Introduction to Machine Learning and Data Mining Lecture-12: Information Retrieval

Prof. Eugene Chang

Today's Agenda

- Information Retrieval overview
  - Text feature extraction and analysis
  - Part of the material from Prof Bing Liu, UIC and Prof Raymond Mooney, UT Austin
- Additional optional resources

  3 most popular

  - Python NLTK: http://www.nltk.org/
  - Stanford NLP: http://nlp.stanford.edu/software/corenlp.shtml
  - Textblob: http://textblob.readthedocs.org/en/dev/index.html
  - Free IR book: http://www-nlp.stanford.edu/IR-book/
- Homework-5 will be published on 08/02
  - Due back on 08/10

Where are we heading?

- Today 07/31: information retrieval
- 08/02: Homework-4 due, homework-5 assigned
- 08/07: data mining topics
- 08/10: Homework-5 due
- 08/14: final exam reviews & project presentations
- 08/21: final exam
  - Close book & notes
  - Calculator only

Introduction

- Text mining refers to data mining using text documents as data.
- Most text mining tasks use Information Retrieval (IR) methods to pre-process text documents.
- These methods are quite different from traditional data pre-processing methods used for relational tables.
- Web search also has its root in IR.

Text Classification Examples

- LABELS = BINARY
  - “spam” / “ham”
- LABELS = TOPICS
  - “finance” / “sports” / “asia”
- LABELS = OPINION
  - “like” / “hate” / “neutral”
- LABELS = AUTHOR
  - “Shakespeare” / “Marlowe” / “Ben Jonson”
  - The Federalist papers

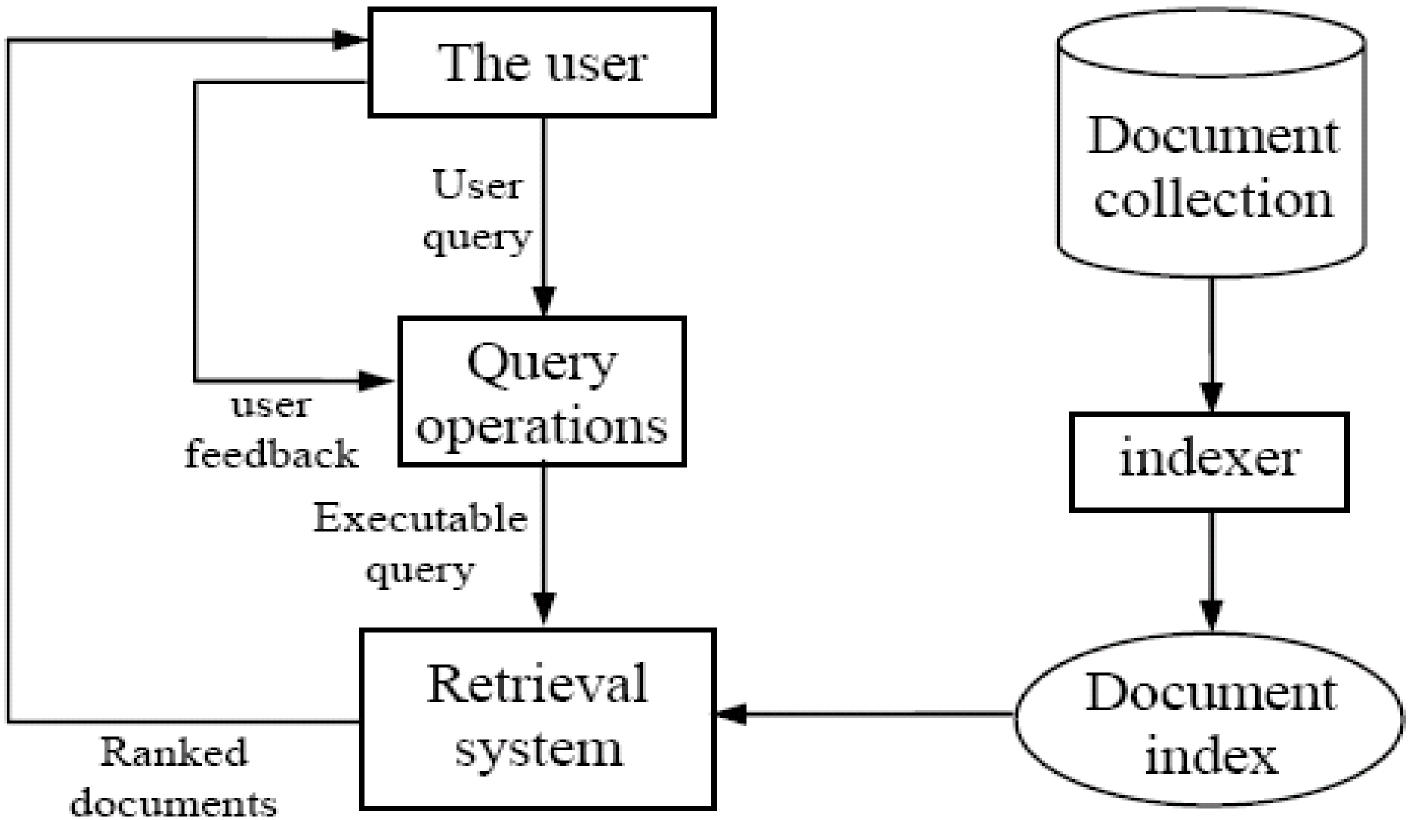
Text Mining Applications

- Web pages
  - Recommending
  - Yahoo-like classification
- Newsgroup/Blog Messages
  - Recommending
  - Spam filtering
  - Sentiment analysis for marketing
- Email messages
  - Routing & prioritizing
  - Spam filtering
  - Advertising (spamming)

Information Retrieval (IR)

- Conceptually, IR is the study of finding needed information. I.e., IR helps users find information that matches their information needs.
  - Expressed as queries
- Historically, IR is about document retrieval, emphasizing document as the basic unit.
  - Finding documents relevant to user queries
- Technically, IR studies the acquisition, organization, storage, retrieval, and distribution of information.

IR architecture



IR queries

- Keyword queries
- Boolean queries (using AND, OR, NOT)
- Phrase queries
- Proximity queries ← close enough, not need to be exact
- Full document queries
- Natural language questions

Information retrieval models

- An IR model governs how a document and a query are represented and how the relevance of a document to a user query is defined.
- Main models:
    - Boolean model
    - Naïve Bayes
    - Vector space model
    - Statistical language model

Boolean model

- Each document or query is treated as a **"bag" of words** or **terms**. Word sequence is not considered.

vocabulary word (term).

- Given a collection of documents D, let $V = \{t_1, t_2, ..., t_{|V|}\}$ be the set of distinctive words/terms in the collection. V is called the vocabulary.
- A weight $w_{ij} > 0$ is associated with each term t_i of a document $\mathbf{d}_j \in D$. For a term that does not appear in document $\mathbf{d}_j$, $w_{ij} = 0$.

$$\mathbf{d}_j = (w_{1j}, w_{2j}, ..., w_{|V|j}),$$

$|V| \leftarrow$ no of elements in V.

Document Collection

- A collection of *n* documents can be represented in the vector space model by a term-document matrix.
- An entry in the matrix corresponds to the "weight" of a term in the document; zero means the term has no significance in the document or it simply doesn't exist in the document.

term

weight: occurences of term T_i

doc

$$\begin{array}{ccccc}
 & T_1 & T_2 & \dots & T_t \\
D_1 & w_{11} & w_{21} & \dots & w_{t1} \\
D_2 & w_{12} & w_{22} & \dots & w_{t2} \\
\vdots & \vdots & \vdots & & \vdots \\
\vdots & \vdots & \vdots & & \vdots \\
D_n & w_{1n} & w_{2n} & \dots & w_{tn}
\end{array}$$

Boolean model (contd)

- Query terms are combined logically using the Boolean operators **AND**, **OR**, and **NOT.**
  - E.g., ((*data* AND *mining*) AND (NOT *text*))
- Retrieval
  - Given a Boolean query, the system retrieves every document that makes the query logically true.
  - Called **exact match**.
- The retrieval results are usually quite poor because term frequency is not considered.

Naïve Bayes for Text

- Modeled as generating a bag of words for a document in a given category by repeatedly sampling with replacement from a vocabulary $V = \{w_1, w_2, \ldots w_m\}$ based on the probabilities $P(w_j \mid c_i)$.
- Smooth probability estimates with Laplace *m*-estimates assuming a uniform distribution over all words ($p = 1/|V|$) and $m = |V|$
  - Equivalent to a virtual sample of seeing each word in each category exactly once.

Text Naïve Bayes Algorithm (Train)

Let V be the vocabulary of all words in the documents in D
For each category $c_i \in C$
  Let D_i be the subset of documents in D in category c_i
  $P(c_i) = |D_i| / |D|$
  Let T_i be the concatenation of all the documents in D_i
  Let n_i be the total number of word occurrences in T_i
  For each word $w_j \in V$
    Let n_{ij} be the number of occurrences of w_j in T_i
    Let $P(w_j \mid c_i) = (n_{ij} + 1) / (n_i + |V|)$ 🡐 *Laplacian smoothing*

Text Naïve Bayes Algorithm (Test)

Given a test document *X*

Let *n* be the number of word occurrences in *X*

Return the category:

$$\underset{c_i \in C}{\operatorname{argmax}} P(c_i) \prod_{i=1}^{n} P(a_i \mid c_i)$$

where a_i is the word occurring the *i*th position in *X*

Underflow Prevention

- Multiplying lots of probabilities, which are between 0 and 1 by definition, can result in floating-point underflow.
- Since $\log(xy) = \log(x) + \log(y)$, it is better to perform all computations by summing logs of probabilities rather than multiplying probabilities.
- Class with highest final un-normalized log probability score is still the most probable.

Naïve Bayes Posterior Probabilities

- Classification results of naïve Bayes (the class with maximum posterior probability) are usually fairly accurate.
- However, due to the inadequacy of the conditional independence assumption, the actual posterior-probability numerical estimates are not.
  - Output probabilities are generally very close to 0 or 1.

Example Naïve Bayes

	docID		c = China?
Training set	1	Chinese Beijing Chinese	Yes
	2	Chinese Chinese Shangai	Yes
	3	Chinese Macao	Yes
	4	Tokyo Japan Chinese	No
Test set	5	Chinese Chinese Chinese Tokyo Japan	?

4 docs
2 sets
(china or
non china)

Two classes: "China", "not China"

V = {Beijing, Chinese, Shangai, Japan, Macao, Tokyo}

N = 4 $\hat{P}(c) = 3/4$ $\hat{P}(\bar{c}) = 1/4$

Example Naïve Bayes

	docID		c = China?
Training set	1	Chinese Beijing Chinese	Yes
	2	Chinese Chinese Shangai	Yes
	3	Chinese Macao	Yes
	4	Tokyo Japan Chinese	No
Test set	5	Chinese Chinese Chinese Tokyo Japan	?

Laplacian smoothing

Estimation

$$\hat{P}(\text{Chinese} \mid c) = (5+1)/(8+6) = 3/7$$

$$\hat{P}(\text{Tokyo} \mid c) = \hat{P}(\text{Japan} \mid c) = (0+1)/(8+6) = 1/14$$

$$\hat{P}(\text{Chinese} \mid \overline{c}) = (1+1)/(3+6) = 2/9$$

$$\hat{P}(\text{Tokyo} \mid c) = \hat{P}(\text{Japan} \mid c) = (1+1)/(3+6) = 2/9$$

Classification

$$P(c \mid d) \propto P(c) \prod_{1 \le k \le n_d} P(t_k \mid c)$$

$$P(c \mid d_5) \propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003$$

$$P(\overline{c} \mid d_5) \propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001$$

eventually becomes very small.

① Naive Bayes
② Text
③ TF-IDF

Textual Similarity Metrics

- Measuring similarity of two texts is a well-studied problem.
- Standard metrics are based on a “bag of words” model of a document that ignores word order and syntactic structure.
- May involve removing common “stop words” and stemming to reduce words to their root form.
- Vector-space model from Information Retrieval (IR) is the standard approach.
- Other metrics (e.g. edit-distance) are also used.

The Vector-Space Model

- Assume t distinct terms remain after preprocessing; call them index terms or the vocabulary.
- These "orthogonal" terms form a vector space.

  Dimension = t = |vocabulary|
- Each term, i, in a document or query, j, is given a real-valued weight, w_{ij}.
- Both documents and queries are expressed as t-dimensional vectors:

  $d_j = (w_{1j}, w_{2j}, \ldots, w_{tj})$

$$D_1 \cdot Q = \frac{(2,3,5).(0,0,2)}{\|(2,3,5)\| . \|(0,0,2)\|} = \frac{2\times 0 + 3\times 0 + 5\times 2 = 10}{\sqrt{38} \times 2}$$

$$D_2 \cdot Q = \frac{(3,7,1).(0,0,2)}{\|(3,7,1)\| . \|0,0,2\|} = \frac{2}{\sqrt{59} \times 2}$$

Graphic Representation

$D_1 . Q > D_2 Q \Rightarrow$ Q is more similar to D_1

usually the vector is normalized to be less than 1.

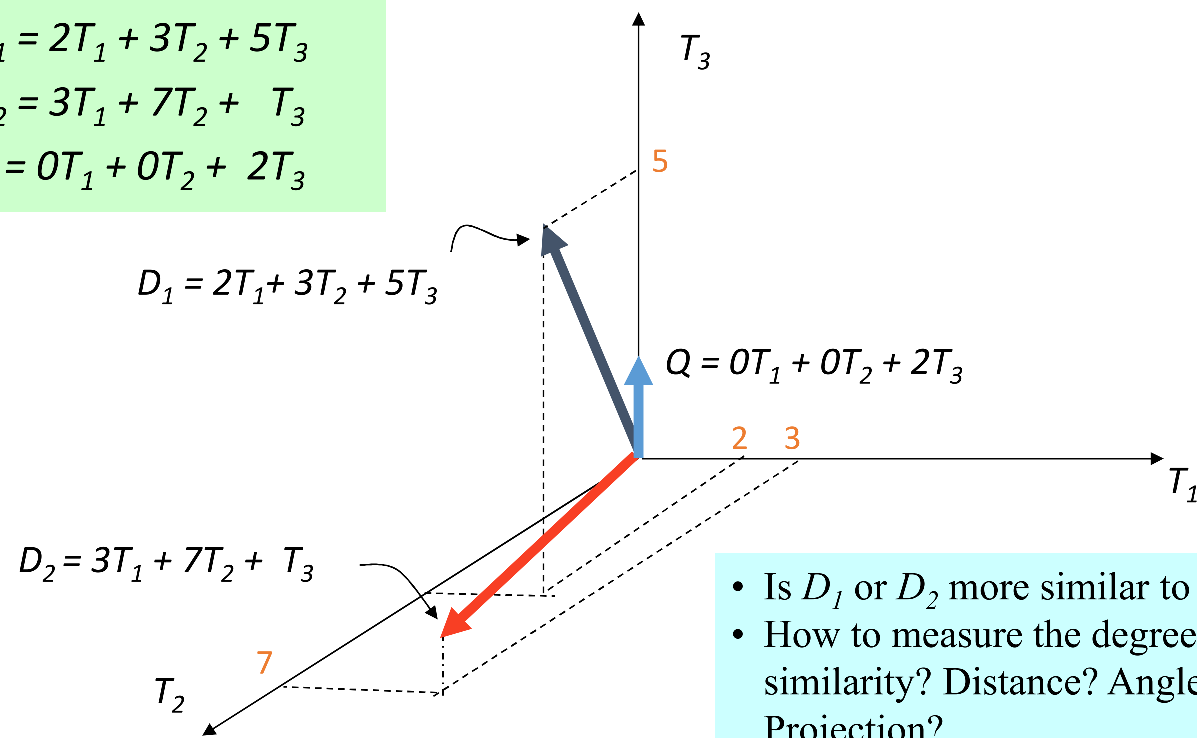
Example:

$D_1 = 2T_1 + 3T_2 + 5T_3$

$D_2 = 3T_1 + 7T_2 + T_3$

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

query



- Is D_1 or D_2 more similar to Q?
- How to measure the degree of similarity? Distance? Angle? Projection?

① Naives Bayes ✓.
② Text - cosine similarity ✓.
weight ?

Vector Space with Weights

- Documents are also treated as a "bag" of words or terms.
- Each document is represented as a vector.
- However, the term weights are no longer 0 or 1. Each term weight is computed based on some variations of TF or TF-IDF scheme.

occurence.

- Term Frequency (TF) Scheme: The weight of a term t_i in document $\mathbf{d}_j$ is the number of times that t_i appears in $\mathbf{d}_j$, denoted by f_{ij}. Normalization may also be applied.

Term Weights: Term Frequency

- More frequent terms in a document are more important, i.e. more indicative of the topic.

  f_{ij} = frequency of term *i* in document *j*

- May want to normalize *term frequency* (*tf*) by dividing by the frequency of the most common term in the document:

  $tf_{ij} = f_{ij} / max_i\{f_{ij}\}$

a b c.

5 3 2.

$f_a = \frac{5}{5}$, $f_b = \frac{3}{5}$, $f_c = \frac{2}{5}$

Term Weights: Inverse Document Frequency

- Terms that appear in many *different* documents are *less* indicative of overall topic.

  df_i = document frequency of term i ← encourage the more freq of a term

  = number of documents containing term i

  idf_i = inverse document frequency of term i, ← to penalize the term that appears in most doc

  $= \log_2 (N/df_i)$

  (N: total number of documents)

- An indication of a term's *discrimination* power.
- Log used to dampen the effect relative to *tf*.

TF-IDF Weighting

- A typical combined term importance indicator is *tf-idf weighting*:

$$w_{ij} = tf_{ij}\, idf_i = tf_{ij} \log_2 (N/\, df_i)$$

- A term occurring frequently in the document but rarely in the rest of the collection is given high weight.
- Many other ways of determining term weights have been proposed.
- Experimentally, *tf-idf* has been found to work well.

Example of TF-IDF

count the occurrence of term in doc

- Original counts=

9 docs

3 terms

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

5 5 6

less discriminate than other 2.

- Tfidf =

```
[[ 1.         0.         0.        ]
 [ 0.         1.         0.        ]
 [ 0.         0.         1.        ]
 [ 0.70710678  0.70710678  0.        ]
 [ 0.74404499  0.         0.66812952]
 [ 0.         0.74404499  0.66812952]
 [ 0.59693793  0.59693793
0.53603191]
 [ 0.74404499  0.         0.66812952]
 [ 0.         0.74404499  0.66812952]]
```

decrease

Retrieval in Vector Space Model

- Query **q** is represented in the same way or slightly differently.
- Relevance of $\mathbf{d}_i$ to **q**: Compare the similarity of query **q** and document $\mathbf{d}_i$.
- Cosine similarity (the cosine of the angle between the two vectors)

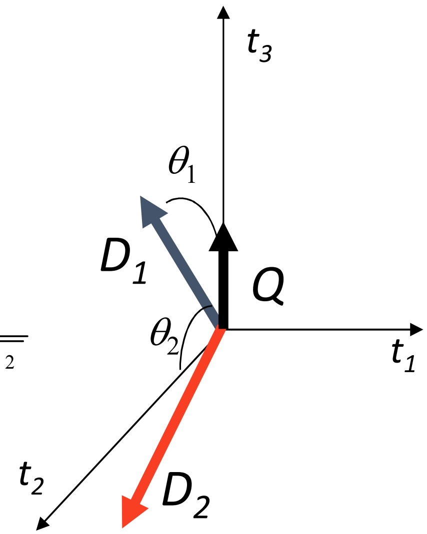
$$cosine(\mathbf{d}_j, \mathbf{q}) = \frac{\langle \mathbf{d}_j \bullet \mathbf{q} \rangle}{\| \mathbf{d}_j \| \times \| \mathbf{q} \|} = \frac{\sum_{i=1}^{|V|} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{|V|} w_{ij}^2} \times \sqrt{\sum_{i=1}^{|V|} w_{iq}^2}}$$

- Cosine is also commonly used in text clustering

measure similarity of 2 vectors

Cosine Similarity Measure

- Cosine similarity measures the cosine of the angle between two vectors.
- Inner product normalized by the vector lengths.



$$\text{CosSim}(\boldsymbol{d}_j, \boldsymbol{q}) = \frac{\vec{d}_j \cdot \vec{q}}{\left|\vec{d}_j\right| \cdot \left|\vec{q}\right|} = \frac{\sum_{i=1}^{t}(w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$

$D_1 = 2T_1 + 3T_2 + 5T_3$ CosSim(D_1 , Q) = $10 / \sqrt{(4+9+25)(0+0+4)} = 0.81$
$D_2 = 3T_1 + 7T_2 + 1T_3$ CosSim(D_2 , Q) = $2 / \sqrt{(9+49+1)(0+0+4)} = 0.13$
$Q = 0T_1 + 0T_2 + 2T_3$

D_1 is 6 times better than D_2 using cosine similarity but only 5 times better using inner product.

Example: Boolean vs Cosine vs TF-IDF

- A document space is defined by three terms:
  - hardware, software, users
  - the vocabulary
- A set of documents are defined as:
  - A1=(1, 0, 0), A2=(0, 1, 0), A3=(0, 0, 1)
  - A4=(1, 1, 0), A5=(1, 0, 1), A6=(0, 1, 1)
  - A7=(1, 1, 1) A8=(1, 0, 1). A9=(0, 1, 1)

9 doc. & weight vector (hw, sw, users)

- If the Query is "hardware and software"
- what documents should be retrieved? → A4, A7.

Example: Boolean vs Cosine vs TF-IDF

- For Boolean query matching:
  - document A4, A7 will be retrieved ("AND")
  - retrieved: A1, A2, A4, A5, A6, A7, A8, A9 ("OR")
- For similarity matching (cosine)
  - q=(1, 1, 0)
  - S(q, A1)=0.71,   S(q, A2)=0.71, S(q, A3)=0
  - S(q, A4)=1,   S(q, A5)=0.5,   S(q, A6)=0.5
  - S(q, A7)=0.82,   S(q, A8)=0.5,   S(q, A9)=0.5
  - Document retrieved set (with ranking)= {A4, A7, A1, A2, A5, A6, A8, A9}
- For similarity with TF-IDF weights
  - 0.71, 0.71, 0, 1, 0.526 , 0.526, 0.844 , 0.526 , 0.526

$$\frac{(1\,1\,0)(1\,0\,0)}{\|1\,1\,0\|\cdot\|1\,0\,0\|} = \frac{1}{\sqrt{2}}$$

A4. A7.

Sk-learn Text Feature Extraction

- Utilities provided
  - Tokenizing
  - Counting
  - Normalizing
- Functions
  - CountVectorizer
  - HashingVectorizer
  - DictVectorizer
  - FeatureHasher
  - TfidfTransformer

Python Examples

- text_feature.py
  - CountVectorizer
  - TfidfTransformer
  - TfidfVectorizer
- grid_search_etxt_feature_extraction.py
  - SGDClassifier
  - GridSearchCV
  - Pipeline
- document_clustering.py
  - fetch_20newsgroups
  - HashingVectorizer
  - KMeans, MiniBatchKMeans
- document_classification_20newsgroups.py ← compare score etc... interesting.

Text Pre-processing

- Word (term) extraction: easy
- Stopwords removal
- Stemming

Stopwords Removal

before TF IDF

- Many of the most frequently used words in English are useless in IR and text mining – these words are called *stop words*.
  - the, of, and, to, ….
  - Typically about 400 to 500 such words
  - For an application, an additional domain specific stopwords list may be constructed
- Why do we need to remove stopwords?
  - Reduce indexing (or data) file size
    - stopwords accounts 20-30% of total word counts.
  - Improve efficiency and effectiveness
    - stopwords are not useful for searching or text mining
    - they may also confuse the retrieval system.

Stemming

- Techniques used to find out the root/stem of a word. E.g.,

• user	engineering
• users	engineered
• used	engineer
• using	

Stemming

- stem: use engineer

Usefulness:

- improving effectiveness of IR and text mining
  - matching similar words
  - Mainly improve recall
- reducing indexing size
  - combing words with same roots may reduce indexing size as much as 40-50%.

Basic Stemming Methods

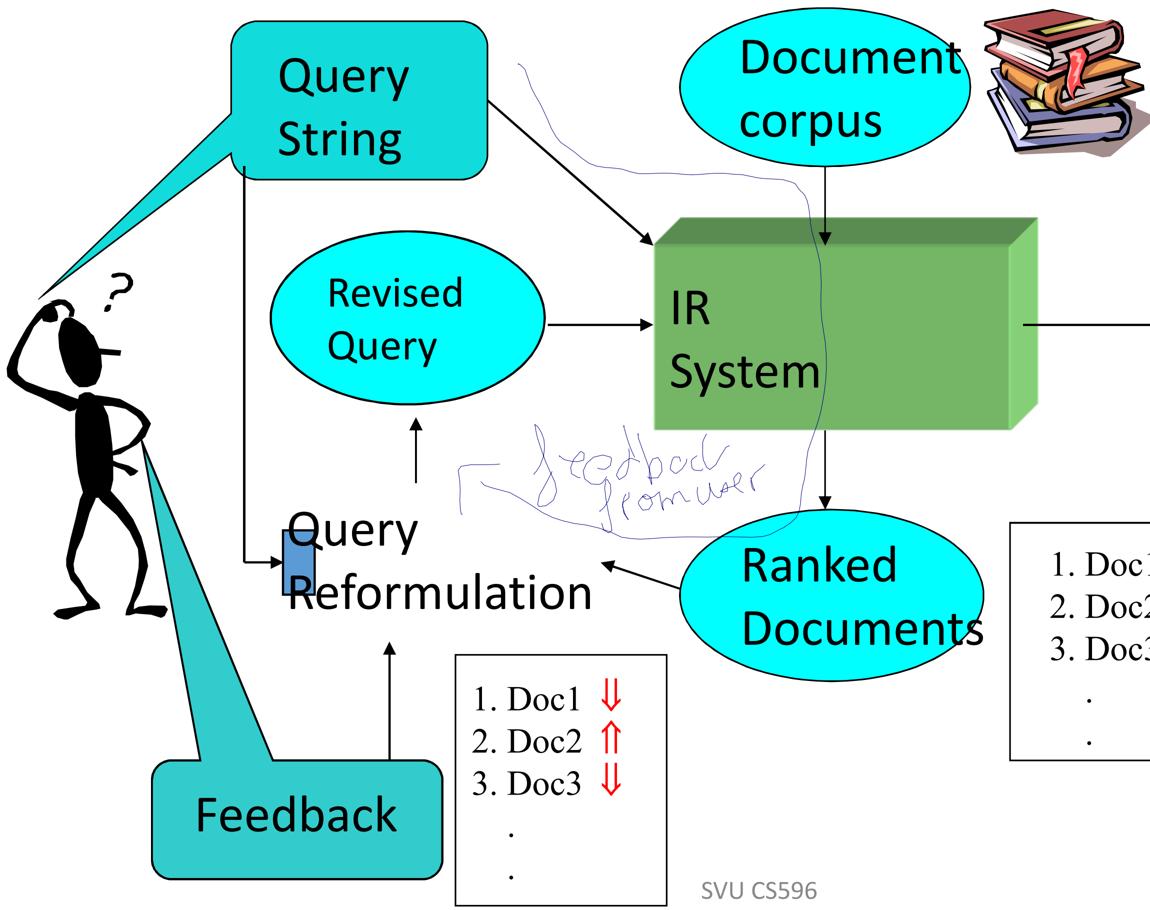
Using a set of rules. E.g.,

- remove ending
  - if a word ends with a consonant other than s,
    followed by an s, then delete s.
  - if a word ends in es, drop the s.
  - if a word ends in ing, delete the ing unless the remaining word consists only of one letter or of th.
  - If a word ends with ed, preceded by a consonant, delete the ed unless this leaves only a single letter.
  - ......
- transform words
  - if a word ends with “ies” but not “eies” or “aies” then “ies --> y.”

Relevance Feedback in IR

- After initial retrieval results are presented, allow the user to provide feedback on the relevance of one or more of the retrieved documents.
- Use this feedback information to reformulate the query.
- Produce new results based on reformulated query.
- Allows more interactive, multi-pass process.

Relevance Feedback Architecture



Relevance feedback

- Relevance feedback is one of the techniques for improving retrieval effectiveness. The steps:
  - the user first identifies some relevant (D_r) and irrelevant documents (D_{ir}) in the initial list of retrieved documents
  - the system expands the query **q** by extracting some additional terms from the sample relevant and irrelevant documents to produce $\mathbf{q}_e$
  - Perform a second round of retrieval.
- Rocchio method (α, β and γ are parameters)

$$\mathbf{q}_e = \alpha\mathbf{q} + \frac{\beta}{|D_r|}\underbrace{\sum_{\mathbf{d}_r \in D_r}\mathbf{d}_r}_{\text{all documents in relevant set}} - \frac{\gamma}{|D_{ir}|}\underbrace{\sum_{\mathbf{d}_{ir} \in D_{ir}}\mathbf{d}_{ir}}_{\text{all docs in irrelevant set.}}$$

Rocchio text classifier

- In fact, a variation of the Rocchio method above, called the **Rocchio classification** method, can be used to improve retrieval effectiveness
  - so are other machine learning methods. Why?
- Rocchio classifier is constructed by producing a prototype vector $\mathbf{c}_i$ for each class *i* (*relevant* or *irrelevant* in this case):

centroid or prototype

$$\mathbf{c}_i = \frac{\alpha}{|D_i|}\sum_{\mathbf{d}\in D_i}\frac{\mathbf{d}}{\|\mathbf{d}\|} - \frac{\beta}{\underbrace{|D - D_i|}_{\text{not in current class}}}\sum_{\mathbf{d}\in D-D_i}\frac{\mathbf{d}}{\|\mathbf{d}\|}$$

in current class

- In classification, cosine is used.
- Also knows as Nearest Centroid Classifier

Using Relevance Feedback (Rocchio)

- Relevance feedback methods can be adapted for text categorization.
- Use standard TF/IDF weighted vectors to represent text documents (normalized by maximum term frequency).
- For each category, compute a *prototype* vector (centroid) by summing the vectors of the training documents in the category.
- Assign test documents to the category with the closest prototype vector based on cosine similarity.

Rocchio Text Categorization (Training)

Assume the set of categories is $\{c_1, c_2, \ldots c_n\}$

For i from 1 to n let $\mathbf{p}_i = \langle 0, 0, \ldots, 0 \rangle$ (*init. prototype vectors*)

For each training example $\langle x, c(x) \rangle \in D$

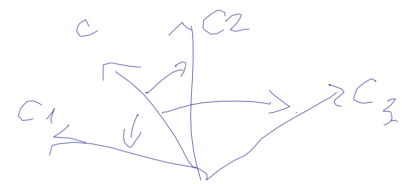
  Let $\mathbf{d}$ be the frequency normalized TF/IDF term vector for doc x

  Let $i = j: (c_j = c(x))$

  (*sum all the document vectors in c_i to get $\boldsymbol{p}_i$*)

  Let $\mathbf{p}_i = \mathbf{p}_i + \mathbf{d}$

Rocchio Text Categorization Prediction



Given test document x
Let $\mathbf{d}$ be the TF/IDF weighted term vector for x
Let $m = -2$ (*init. maximum cosSim*)
For i from 1 to n:
    (*compute similarity to prototype vector*)
    Let s = cosSim($\mathbf{d}$, $\mathbf{p}_i$)
    if $s > m$
        let $m = s$
        let $r = c_i$ (*update most similar class prototype*)
Return class r

Evaluation: Precision and Recall

- Given a query:
  - Are all retrieved documents relevant?
  - Have all the relevant documents been retrieved?
- Measures for system performance:
  - The first question is about the precision of the search
  - The second is about the completeness (recall) of the search.

Precision-recall curve

Example 2: Following Example 1, we obtain the interpolated precisions at all 11 recall levels in the table of Fig. 6.4. The precision-recall curve is shown on the right.

i	$p(r_i)$	r_i
0	100%	0%
1	100%	10%
2	100%	20%
3	100%	30%
4	80%	40%
5	80%	50%
6	71%	60%
7	70%	70%
8	70%	80%
9	62%	90%
10	62%	100%

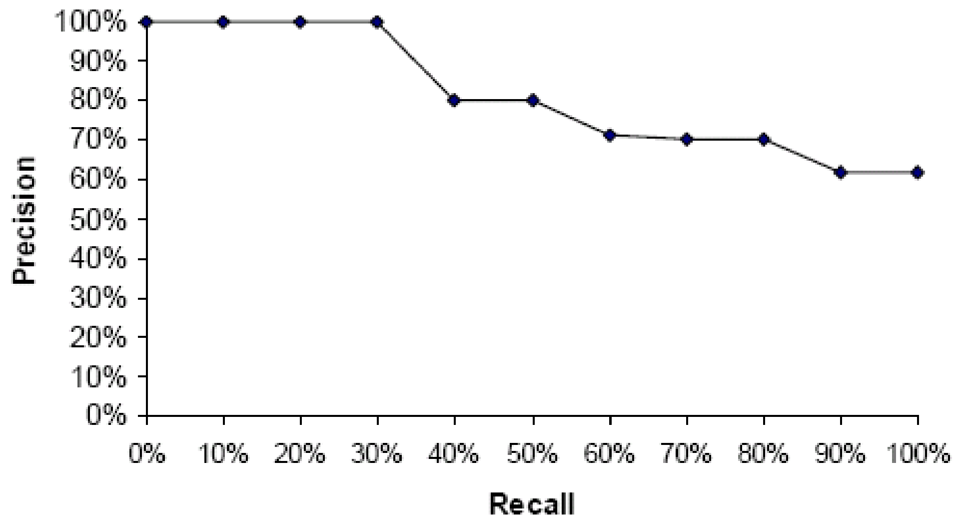


Fig. 6.4. The precision-recall curve ■

Compare different retrieval algorithms

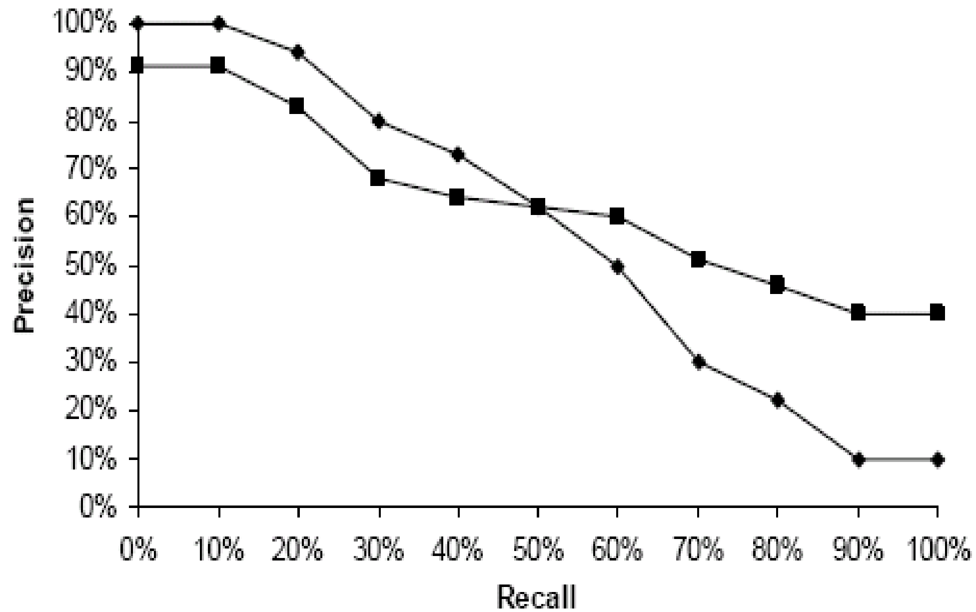


Fig. 6.5. Comparison of two retrieval algorithms based on their precision-recall curves

Compare with multiple queries

- Compute the average precision at each recall level.

$$\overline{p}(r_i) = \frac{1}{|Q|}\sum_{j=1}^{|Q|} p_j(r_i), \tag{22}$$

where Q is the set of all queries and $p_j(r_i)$ is the precision of query j at the recall level r_i. Using the average precision at each recall level, we can also draw a precision-recall curve.

- Draw precision recall curves
- Do not forget the F-score evaluation measure.

Rank precision

- Compute the precision values at some selected rank positions.
- Mainly used in Web search evaluation.
- For a Web search engine, we can compute precisions for the top 5, 10, 15, 20, 25 and 30 returned pages
  - as the user seldom looks at more than 30 pages.
- Recall is not very meaningful in Web search.
  - Why?

Web Search as a huge IR system

- A Web crawler (robot) crawls the Web to collect all the pages.
- Servers establish a huge inverted indexing database and other indexing databases
- At query (search) time, search engines conduct different types of vector query matching.

each word has an inverted array that reference the document.
for example $w_1 = d_1, d_3, d_5 \ldots$
only need to index, no

Inverted index

- The inverted index of a document collection is basically a data structure that
  - attaches each distinctive term with a list of all documents that contains the term.
- Thus, in retrieval, it takes constant time to
  - find the documents that contains a query term.
  - multiple query terms are also easy handle as we will see soon.

An example

Example 3: We have three documents of id_1, id_2, and id_3:

id_1: Web mining is useful.
1 2 3 4

id_2: Usage mining applications.
1 2 3

id_3: Web structure mining studies the Web hyperlink structure.
1 2 3 4 5 6 7 8

Applications:	id_2
Hyperlink:	id_3
Mining:	id_1, id_2, id_3
Structure:	id_3
Studies:	id_3
Usage:	id_2
Useful:	id_1
Web:	id_1, id_3

(A)

frequency index.

Applications:	<id_2, 1, [3]>
Hyperlink:	<id_3, 1, [7]>
Mining:	<id_1, 1, [2]>, <id_2, 1, [2]>, <id_3, 1, [3]>
Structure:	<id_3, 2, [2, 8]>
Studies:	<id_3, 1, [4]>
Usage:	<id_2, 1, [1]>
Useful:	<id_1, 1, [4]>
Web:	<id_1, 1, [1]>, <id_3, 2, [1, 6]>

(B)

Fig. 6.7. Two inverted indices: a simple version and a more complex version

Index construction

- Easy! See the example,

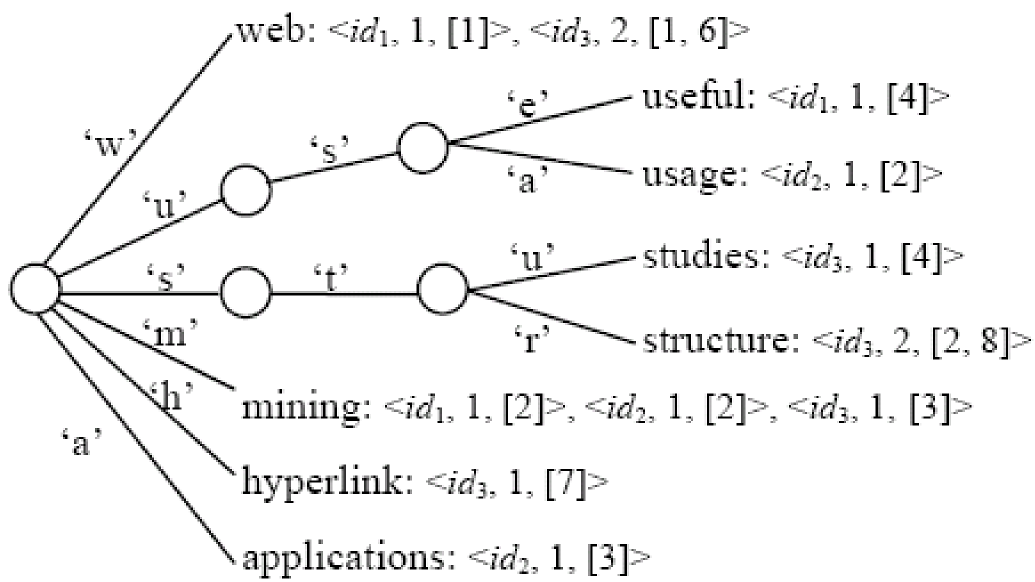


Fig. 6.8. The vocabulary trie and the inverted lists

Search using inverted index

Given a query **q**, search has the following steps:

- Step 1 (**vocabulary search**): find each term/word in **q** in the inverted index.
- Step 2 (**results merging)**: Merge results to find documents that contain all or some of the words/terms in **q**.
- Step 3 (**Rank score computation**): To rank the resulting documents/pages, using,
  - content-based ranking
  - link-based ranking

Different search engines

- The real differences among different search engines are
  - their index weighting schemes
    - Including location of terms, e.g., title, body, emphasized words, etc.
  - their query processing methods (e.g., query classification, expansion, etc)
  - their ranking algorithms most important.
  - Few of these are published by any of the search engine companies. They are tightly guarded secrets.

Inverted Index

- Linear search through training texts is not scalable.
- An index that points from words to documents that contain them allows more rapid retrieval of similar documents.
- Once stop-words are eliminated, the remaining words are rare, so an inverted index narrows attention to a relatively small number of documents that share meaningful vocabulary with the test document.

Summary

- We only give a VERY brief introduction to IR. There are a large number of other topics, e.g.,
  - Statistical language model
  - Latent semantic indexing (LSI and SVD).
  - (read an IR book or take an IR course)
- Many other interesting topics are not covered, e.g.,
  - Web search
    - Index compression
    - Ranking: combining contents and hyperlinks
  - Web page pre-processing
  - Combining multiple rankings and meta search
  - Web spamming
- Want to know more? Read the textbook

Python Text Analysis Tools

Tokenization, Tagging, Parsing, Word Embeddings

From Vicente Ordonez

Natural Language Processing

- Concerned with interactions between computers and human languages
- Derive meaning from text
- Many NLP algorithms are based on machine learning

Python Scikit-learn: Bag of Words

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(analyzer = "word",   \
                             tokenizer = tokenize,    \
                           stop_words = "english",   \
                             max_features = 5000)

// Compute vocabulary of top 5000 most frequent words
corpus_features = vectorizer.fit_transform(text_corpus)


// On test data
features = vectorizer.transform('My cat likes eating bananas')
```

Python NLTK

http://www.nltk.org/

“My cat likes eating bananas”

Python NLTK

- Natural Language ToolKit
- Access to over 50 corpora
  - Corpus: body of text
- NLP tools
  - Stemming, tokenizing, etc
- Resources for learning

Python NLTK

- Stopword removal

```
from nltk.corpus import gutenberg

', '.join(gutenberg.words()[:100])
```

```
"[, Emma, by, Jane, Austen, 1816, ], VOLUME, I, CHAPTER, I, Emma, Wo
odhouse, ,, handsome, ,, clever, ,, and, rich, ,, with, a, comfortab
le, home, and, happy, disposition, ,, seemed, to, unite, some, of, t
he, best, blessings, of, existence, ;, and, had, lived, nearly, twen
ty, -, one, years, in, the, world, with, very, little, to, distress,
 or, vex, her, ., She, was, the, youngest, of, the, two, daughters,
of, a, most, affectionate, ,, indulgent, father, ;, and, had, ,, in,
 consequence, of, her, sister, ', s, marriage, ,, been, mistress, of
, his, house, from, a, very, early, period, ., Her"
```

Python NLTK

- Stopword removal

```
from nltk.corpus import stopwords

stop = stopwords.words('english')
stop_removed = [word for word in gutenberg.words() if word not in stop]

', '.join(stop_removed[:100])
```

```
'Emma, Jane, Austen, 1816, VOLUME, I, CHAPTER, I, Emma, Woodhouse, handsome,
 clever, rich, comfortable, home, happy, disposition, seemed, unite, best, b
lessings, existence, lived, nearly, twenty, one, years, world, little, distr
ess, vex, She, youngest, two, daughters, affectionate, indulgent, father, co
nsequence, sister, marriage, mistress, house, early, period, Her, mother, di
ed, long, ago, indistinct, remembrance, caresses, place, supplied, excellent
, woman, governess, fallen, little, short, mother, affection, Sixteen, years
, Miss, Taylor, Mr, Woodhouse, family, less, governess, friend, fond, daught
ers, particularly, Emma, Between, _them_, intimacy, sisters, Even, Miss, Tay
lor, ceased, hold, nominal, office, governess, mildness, temper, hardly, all
owed, impose, restraint, shadow, authority, long, passed, away'
```

Python NLTK

- Stemming

```
from nltk.stem.porter import PorterStemmer

stemmer = PorterStemmer()
stemmed = [stemmer.stem(word) for word in stop_removed]

', '.join(stemmed[:100])
```

```
'Emma, Jane, Austen, 1816, VOLUM, I, CHAPTER, I, Emma, Woodhous, handsom, cl
ever, rich, comfort, home, happi, disposit, seem, unit, best, bless, exist,
live, nearli, twenti, one, year, world, littl, distress, vex, She, youngest,
 two, daughter, affection, indulg, father, consequ, sister, marriag, mistres
s, hous, earli, period, Her, mother, die, long, ago, indistinct, remembr, ca
ress, place, suppli, excel, woman, gover, fallen, littl, short, mother, affe
ct, Sixteen, year, Miss, Taylor, Mr, Woodhous, famili, less, gover, friend,
fond, daughter, particularli, Emma, Between, _them_, intimaci, sister, Even,
 Miss, Taylor, ceas, hold, nomin, offic, gover, mild, temper, hardli, allow,
 impos, restraint, shadow, author, long, pass, away'
```

http://www.nltk.org/

Python NLTK: Tokenization

```
import nltk

nltk.word_tokenize("My cat likes eating bananas")

>>['My', 'cat', 'likes', 'eating', 'bananas']
```

http://www.nltk.org/

Python NLTK: POS Tagging

```
import nltk

words = nltk.word_tokenize("My cat likes eating bananas")

nltk.pos_tag(words)
```

noun verb. verb-ing

```
>>[('My', 'PRP$'), ('cat', 'NN'), ('likes', 'VBZ'), ('eating', 'VBG'), ('bananas',
'NNS')]
```

Penn Treebank Postagging
http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

http://www.nltk.org/

Python NLTK: Named Entities

```
import nltk

words = nltk.word_tokenize("My uncle Fred's cat likes eating bananas")

tags = nltk.pos_tag(words)

nlkt.ne_chunk(tag)

>>Tree('S', [('My', 'PRP$'), ('uncle', 'NN'), Tree('PERSON', [(Fred',
'NNP')]), ("'s", 'POS'), ('cat', 'NN'), ('likes', 'VBZ'), ('eating', 'VBG'),
('bananas', 'NNS')])
```

name is important when doing social media txt mining.

http://www.nltk.org/

Python NLTK: Wordnet

```
from nltk.corpus import wordnet

wordnet.synsets('dog') // works even if you use ‘dogs’ instead

>> [Synset('dog.n.01'), Synset('frump.n.01'), Synset('dog.n.03'),
Synset('cad.n.01'),
Synset('frank.n.02'), Synset('pawl.n.01'), Synset('andiron.n.01'),
Synset('chase.v.01')]

synset = wn.synset('dog.n.01')

// You can get definition, lemmas, examples, hypernyms (“parent
words”), hyponyms (“children words”), etc
```

http://www.nltk.org/

Python NLTK: Wordnet Similarity

```
from nltk.corpus import wordnet

dog = wn.synset('dog.n.01')
cat = wn.synset('cat.n.01')

similarity_score = dog.path_similarity(cat)
similarity_score = dog.wup_similarity(cat)
```

need to do nltk download first.

[end] 5)

nltk_example.py

Python NLTK

- Other things
  - Lemmatizing, tokenization, tagging, parse trees
  - Classification
  - Chunking
  - Sentence structure

$$C_{ham} = \frac{(1,1,0,0,1)}{\sqrt{3}} + \frac{(1,0,1,1,0)}{\sqrt{3}} = \frac{\quad}{2}$$

HW Prob NB classifier.

		it	is	load	noise	noisy
spam	It is noisy	1	1	0	0	1
ham	Noise is load	1	1	1	1	0
spam	it is it	1	2	0	0	0
ham	is it noise	1	1	0	1	0

http://nlp.stanford.edu/software/corenlp.shtml

Stanford Core NLP: Parsing

```
Properties props = new Properties();
props.setProperty("annotators", "tokenize, ssplit, pos, lemma, ner, parse");

StanfordCoreNLP pipeline = new StanfordCoreNLP(props);

String text = "My cat likes eating bananas";
Annotation document = new Annotation(text);
pipeline.annotate(document);

List<CoreMap> sentences = document.get(SentencesAnnotation.class);

for(CoreMap sentence: sentences) {
        Tree tree = sentence.get(TreeAnnotation.class);
         // Do something here with the tree
}
```

DEMO: http://nlp.stanford.edu:8080/parser/index.jsp

http://nlp.stanford.edu/software/corenlp.shtml

Stanford Core NLP: Dependencies

```
Properties props = new Properties();
props.setProperty("annotators", "tokenize, ssplit, pos, lemma, ner, parse");

StanfordCoreNLP pipeline = new StanfordCoreNLP(props);

String text = “My cat likes eating bananas";
Annotation document = new Annotation(text);
pipeline.annotate(document);

List<CoreMap> sentences = document.get(SentencesAnnotation.class);

for(CoreMap sentence: sentences) {
        SemanticGraph graph = sentence.get(
            CollapsedCCProcessedDependenciesAnnotation.class);
            // Do something here with the graph
}
```

DEMO: http://nlp.stanford.edu:8080/parser/index.jsp

http://nlp.stanford.edu/software/corenlp.shtml

Stanford Core NLP: Sentiment

```
Properties props = new Properties();
props.setProperty("annotators", "tokenize, ssplit, pos, lemma, ner, parse, sentiment");

StanfordCoreNLP pipeline = new StanfordCoreNLP(props);

String text = "My cat likes eating bananas";
Annotation document = new Annotation(text);
pipeline.annotate(document);

List<CoreMap> sentences = document.get(SentencesAnnotation.class);

for(CoreMap sentence: sentences) {
    Tree tree = sentence.get(SentimentCoreAnnotations.AnnotatedTree.class);
    int sentiment = RNNCoreAnnotations.getPredictedClass(tree);
    // Do something here with the sentiment tree
}
```

DEMO: http://nlp.stanford.edu:8080/sentiment/rntnDemo.html

Text Analysis Summary

- Basic Text Analysis using NTLK
  - Splitting a sentence into words – tokenization.
  - Extracting nouns, verbs, adjectives, etc – POS-tagging
  - Computing word similarities – Wordnet
- Sentence Parsing using StanfordNLP
  - Breaking sentences into its subject, predicate, etc.
  - Resolving word dependencies.
  - Sentiment Analysis
- Word representations
  - Bag of Words - Scikit-learn
  - Neural Networks – Word2vec