COMP348 — Document Processing and the Semantic Web

Week 02 Lecture 1: Searching for Information

Diego Mollá

COMP348 2019H1

Abstract

In this lecture we will survey the key technology that makes it possible to search for relevant documents in a collection of documents.

Update March 2, 2019

Contents

Reading

Essential Reading

- NLTK chapter 6 section 3.3 (precision and recall).
- Manning et al. IR book, chapter 1 (Boolean retrieval), chapter 6 section 2 (Td.idf), chapter 8 section 3 (precision and recall). http://nlp.stanford.edu/IR-book/

Additional Reading

- Brin and Page (1998): http://infolab.stanford.edu/~backrub/google.html — a famous paper by the founders of Google.

1 Information Retrieval

Need for Search

The Problem

- The Web can be seen as a very large, unstructured data store.
- There exist hundreds of millions of Web pages but there is no central index.
- Even worse: It is unknown where all the Web servers are.

The Solution

Search engines.

Information Retrieval

Information Retrieval (IR)

- IR is about searching for information.
- IR typically means "document retrieval".
- IR is one of the core components of Web search.

`http://boston.lti.cs.cmu.edu/classes/11-744/treclogo-c.gif`

Stages in an IR System

1: Indexing

- This stage is done off-line, prior to any searches.
- The goal is to reduce the documents to a description: the indices.
- Optimise the representation: ignore the terms that do not contribute.

2: Retrieval

- Use the indices to retrieve the documents (ignore the remaining information in the documents).

2 Evaluation

Why Evaluate?

- Document processing systems almost never give 100% correct results.
- When you develop a document processing system, you want to know how good it is.
- You want to know if a modification in a system is an improvement.
- Human evaluations are expensive to produce.
- In this lecture we will focus on automatic evaluations.

Of course, *in addition* you have to debug the system.

Training *vs.* Test Data

- For pretty much all evaluations, you want to divide your data into at least two sets: training and test.
- Training data is what you use to develop your models.
  - You only look at the training data.
  - For statistical models (coming later in this course), this is what you use to calculate your statistics.
- Test data is separate.
- You may also have a third set of data to help develop your system (DevTest).
  - You'll see the use of the DevTest set when we look at statistical models.

Golden Rule

You don't ever, ever, look at the test data (you only look at its evaluation results).

2.1 Precision and Recall

Types of Errors

Errors by a system making a binary choice can often be broken into two types:

1. Selecting something when it's not supposed to be selected.
2. Not selecting something when it is supposed to be selected.

Examples

1. If the task is to identify whether an email is spam or not, the system can mistakenly classify an email as spam when it is not spam.
2. If the task is to identify documents as relevant or not, the system can mistakenly classify irrelevant documents as relevant, or relevant documents as irrelevant.

Positives and Negatives

We can group results of the system into four categories: tp (true positive), fp (false positive), fn (false negative), tn (true negative)

system decision	actual case: target	actual case: not target
selected	tp	fp
not selected	fn	tn

Example: Positives and Negatives in Information Retrieval

In IR, "relevant" documents belong to the target category.

system decision	actual case: relevant	actual case: not relevant
retrieved	tp	fp
not retrieved	*fn*	tn

- Our retrieval system fails to retrieve a relevant document: this is a *false negative*.

Example: Spam Filtering

In spam filtering, "spam" emails belong to the target category.

	actual case: spam	actual case: not spam
system		
marked spam	tp	*fp*
not marked spam	fn	tn

- Our spam filter classifies a legitimate email as spam: this is a *false positive*.

Question

False positives in spam filtering are usually more dangerous than false negatives; why?

Precision and Recall

Formulas

- precision = tp / (tp + fp)
- recall = tp / (tp + fn)

Example

From a total collection of 200 documents, a retrieval system returned 30 documents, but 5 were not relevant. It also missed 12 documents.

Example

	actual case: target	actual case: non target
system		
selected	25	5
not selected	12	158

Values of measures

- precision = 25/30
- recall = 25/37

Accuracy

- Accuracy is the number correctly classified out of the whole set.
  - accuracy = (tp + tn) / (tp + fp + tn + fn)
  - For previous example, accuracy is 183/200
- Sometimes used (inaccurately) to refer to precision.

Question
What happens if you have unbalanced classes (e.g. 90% of the data belongs to class 1)?

F-Measure

- Another way of getting a single measure for a system is to combine precision and recall.
- For the general case,
$$F_\beta = (1 + \beta^2)\frac{\text{precision} \times \text{recall}}{\beta^2\text{precision} + \text{recall}}$$
- The most commonly used instance is when $\beta = 1$, referred to as F_1 :
$$F_1 = 2\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$
- This is just the harmonic mean of precision and recall.
- For previous example, $F_1 = 0.746$

Exercise: Spam Filtering

Exercise
Assume your system processes 1000 emails. It classifies 640 as spam, of which 480 are actually spam. It missed 120 spam emails. What are the precision and recall of the spam detection and the accuracy of the system?

3 Indexing and Retrieval

3.1 Indexing

Bag of Words Representation

Bag of words (BoW)

- At indexing time, a compact representation of the document is built.
- The document is seen as a bag of words.
- Information about word position is (often) discarded.
- Only the important words are kept.

The bag-of-words model is a simplifying representation used in natural language processing and information retrieval (IR). In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. Recently, the bag-of-words model has also been used for computer vision. $\Longrightarrow$

{bag, bag-of-words, computer, disregarding, document, grammar, information, IR, keeping, language, model, multiplicity, multiset, natural, order, processing, representation, represented, retrieval, sentence, simplifying, text, vision, word, words}

Stop Words

Stop words

- A simple solution to determine important words is to keep a list of non-important words: the stop words.
- All stop words in a document are ignored.
- Stop words are language-specific.
- Typically, stop words are connecting words.

Stop words in NLTK

```
>>> from nltk.corpus import stopwords
>>> stop = stopwords.words('english')
>>> stop[:5]
['i', 'me', 'my', 'myself', 'we']
```

Term Frequency

Term Frequency

- Words that are not frequent are usually not important.
- Words that are too frequent may occur in most documents and therefore can't be used to discriminate among documents.
- Usually, important words are in the middle.

Zipf's Law for term frequency

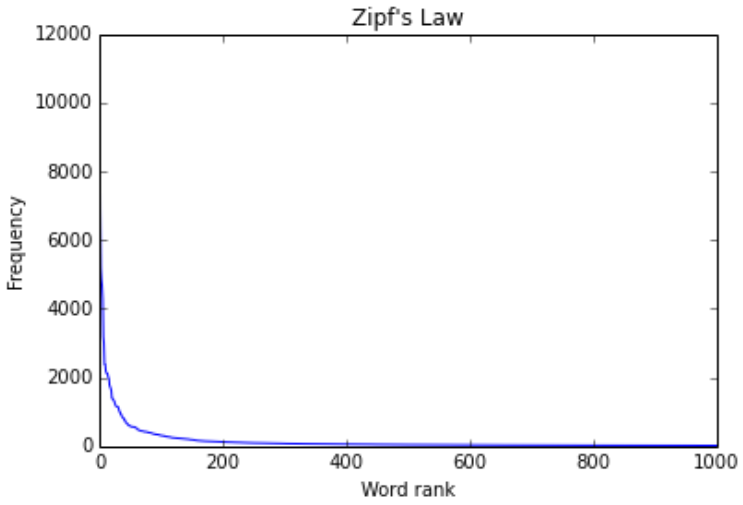
- A small percentage of words are very frequent.
- A large percentage of words have very little frequency.
- The relation approximates a Zipfian distribution.
- This is also referred as "long-tailed" distribution.

Zipf's Law in Action

Python code

```
import nltk
import collections
import matplotlib.pyplot as plt
words = nltk.corpus.gutenberg.words('austen-emma.txt')
fd = collections.Counter(words)
data = sorted([fd[k] for k in fd], reverse=True)
plt.plot(data[:1000])
plt.show()
```

500 most frequent words



tf.idf

tf.idf

- Term frequency: If a word is very frequent in a document, it is important for the document.

$$tf(t,d) = \text{frequency of word } t \text{ in document } d$$

- Inverse document frequency: If a word appears in many documents, it is not important for any document.

$$idf(t) = \log \frac{\text{number of documents}}{\text{number of documents that contain } t}$$

- $tf.idf$ combines these two characteristics.

$$tf.idf(t,d) = tf(t,d) \times idf(t)$$

Note that tf is a function of the term and the document, whereas idf is a function of the term, irrelevant of the document. To compute $tf.idf$ we need to have a collection of documents, otherwise idf is irrelevant.

Problems with Bag of Word Representations

BoW representations ignore important information such as:

Word position: "Australia beat New Zealand" is not the same as "New Zealand beat Australia"

Morphology: If you search for "table", a webpage that uses the word "tables" might be relevant.

Words with similar meanings: If you search for "truck", a webpage that uses the word "lorry" might be relevant.

Ambiguity: If you search for "Apple" you might be interested in the company and not in the fruit.

Still, BoW representations are very simple, fast, and often surprisingly good.

Beyond BoW Representations

- A simple way to account for (some) information about word positions is to use n-grams:
  - Bigrams, trigrams, 4-grams (usually there is no need for longer n-grams).
- Thus, instead of representing a text as a bag of words, it can be represented as a bag of n-grams.
- Using n-grams instead of words may introduce other kinds of problems (we will see some of these problems in a future lecture).

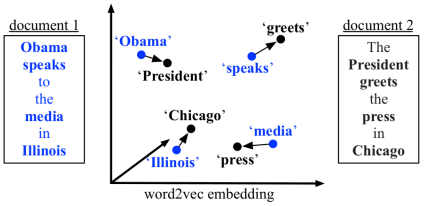
Accounting for Word Meaning

Ambiguity

- Word disambiguation attempts to determine the sense of a word.
- A word like "Apple" could be disambiguated as "apple1" or "apple2" to account for its several meanings.
- Word disambiguation systems usually look at the "context" of the word:
  - Yesterday I ate an apple_1.
  - Apple_2 reported a benefit last fiscal year.

Synonymy

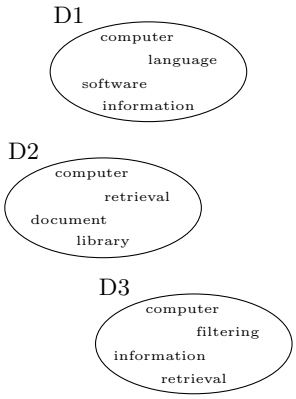
- There are lexical resources such as thesauri (singular: thesaurus) that list words with related meanings.
  - WordNet is a popular resource
- Recent innovations include the use of distributional semantics to determine the similarity between two words.
  - Word2Vec and Glove are two systems that map words to vectors called word embeddings.



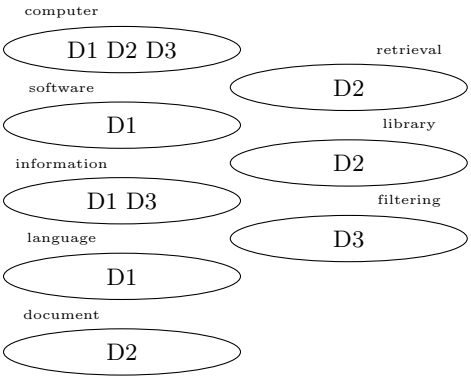
https://www.springboard.com/blog/introduction-word-embeddings/

Inverted Indices

Index



Inverted Index



Inverted indices are called like this because, in a sense, they do the inverse of a normal index. An index would indicate all the words listed in a document. In contrast, an inverted index indicates all the documents that contain a particular word. In the example above:

- index(D1) = { computer, language, software, information }.
- inverted_index(computer) = { D1, D2, D3 }.

3.2 Boolean Retrieval

Retrieval

- In the retrieval stage, the index is searched.
- This enables fast retrieval.
- Note that the index does not contain the full information from the documents.
- For example, searching a stop word will be useless.

Boolean Retrieval

- Use Boolean operations among the search terms.

  x AND y Documents that contain both terms.

  x OR y Documents that contain at least one term.

  NOT x Documents that do not contain the term.

- The use of inverted indices simplifies this method.

  x AND y Set intersection.

  x OR y Set union.

  NOT x Set complement.

Example of Boolean Retrieval

Keywords
D1:{computer, software, information, language}
D2:{computer, document, retrieval, library}
D3:{computer, information, filtering, retrieval}

Inverted Index
computer → {D1, D2, D3}, software → {D1}, information → {D1,D3},
language → {D1}, document → {D2}, retrieval → {D2, D3},
library → {D2}, filtering → {D3}

Boolean Query
(information OR document) AND retrieval

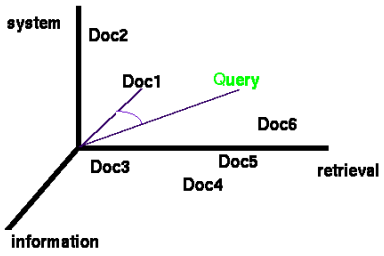
Result
({D1,D3} ∪ {D2}) ∩ {D2,D3} = {D2,D3}

3.3 Vector Retrieval

Vector Retrieval

Boolean retrieval and ranking

- There are no obvious methods to rank the results of Boolean retrieval.
  - Google introduced PageRank but we will see this later...
- An easy method to rank documents is to represent them as vectors and use well-established methods for vector comparison.



Vector Space Model

Template:

{*computer,software,information,document,retrieval,language,library,filtering*}

Initial documents
D1:{computer, software, information, language}
D2:{computer, document, retrieval, library}
D3:{computer, information, filtering, retrieval}

Document vectors

D1: (1,1,1,0,0,1,0,0)
D2: (1,0,0,1,1,0,1,0)
D3: (1,0,1,0,1,0,0,1)

Document matrix
(typically a **sparse matrix**)

$$D = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

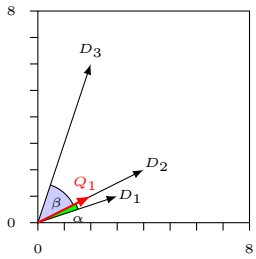
Information in the Vectors

- In the vector space model, each word in the vocabulary represents an element in a vector.
- The final document matrix will typically be **sparse** since a document will typically contain only a small fraction of all the possible words.
- Possible information to store in the vector:
  - The occurrence of a word/stem/n-gram (1) or not (0) ← as in our example.
  - The word frequency.
  - $tf.idf$ ← a popular choice.
  - Distributional semantics ← a hot research topic.

Cosine Similarity

Cosine Method

- Compare the cosine of the angle between vectors.
- If the angle is zero, then the cosine is 1.



$$\begin{aligned} \cos(D_1, Q_1) &= \cos(\alpha) \\ \cos(D_2, Q_1) &= \cos(0) = 1 \\ \cos(D_3, Q_1) &= \cos(\beta) \end{aligned}$$

Cosine Similarity: Formulas

General Formula

$$\cos(D_j, Q_k) = \frac{\Sigma_{i=1}^{N} D_{j,i} Q_{k,i}}{\sqrt{\Sigma_{i=1}^{N} D_{j,i}^2} \sqrt{\Sigma_{i=1}^{N} Q_{k,i}^2}} = \frac{D_j \cdot Q_k}{||D_j||_2 \, ||Q_k||_2}$$

If the vectors are normalised

$$\cos(D_j, Q_k) = \Sigma_{i=1}^{N} D_{j,i} Q_{k,i} = D_j \cdot Q_k$$

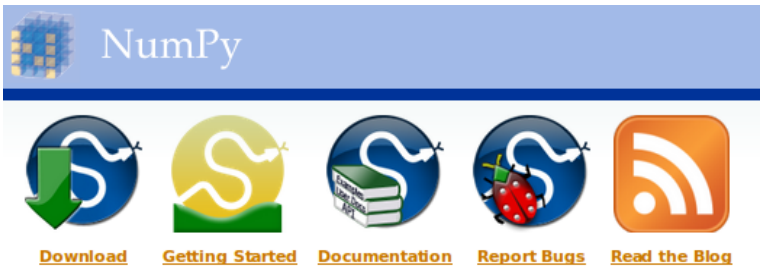
The terms at the rightmost side of the equations use vector operations: $A \cdot B$ is the scalar ("dot") product of vectors A and B, and $||A||$ is the Euclidean norm of A (also called l^2 norm).

3.4 Vector Retrieval in Python

Vectors and Matrices in Python

numpy

- Python's numpy is a collection of libraries that include manipulation of vectors and matrices.
- http://www.numpy.org/
- It's pre-loaded in the Anaconda distribution.



Manipulating Vectors

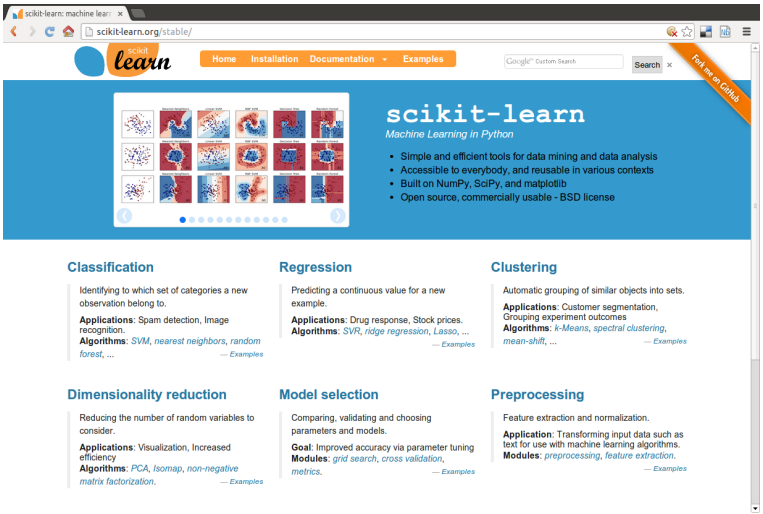
```
>>> import numpy as np
>>> a = np.array([1,2,3,4])
>>> a[0]
1
>>> a[1:3]          # slicing
array([2, 3])
>>> a+1             # add a constant to a vector
array([2, 3, 4, 5])
>>> b=np.array([2,3,4,5])
>>> a+b             # add two vectors
array([3, 5, 7, 9])
>>> a*b             # pairwise multiplication
array([ 2,  6, 12, 20])
>>> np.dot(a,b) # dot product between vectors, a . b
40
```

Manipulating Matrices

```
>>> x = np.array([[1,2,3],[4,5,6]])
>>> x
array([[1, 2, 3],
       [4, 5, 6]])
>>> y = np.array([[1,1,1],[2,2,2]])
>>> x+y                    # add two matrices
array([[2, 3, 4],
       [6, 7, 8]])
>>> x*y                    # pairwise multiplication
array([[ 1,  2,  3],
       [ 8, 10, 12]])
>>> x.T                    # transpose
array([[1, 4],
       [2, 5],
       [3, 6]])
>>> np.dot(x.T,y)  # dot product
array([[ 9,  9,  9],
       [12, 12, 12],
       [15, 15, 15]])
>>>
```

Scikit-learn

- http://scikit-learn.org/
- Incorporates an extensive set of machine learning algorithms into Python.
- It has a consistent and intuitive interface.
- The documentation is very complete.
- Includes generic tutorials on the main machine learning algorithms.
- It's pre-loaded in the Anaconda distribution.



tf.idf **with scikit-learn**

```
>>> import glob
>>> files = glob.glob('enron1/ham/*.txt')
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> tfidf = TfidfVectorizer(input='filename',stop_words='english')
>>> tfidf_values = tfidf.fit_transform(files)
>>> len(tfidf.get_feature_names())
19892
>>> tfidf.get_feature_names()[10000:10005]
['grandma', 'grandpa', 'grandsn', 'grandsons', 'grant']
>>> type(tfidf_values)
scipy.sparse.csr.csr_matrix
>>> type(tfidf_values.toarray())
numpy.ndarray
>>> tfidf_values.shape
(3672, 19892)
```

In this code, tfidf is an instance of the TfidfVectorizer class, which will accept a list of text files and will ignore stop words.

The tf.idf values are calculated and stored in the variable tfidf_values. This is done by calling the method fit_transform, which will return a sparse matrix. Scikit-learn provides many functions that operate with sparse matrices, so often we will not need to convert sparse matrices to regular arrays. If we need to convert a sparse matrix to an array we can use the "toarray" method.

The method "shape" returns the dimensions of the array or sparse matrix. In our case, the output of the function says that the variable tfidf_values has 3,672 rows (one row per file) and 19892 columns (one column per distinct word).

Normalised tf.idf and cosine similarity in Python

```
>>> tfidf_norm = TfidfVectorizer(input='filename',
                                 stop_words='english',
                                 norm='l2')
>>> tfidf_norm_values = tfidf_norm.fit_transform(files).toarray()
>>> import numpy as np
>>> def cosine_similarity(X,Y):
        return np.dot(X,Y)
>>> cosine_similarity(tfidf_norm_values[0,:],
                      tfidf_norm_values[1,:])
0.017317648885111028
```

This code produces normalised tf.idf so that the computation of the cosine similarity is faster. To normalise tf.idf we use the l^2 norm, which is the standard Euclidean norm.

Scikit-learn provides an implementation of cosine similarity. For details, look at the documentation page at http://scikit-learn.org/stable/modules/metrics.html.

Some Open Source Search Engines

If you don't want to implement your search engine from scratch, try these (http://www.mytechlogy.com/IT-blogs/8685/top-5-open-source-search-engines/):

1. Elasticsearch: https://www.elastic.co/
2. Solr: https://lucene.apache.org/solr/

3. Lucene: https://lucene.apache.org/
4. Sphinx: http://sphinxsearch.com/
5. Xapian: https://xapian.org/
6. Indri: https://www.lemurproject.org/
7. Zettair: http://www.seg.rmit.edu.au/zettair/

The following is a Python library that can be used for indexing and retrieving documents (among many other things):

1. Gensim: https://radimrehurek.com/gensim/

Take-home Messages

1. What is indexing? what is retrieval?
2. What is an inverted index?
3. Perform Boolean retrieval by hand.
4. Implement Boolean retrieval in Python.
5. Use sklearn to build a vector model with tf.idf.
6. Use sklearn to implement cosine similarity.

What's Next

Friday

- Web Search

Reading

- Brin and Page (1998) — a famous paper by the founders of Google.