) + 1]/[C(W_{n-1}) + V]$

ainsi de la même manière que l'on peut dire que le Royaume-Uni est un pays, on peut dire que l'Angleterre est un pays.

C(<start>) = 1

P(p| <start>) = [0 + 1]/[1 + 27] = 0.03571428571

P(a|p) = [2 + 1]/[4 + 27] = 0.09677419354

P(y|a) = [2 + 1]/[7 + 27] = 0.08823529411

P(s|y) = [2 + 1]/[3 + 27] = 0.1

Thus, P(pays_French) = 0.00003049606

Exercise 4

$P(W_n|W_{n-1}) = [C(W_{n-1}, W_n) + 1]/[C(W_{n-1}) + V]$

the same way that we can say that United Kingdom is a country, we can also say that England is a country.

C(<start>) = 1

p(p| <start>) = [0 + 1]/[1 + 23] = 0.04166666666

p(a|p) = [0 + 1]/[0 + 23] = 0.04347826086

p(y|a) = [3 + 1]/[13 + 23] = 0.11111111111

p(s|y) = [0 + 1]/[5 + 23] = 0.03571428571

Thus, P(pays_English) = 0.00000718886

0.00003049606 > 0.00000718886, so the prediction is that 'pays' belongs to French

- By Zipf, a word appearing f times has rank $r_f = AN/f$
- Several words may occur f times, assume rank r_f applies to the last of these.
- Therefore, r_f words occur f or more times and r_{f+1} words occur $f+1$ or more times.
- So, the number of words appearing **exactly** f times is:

$$I_f = r_f - r_{f+1} = \frac{AN}{f} - \frac{AN}{f+1} = \frac{AN}{f(f+1)}$$

- Assume highest ranking term occurs once and therefore has rank $V = AN/1$
- Fraction of words with frequency f is:

$$\frac{I_f}{V} = \frac{1}{f(f+1)}$$

- Fraction of words appearing only once is therefore ½.

- Assuming Zipf's Law with a corpus constant A=0.1, what is the fewest number of most common words that together account for more than 25% of word occurrences (i.e. the minimum value of m such as at least 25% of word occurrences are one of the m most common words)

$f = \frac{AN}{r}$ $\quad p_r = \frac{f}{N} = \frac{A}{r}$ for corpus const. A

$\frac{AN}{1} + \frac{AN}{2} + \cdots \frac{AN}{n} = 0.25\,N$

$n = 7$

Zip's law → how often a word comes up.

- models the distribution of terms in a corpus
- $p_r = f / N = A / r$, for corpus constant A
  - p_r - probability of the word, at rank r, occurring in the corpus
  - r - rank, numerical position of a word in list sorted by decreasing frequency (f) of occurrence in corpus
  - f - frequency of occurrence of word in corpus
  - N - total number of word occurrences (# of words in corpus)

$$\frac{\text{count}}{\text{total}} = \frac{A}{\text{rank}}$$

- If a corpus has 500,000 total words and the corpus constant is 0.1, approximately how many times does the 5th most frequent word appear in the corpus?
- Reminder: $p_r = f / N = A / r$ for corpus constant A

$$\frac{t}{N} = \frac{A}{r} \qquad t = \frac{AN}{r} = \frac{0.1 \cdot 500{,}000}{5}$$

$$= 10{,}000$$

Heaps Law → total number of words relate with unique words.

Heaps' Law

- models the number of words in the vocabulary as a function of the corpus size
- $V = Kn^B$
  - V = size of vocabulary, aka # of unique words in the corpus
  - n = length of corpus in words, aka # of total words
  - K, B are constants

Exercise

- For 10000 words, there are 4000 unique words: $4000 = K10000^B$
- For 50000 words, there are 9000 unique words: $9000 = K50000^B$

$$K = \frac{9000}{50000^B}$$

log(4000) = log(k) + B*log(10000)

log(9000) = log(k) + B*log(50000)

→ k ≈ 40 → plug this back into one of the above eqs and solve for B, B ≈ .5

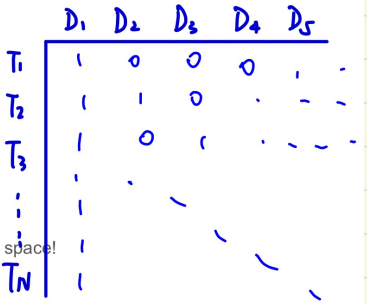
→ now with k and B, can solve for V: V = KN^B = 40 * (100000)^.5 ≈ 12,649

$$V = Kn^B \qquad \frac{9000}{4000} = \frac{k}{k}\left(\frac{50000}{10000}\right)^B$$

Boolean Retrieval

Boolean Retrieval

- desired information (**information need**): Which plays by Shakespeare mention kings but not Henry?
- **boolean query**: kings AND NOT Henry
- **Search procedure**
- term-document incidence matrix
  - one axis has terms, one axis has documents
  - 1 if document contains word, 0 otherwise
  - Scalability issues: 99.8% entries are 0s!
- inverted index
  - store only non-zero entries of term-doc matrix -> save space!
  - map terms to list of documents containing it



TN D. D1 D3 . . . .

Inverted Index

insect	→	2	4	8	16	32	64	128	
leaf	→	1	2	3	5	8	13	16	34
stick	→	13	16						

Word -> a list of document id

Document id should be sorted.

OR AND

when use or / and

It is much easier to do operator

Document frequency can be added too

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

AND merge

OR

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

ADD (ans. docID(p1))

ADD (ans. docID(p2))

if p1 != NIL add remaining element ⇒ same as p2.

Extensions to the Boolean model

- Phrase queries
  - Biword/bigram indexing
    - "computer science and engineering" – *computer science*, *science and, and engineeri*
  - Positional indexing
    - Extract indexes of documents and positions for all tokens in the query
    - computer: 2: 1, 16, 23; **3: 2,** 13, 14; 4: 3, 6, 15; …
    - science: 1: 4, 18, 21; **3: 3,** 20, 34; 4: 7, 19, 30; ...
    - Query Processing: merge algorithm (position level & document level)
  - What are some problems with boolean retrieval?
  - Combined Strategies

- Proximity Queries
  - Altavista, Google, many commercial search engines

Rank Retrieve

score between match and documents

- Returns an **ordering** over the (**top**) documents in the collections for a **query**
- The user's query is **free text**
- The large result set won't overwhelm the users
- Use **scoring** as the basis

help order

① Jaccard coefficien

Assume the following query:

- Query: *march of dimes*

And the following two documents:

- Document 1: *caesar died in march*
- Document 2: *the long march*

$|Q \cap D_1| = 1$

$|Q \cup D_1| = 6$

$J(Q, D_1) = \frac{1}{6}$

$J(Q, D_2) \frac{1}{5}$

- It does not consider *term frequency* (how many times a term occurs in a document)
- Rare terms in a collection are more informative than frequent terms. Jaccard does not consider this information
- We need a more sophisticated way of normalizing for length

- df_t is the document frequency of t: the number of documents that contain t
  - df_t is an inverse measure of the informativeness of t
- N is the total number of documents in the collection
  - $df_t \leq N$
- We define the idf (inverse document frequency) of t by

$$idf_t = \log_{10}(N/df_t)$$

  - We use log (N/df_t) instead of N/df_t to "dampen" the effect of idf.

- The term frequency $tf_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.
- More frequent terms in a document are more important, i.e. more indicative of the topic.
- May want to normalize *term frequency* (*tf*) :

$$tf_{t,d} = f_{t,d} / max\{f_{t,d}\}$$

- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

②

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$w_{t,d} = \mathrm{tf}_{t,d} \times \log_{10}(N / \mathrm{df}_t)$$

- Best known weighting scheme in information retrieval
  - Theoretically proven to work well
  - Note: the "-" in tf-idf is a hyphen, not a minus sign!
  - Alternative names: tf.idf, tf x idf
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Given a document containing terms with given frequencies:

A(3), B(2), C(1)

Assume collection contains 10,000 documents and document frequencies of these terms are:

A(50), B(2500), C(100)

Then (assuming $\log_{10}$)

A: tf = 3/3; idf = log(10000/50) = 2.3; tf-idf = 2.3

B: tf = 2/3; idf = log(10000/2500) = 0.60; tf-idf = 0.40

C: tf = 1/3; idf = log(10000/100) = 2.0; tf-idf = 0.66

③ Similarity.

$$EuclideanDist(X,Y) = \sqrt{\sum(x_i - y_i)^2}$$

$$sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\left|\vec{d}_j\right|\left|\vec{d}_k\right|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2}\sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$$

$$ManhDist(X,Y) = \sum_{i=1}^{n} | x_i - y_i |$$

- Cosine of angle between two vectors
- The denominator involves the lengths of the vectors
- So the cosine measure is also known as the *normalized inner product*

$$\mathrm{sim}(d_j, q) = d_j \bullet q = \sum_{i=1}^{t} w_{ij} \cdot w_{iq}$$

$$\text{Length } \left|\vec{d}_j\right| = \sqrt{\sum_{i=1}^{n} w_{i,j}^2}$$

$D_1 = 2T_1 + 3T_2 + 5T_3$ $\quad \mathrm{CosSim}(D_1, Q) = 10 / \sqrt{(4+9+25)(0+0+4)} = 0.81$

$D_2 = 3T_1 + 7T_2 + 1T_3$ $\quad \mathrm{CosSim}(D_2, Q) = 2 / \sqrt{(9+49+1)(0+0+4)} = 0.13$

$Q = 0T_1 + 0T_2 + 2T_3$

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

- cos(SaS,PaP) ≈ 0.789 × 0.832 + 0.515 × 0.555 + 0.335 × 0.0 + 0.0 × 0.0 ≈ 0.94
- cos(SaS,WH) ≈ 0.79
- cos(PaP,WH) ≈ 0.69

Why do we have cos(SaS,PaP) > cos(SaS,WH)?

- **Doc1**: apple orange banana peach
- **Doc2**: orange orange apple apple
- **Doc3**: banana tangerine peach
- **Doc4**: peach peach apple banana

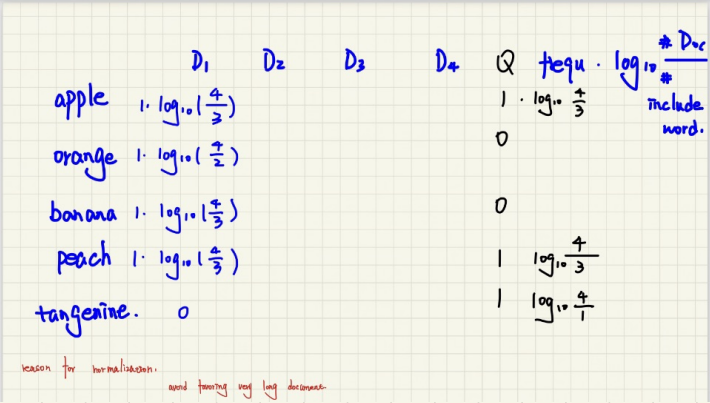
And the query:

- **Query**: apple peach tangerine

Assume no pre-processing (i.e., no normalization, no case-folding, no stopword-removal, no stemming), and a tfidf weighting scheme (tf not normalized, log based 10 for idf).

1. Write the vector representations for each of the four documents and for the query.

2. Determine the cosine similarities between each document and the query, and rank the documents.



Evaluation of IR model

- Precision
  - $\frac{retrieved \cap relevant}{retrieved}$
- Recall
  - $\frac{retrieved \cap relevant}{relevant}$
- Relevant documents:
  - cat cat cat dog (doc4)
  - fish cat dog bird (doc3)
  - cat cat dog (not present)

retrieved

Query	cat dog
Doc4	cat cat cat dog
Doc1	cat dog dog bird
Doc3	fish cat dog bird
Doc2	dog bird bird fish

- Precision for top 3 documents $\frac{2}{3} = 0.66$
- Recall for top 3 documents $\frac{2}{3}$
- Precision for top 4 documents $\frac{2}{4}$
- Recall for top 4 documents $\frac{2}{3}$

	retrieved	not retrieved	
relevant	w	x	$n_1 = w + x$
not relevant	y	z	

$n_2 = w + y$

$N = \text{total documents} = w + x + y + z$

From all the documents that are relevant out there, how many did the IR system retrieve?

Recall: $\frac{w}{w+x}$

From all the documents that are retrieved by the IR system how many are relevant?

Precision: $\frac{w}{w+y}$

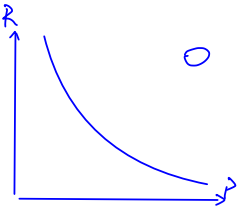
- Assume a collection of 10 documents: D1, D2, D3, … , D10
- Assume a query Q with relevant documents D3, D5, D7
- Assume a system that returns the following documents: D2, D7, D3, D10
- What is the precision and recall of this system?

Precision = $\frac{2}{4}$

Recall = $\frac{2}{3}$

return 100% recall

? system good.



Exercise 2

Consider the following relevance judgments:

relevants
- Query1: Doc5, Doc8, Doc20, Doc22
- Query2: Doc1, Doc19, Doc22
- Query3: Doc4, Doc6, Doc7, Doc8
- Query4: Doc1, Doc2, Doc3, Doc7

and the ranking produced by your system (assume this is for all queries):

Doc20, Doc8, Doc5, Doc7, Doc1, Doc6, Doc22

retrieved

What are the macro and micro-averaged precision and recall values at rank 1 and rank 5?

Doc 20

	P	R
Q_1:	$\frac{1}{1}=1$	$\frac{1}{4}=0.25$
Q_2:	$\frac{0}{1}$	$\frac{0}{3}$
Q_3:	$\frac{0}{1}$	$\frac{0}{4}$
Q_4:	$\frac{0}{1}$	$\frac{0}{4}$

Precision = # relevant returned/total # retrieved

Recall = # of relevant returned/total # of relevant

macro precision: $\frac{1+0+0+0}{4}$

macro recall: $\frac{0.25+0+0+0}{4}$

micro p: $\frac{1+0+0+0}{1+1+1+1}=\frac{1}{4}$

micro r: $\frac{1+0+0+0}{4+3+4+4}$

Probabilistic Model

- Most probabilistic models are based on combining probabilities of relevance and non-relevance of individual terms
  - Probability that a term will appear in a relevant document
  - Probability that the term will *not appear in a non-relevant document*
- These probabilities are estimated based on counting term appearances in document descriptions

term frequency.

Term Reweighting

F4 weight.

$$F4 = \log\left(\frac{\frac{r_i + 0.5}{0.5 + R - r_i}}{\frac{0.5 + n_i - r_i}{0.5 + N - n_i - R + r_i}}\right)$$

- r_i- relevant documents containing term i
- n_i- number of docs containing term i
- R - number of relevant documents in total
- N - total number of documents

r_i
relevant + non relevant

- F4 gives value/weight to documents that don't contain the query search terms