COMP348 — Document Processing and the Semantic Web

Week 02 Lecture 1: Searching for Information

Diego Mollá

Department of Computer Science
Macquarie University

COMP348 2016H1

Programme

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.

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.

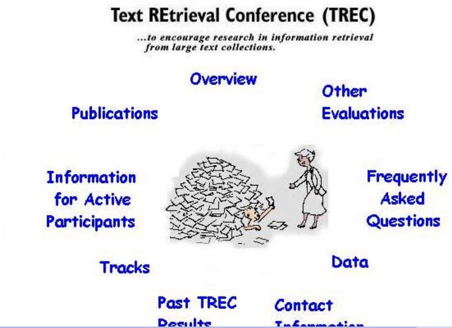
The Solution

Search engines.

Information Retrieval

Information Retrieval (IR)

- IR is about searching for the information.
- It typically means “text retrieval”.
- IR is the core technology behind Web search.



Text REtrieval Conference (TREC)

...to encourage research in information retrieval from large text collections.

TREC is a well-known US-funded competition-based research conference where all participants compete to produce the best IR system.

Stages in an IR System

1. Indexing
    - Done in an off-line stage.
    - 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).

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 of 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 evaluation, 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), this is what you calculate your statistics over.
- 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.

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 a period is the end of a sentence or not, the system can mistakenly classify abbreviations as end-of-sentence markers, or vice versa.
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

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

	actual case	
system	target	non target
selected	tp	fp
not selected	fn	tn

Precision and Recall

Formulas

- precision = tp / (tp + fp)
- recall = tb / (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	
system	target	non target
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,

$$F_\beta = (1 + \beta^2)\frac{\text{precision} \cdot \text{recall}}{\beta^2\text{precision} + \text{recall}}$$

- The most commonly used instance is when $\beta = 1$, referred to as F_1 :

$$F_1 = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

- This is just the harmonic mean of precision and recall.
- For previous example, $F_1 = 0.746$

Exercise

Another task with binary categories is the separating of spam from non-spam.

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?

Bag of Words Representation

Bag of words

- 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 can't be used to discriminate documents.
- Usually, important words are in the middle.

Zipf's Law for term frequency

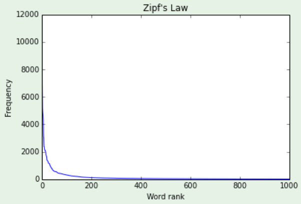
- A small percentage of words are very frequent.
- A large percentage of words have very little frequency.
- The relation approximates a Zipfian distribution.
- This is also referred as "long-tail" distribution.

Zipf's Law in Action

Python code

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

500 most frequent words



tf.idf

tf.idf

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

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

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

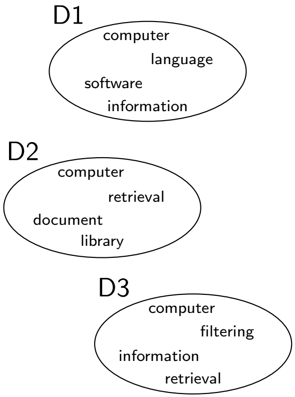
$$idf(t) = \log \frac{\text{number of documents}}{\text{number of documents that contain } t}$$

- *tf.idf* combines these two characteristics.

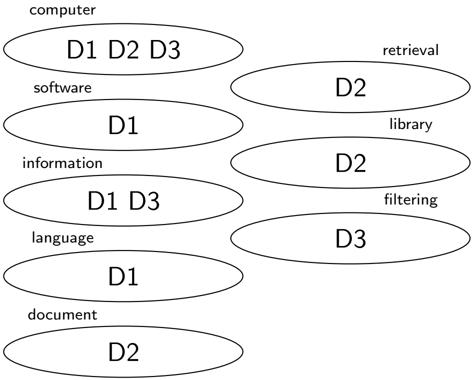
$$tf.idf(t,d) = tf(t,d) \times idf(t)$$

Inverted Indices

Index



Inverted Index



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 $\rightarrow$ {D1, D2, D3}, software $\rightarrow$ {D1}, information $\rightarrow$ {D1,D3},
language $\rightarrow$ {D1}, document $\rightarrow$ {D2}, retrieval $\rightarrow$ {D2, D3},
library $\rightarrow$ {D2}, filtering $\rightarrow$ {D3}

Boolean Query

(information OR document) AND retrieval

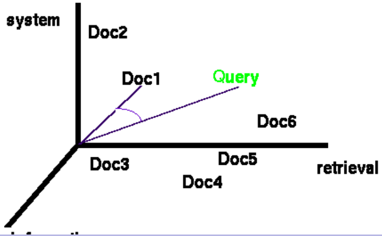
Result

$(\{D1,D3\} \cup \{D2\}) \cap \{D2,D3\} = \{D2,D3\}$

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 functions 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**)

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

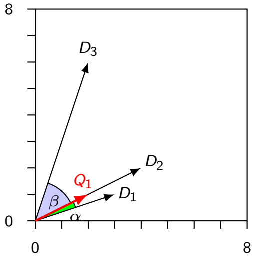
Information in the Vectors

- In the vector space model, each word in the bag of words 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 (1) or not (0) $\leftarrow$ as in our example.
  - The word frequency.
  - *tf.idf* $\leftarrow$ a popular choice.

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

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

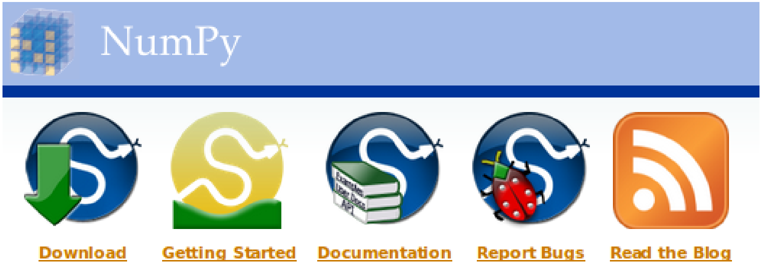
If the vectors are normalised

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

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 part of the scipy package `http://scipy.org/`



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
```

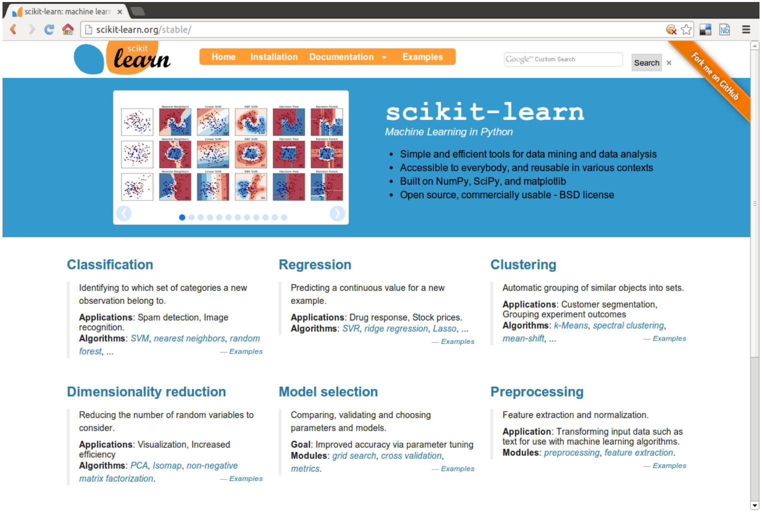
Manipulating Matrices

```
>>> 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],
```

Scikit-learn I

- `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.

Scikit-learn II



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which set of categories a new observation belong to.

Applications: Spam detection, Image recognition.
Algorithms: *SVM*, *nearest neighbors*, *random forest*, ... — *Examples*

Regression

Predicting a continuous value for a new example.

Applications: Drug response, Stock prices.
Algorithms: *SVR*, *ridge regression*, *Lasso*, ... — *Examples*

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: *k-Means*, *spectral clustering*, *mean-shift*, ... — *Examples*

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: *PCA*, *Isomap*, *non-negative matrix factorization*. — *Examples*

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: *grid search*, *cross validation*, *metrics*. — *Examples*

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: *preprocessing*, *feature extraction*. — *Examples*

tf.idf with scikit-learn

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

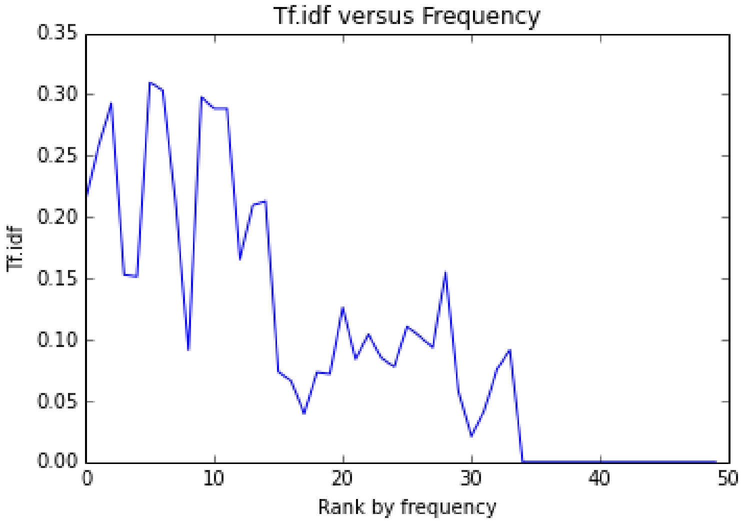
Normalised tf.idf and cosine similarity in Python

```
>>> tfidf_norm = TfidfVectorizer(input='filename',
                                 stop_words='english',
                                  norm='l2')
>>> tfidf_norm_values = tfidf_norm.fit_transform(files).toarr
>>> 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
```

tf.idf *versus* frequency I

```python
import glob
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from matplotlib import pyplot as plt
files = glob.glob('enron1/ham/*.txt')
tfidf = TfidfVectorizer(input='filename',stop_words='english')
tfidf_array = tfidf.fit_transform(files).toarray()
count = CountVectorizer(input='filename',stop_words='english')
count_array = count.fit_transform(files).toarray()
count_words = count.get_feature_names()
sorted_words = sorted(count_words,
                      key=lambda x: count_array[0,count_words.index(x)],
                      reverse = True)
tfidf_words = tfidf.get_feature_names()
data_tfidf = [tfidf_array[0,tfidf_words.index(x)]
              for x in sorted_words]
plt.plot(data_tfidf[:50])
plt.show()
```

tf.idf *versus* frequency II



tf.idf *versus* frequency, take II

```
import glob
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from matplotlib import pyplot as plt
files = glob.glob('enron1/ham/*.txt')
tfidf = TfidfVectorizer(input='filename',stop_words='english')
tfidf.fit(files)
tfidf_doc1 = tfidf.transform([files[0]]).toarray()[0,:]
count = CountVectorizer(input='filename',stop_words='english')
count.fit(files)
count_doc1 = count.transform([files[0]]).toarray()[0,:]
count_words = count.get_feature_names()
sorted_words = sorted(count_words,
                      key=lambda x: count_doc1[count_words.index(x)],
                      reverse = True)
tfidf_words = tfidf.get_feature_names()
data_tfidf = [tfidf_doc1[tfidf_words.index(x)]
              for x in sorted_words]
plt.plot(data_tfidf[:50])
plt.show()
```

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.