Information Retrieval and Applications
Knowledge Discovery and Data Mining 2 (VU) (707.004)

Roman Kern, Heimo Gursch

KTI, TU Graz

2015-03-19
Outline

1. Basics
   - Inverted Index
   - Probabilistic IR

2. Advanced Concepts
   - Latent Semantic Indexing
   - Relevance Feedback
   - Query Processing

3. Evaluation

4. Applications of IR
   - Classification
   - Content-based Recommender systems

5. Solutions
   - Open Source Solutions
   - Commercial Solutions
Basics

Inverted Index, Text-Preprocessing, Language-based & Probabilistic IR
Definition

Information Retrieval (IR) is **finding material** (usually documents) of an **unstructured** nature (usually text) that satisfies an **information need** from within **large collections** (usually stored on computers).

The term "term" is very common in Information Retrieval and typically refers to single words (also common: bi-grams, phrases, ...)
Assumptions

Basic assumptions of Information Retrieval

**Collection**  Fixed set of documents

**Goal**  Retrieve documents with information that is relevant to user’s information need and help her complete a task.
Functional requirements of an index

- Search for single keyword
- Search for multiple keywords
- Search for phrases, i.e. keywords in particular order
- Boolean combination of keywords (AND, OR, NOT)
- Rank results by number of keyword occurrences
- Find the positions of keywords in a document
- Compose snippets of text around the keyword positions

Motivation

How can this goals be achieved?
Structured, Semi-Structured Data

Structured data
- Format and type of information is known
- IR task: normalize representations between different sources
- Example: SQL databases

Semi-structured data
- Combination of structured and unstructured data
- IR task: extract information from the unstructured part and combine it with the structured information
- Example: email, document with meta-data
Unstructured data

- Format of information is not known, information is hidden in (human-readable) text
- IR task: extract information at all
- Example: blog-posts, Wikipedia articles

This lecture focuses on unstructured data.
Simple Approach---Sequential Search

**Advantage** Simple

**Disadvantages** Slow and hard to scale

**Usage**
- Small texts
- Frequently changing text
- Not enough space for index available

**Example** `grep` on the UNIX command line

Not feasible for large texts. A different approach is needed. But how?
Overview Indexed Searching

Split process

**Indexing** Convert all input documents into a data-structure suitable for fast searching

**Searching** Take an input query and map it onto the pre-processed data-structure to retrieve the matching documents

Inverted index

- Reverses the relationship between documents and contained terms (words)
- Central data-structure of a search engine
Information Stored in an Inverted Index

**Dictionary**
Sorted List of all terms found in all documents

**Postings**
Information about the occurrence of a term, consisting of:
- **Document (DocID)** Document identifier, i.e. in which document the term was found
- **Position(s) (Pos)** Location(s) of the term in the original document
- **Term Frequency (TF)** How often the term occurs in the document

The term frequencies could also be calculated by counting the position entries. As computing time is more precious than memory, the term frequencies are stored as well.
Example - Documents

Document 1
Sepp was stopping off in Hawaii. Sepp is a co-worker of Heidi.

Document 2
Heidi stopped going to Hawaii. Heidi is a co-worker of Sepp.
## Example - Index

<table>
<thead>
<tr>
<th>Dictionary</th>
<th>Postings</th>
</tr>
</thead>
<tbody>
<tr>
<td>co-worker</td>
<td>(Doc:1; TF:1; Pos:10); (Doc:2; TF:1; Pos:9)</td>
</tr>
<tr>
<td>going</td>
<td>(Doc:2; TF:1; Pos:3)</td>
</tr>
<tr>
<td>Hawaii</td>
<td>(Doc:1; TF:1; Pos:6); (Doc:2; TF:1; Pos:5)</td>
</tr>
<tr>
<td>Heidi</td>
<td>(Doc:1; TF:1; Pos:12); (Doc:2; TF:2; Pos:1,6)</td>
</tr>
<tr>
<td>Seppl</td>
<td>(Doc:1; TF:2; Pos:1,7); (Doc:2; TF:1; Pos:11)</td>
</tr>
<tr>
<td>stopped</td>
<td>(Doc:2; TF:1; Pos:2)</td>
</tr>
<tr>
<td>stopping off</td>
<td>(Doc1; TF:1; Pos 3)</td>
</tr>
</tbody>
</table>
Properties

- Search for terms
- Formulate boolean queries
- Search for phrases (i.e. terms in a particular order)
- Information where the term occurred in the document
- Possible creation of snippets
- Documents can be ranked by the number of terms they contain

Keep in mind that an inverted index

- ... takes up additional space, not only storing the original documents, but also the inverted index
- ... must be kept up to date
To make a text indexable, a couple of steps are necessary:

1. Detect the language of the text
2. Detect sentence boundaries
3. Detect term (word) boundaries
4. Stemming and normalization
5. Remove stop words
Language Detection

Letter Frequencies

<table>
<thead>
<tr>
<th>Letter</th>
<th>Percentage of occurrence English</th>
<th>Percentage of occurrence German</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>8.17</td>
<td>6.51</td>
</tr>
<tr>
<td>E</td>
<td>12.70</td>
<td>17.40</td>
</tr>
<tr>
<td>I</td>
<td>6.97</td>
<td>7.55</td>
</tr>
<tr>
<td>O</td>
<td>7.51</td>
<td>2.51</td>
</tr>
<tr>
<td>U</td>
<td>2.76</td>
<td>4.35</td>
</tr>
</tbody>
</table>

N-Gram Frequencies

Better---more elaborate---methods use statistics over more than one letter, e.g. statistics over two, three or even more consecutive letters.
Sentence Detection

First approach
Every period marks the end of a sentence

Problem
Periods also mark abbreviations, decimal points, email-addresses, etc.

Utilize other features
- Is the next letter capitalized?
- Is the term in front of the period a known abbreviation?
- Is the period surround by digits?
- ...

Roman Kern, Heimo Gursch (KTI, TU Graz)
Tokenization -- Problem

Goal

- Split sentences into tokens
- Throw away punctuations

Possible Pit Falls

- White spaces are no safe delimiter for word boundaries (e.g. New York)
- Non-alphanumerical characters can separate words, but need not (e.g. co-worker)
- Different forms (e.g. white space vs. whitespace)
- German compound nouns (e.g. Donaudampfschiffahrtsgeellschaft, meaning Danube Steamboat Shipping Company)
Tokenization -- Solutions

Supervised Learning
- Train a model with annotated training data, use the trained model to tokenize unknown text
- Hidden-Markov-Models and conditional random fields are commonly used

Dictionary Approach
- Build a dictionary (i.e. list) of tokens
- Go over the sentence and always take the longest fitting token (greedy algorithm!)
- Remark: Works well for Asian languages without white spaces and short words. Problematic for European languages
Normalization

Some definitions

**Token**  Sequence of characters representing a useful semantic unit, instance of a type

**Type**  Common concept for tokens, element of the vocabulary

**Term**  Type that is stored in the dictionary of an index

Task

1. Map all possible tokens to the corresponding type
2. Store all representing terms of one type in the dictionary
Example
Posible Ways for Normalization

- Ignore cases
- Rule based removal of hyphens, periods, white spaces, accents, diacritics (Be aware of possible different meaning after removal)
- Provide a list of all possible representations of a type
- Map different spelling version by using a phonetic algorithm. Phonetic algorithms translate the spelling into their pronunciation equivalent (e.g. Double Metaphone, Soundex).
Stemming & Lemmatization

Definition

Goal of both  Reduce different grammatical forms of a word to their common infinitive

Stemming  Usage of heuristics to chop off / replace last part of words (Example: Porter's algorithm).

Lemmatization  Usage of proper grammatical rules to recreate the infinitive.

\(^a\)The common infinitive is the form of a word how it is written in a dictionary. This form is called lemma, hence the name *Lemmatization*.

Examples

- Map *do, doing, done*, to common infinitive *do*
- Map *digitalizing, digitalized* to digital
Stop Words

Definition  Extremely common words that appear in nearly every text

Problem  As stop words are so common, their occurrence does not characterize a text

Solution  Just drop them, i.e. do not put them in the index directory

Common stop words

- Write a list with stop words (List might be topic specific!)
- Usual suspects: articles \((a, an, the)\), conjunction \((and, or, but, \ldots)\), pre- and postposition \((in, for, from)\), etc.

Solution

Stop word list  Ignore word that are given on a list (black list)

Problem  Special names and phrases \((The Who, Let It Be, \ldots)\)

Solution  Make another list... (white list)
Simple / Traditional Information Retrieval

Bag of Words Model

Basic Idea

Just count occurrences of search terms

Term Frequency - Inverse Document Frequency (TF-IDF)

Document is relevant if

- The document contains a search term very often
- Other documents contain the search term very rarely

TF-IDF

\[
 tf-idf = tf \cdot idf = tf \cdot \log \frac{N}{d}
\]

with

- \( tf \): Number of term occurrences in current document
- \( N \): Number of all documents
- \( d \): Number of documents containing the term
Information flow

The classic search model

![Diagram of the classic search model]

- Task
- Info Need
- Verbal form
- Mis-conception
- Mis-translation
- Mis-formulation
- Query
- SEARCH ENGINE
- Results
- Corpus

Find this: mouse trap [any language] Search
Basics

Probabilistic IR

Probabilistic Information Retrial

Every transformation can hold a possible information loss

1. Task to query
2. Query to document

- Information loss leads to uncertainty
- Probability theory can deal with uncertainty

Basic idea behind probabilistic IR

- Model the uncertainty, which originates in the imperfect transformations
- Use this uncertainty as a measurement how relevant a document is, given a certain query
Probabilistic Information Retrieval Models

Binary Independence Model

- **Binary**: documents and queries represented as binary term incidence vectors
- **Independence**: terms are assumed to be independent of each other

Okapi BM25

- Takes number of term frequencies and document length into account
- Parameters to tune influence of term frequencies and document length

Bayesian networks for text retrieval

Language Models Coming now...
Language Models

Basic Idea

- Train a probabilistic model $M_d$ for document $d$, i.e. as many trained models as documents
- $M_d$ ranks documents on how possible it is that the user's query $q$ was created by the document $d$
- Results are document models (or documents, respectively) which have a high probability to generate the user's query
Examples for Language Models

Successive Probability of query terms

\[ P(t_1 t_2 \ldots t_n) = P(t_1)P(t_2|t_1)P(t_3|t_1t_2)\ldots P(t_n|t_1 \ldots t_{n-1}) \]

Unigram Language Model

\[ P_{uni}(t_1 t_2 \ldots t_n) = P(t_1)P(t_2)\ldots P(t_n) \]

Bigram Language Model

\[ P(t_1 t_2 \ldots t_n) = P(t_1)P(t_2|t_1)P(t_3|t_2)\ldots P(t_n|t_{n-1}) \]

Probability \( P(t_n) \) Probabilities are generated from the term occurrence in the document in question

And many more... A lot more, and more complex models are available
Advanced Concepts

LSA, Relevance Feedback, Query Processing, ...
Index Construction

Going beyond simple indexing

- Term-document matrices are very large
- But the number of **topics** that people talk about is small (in some sense)
  - Clothes, movies, politics, ...
- Can we represent the term-document space by a lower dimensional latent space?
Latent Semantic Indexing (LSI)

- is based on Latent Semantic Analysis (see KDDM1), which
- is based on Singular Value Decomposition (SVD), which
- creates a new space (new orthonormal base), which
- allows to be mapped to lower dimensions, which
- might improve the search performance
Singular Value Decomposition

SVD overview

- For an $m \times n$ matrix $A$ of rank $r$ there exists a factorization (Singular Value Decomposition = SVD) as follows:
  - $M = U\Sigma V^T$
  - $U$ is an $m \times m$ unitary matrix
  - $\Sigma$ is an $m \times n$ diagonal matrix
  - $V^T$ is an $n \times n$ unitary matrix

- The columns of $U$ are orthogonal eigenvectors of $AA^T$
- The columns of $V$ are orthogonal eigenvectors of $A^TA$
- Eigenvalues $\lambda_1 \ldots \lambda_r$ of $AA^T$ are the eigenvalues of $A^TA$. 
Singular Value Decomposition

Matrix Decomposition

- SVD enables lossy compression of a term-document matrix
  - Reduces the dimensionality or the rank
  - Arbitrarily reduce the dimensionality by putting zeros in the bottom right of sigma
  - This is a mathematically optimal way of reducing dimensions
Singular Value Decomposition

Reduced SVD
- If we retain only $k$ singular values, and set the rest to 0
- Then $\Sigma$ is $k \times k$, $U$ is $M \times k$, $V^T$ is $k \times n$, and $A_k$ is $m \times n$
- This is referred to as the **reduced SVD**
- It is the convenient (space-saving) and usual form for computational applications

Approximation Error
- How good (bad) is this approximation?
- It's the best possible, measured by the Frobenius norm of the error
Latent Semantic Indexing

Application of SVD - LSI
- From term-document matrix $A$, we compute the approximation $A_k$.
- There is a row for each term and a column for each document in $A_k$.
- Thus documents live in a space of $k << r$ dimensions (these dimensions are not the original axes).
- Each row and column of $A$ gets mapped into the $k$-dimensional LSI space, by the SVD.
- Claim – this is not only the mapping with the "best" approximation to $A$, but in fact improves retrieval.
Latent Semantic Indexing

Application of SVD - LSI

- A query $q$ is also mapped into this space
- ... within this space the query $q$ is compared to all the documents
- ... the document closest to the query are returned as search result

LSI - Results

- Similar terms map to similar location in low dimensional space
- Noise reduction by dimension reduction
Relevance Feedback

Integrate feedback from the user

- Given a search result for a user's query
- ... allow the user to rate the retrieved results (good vs. not-so-good match)
- This information can then be used to re-rank the results to match the user's preference
- Often automatically inferred from the user's behaviour using the search query logs
- ... and ultimately improve the search experience

The query logs (optimally) contain the queries plus the items the user has clicked on (the user's session)
Relevance Feedback

**Pseudo relevance feedback**

- No user interaction is needed for the pseudo relevance feedback
- ... the first n results are simply assumed to be a good match
- As if the users has clicked on the first results and marked them as good match
- → often improves the quality of the search results

Pseudo relevance feedback is sometimes also called blind relevance feedback
Query Reformulation

Modify the query, during of after the user interaction

- Query suggestion & completion
- (Automatic) query correction
- (Automatic) query expansion
Query Suggestion

- The user starts typing a query
- ... automatically gets suggestions
- → to complete the currently typed word
- → to complete a whole phrase (e.g. the next word)
- → suggest related queries
Query Correction

- The user entered a query, which may contain spelling errors (e.g. Britni Spears)
- ... automatically correct or suggest possible rectified queries
- ... also known as Query Reformulation
- → use other users behaviour (harvest query logs)
- → use the frequency of terms found in the indexed documents (and the similarity with the entered words)

Notebook with various distances (Levenshtein being the best known): http://nbviewer.ipython.org/gist/MajorGressingham/7691723
Query Expansion

- The user has entered a query
- ... automatically add (related) words to the query
- → Global query expansion - only use the query (plus corpus or background knowledge)
- → Local query expansion - conduct the search and analyse the search results
Global Query Expansion

- Uses only the query
- Example: Use of a thesaurus
  - Query: [buy car]
  - Use synonyms from the thesaurus
  - Expanded query: [(buy OR purchase) (car OR auto)]
Local query expansion

- Uses the query plus the search results
- ... conceptually similar to the pseudo relevance feedback
- Look for terms in the first \( n \) search results and add them to the query
- Re-run the query with the added terms
A common feature of search engines is the more like this search

- Given a search results
- ... the user wants items similar to one of the presented results
- This can be seen as: taking a document as query
- → which is a common implementation (but restricting the query to the most discriminative terms)
Evaluation

Assess the quality of search
Overview: Evaluation

Two basic questions / goals

- Are the retrieved document relevant for the user's information need?
- Have all relevant document been retrieved?

In practice these two goal contradict each other.
Evaluation

Precision & Recall

Evaluation Scenario

- Fixed document collection
- Number of queries
- Set of documents marked as relevant for each query

Main Evaluation Metrics

- Precision = \( \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} \)
- Recall = \( \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} \)
- \( F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \)
Precision & Recall
Advanced Evaluation Measures

**Mean Average Precision - MAP**

- ... mean of all average precision of multiple queries
- \( Map = \frac{1}{|Q|} \sum_{q \in Q} AP_q \)
- \( AP_q = \frac{1}{n} \sum_{k=1}^{n} Precision(k) \)

**(Normalised) discounted cumulative gain (nDCG)**

- ... where there is a reference ranking the relevant results
Selected Applications of IR
Classification & Recommender Systems
Overview

Classification

- Supervised Learning: labelled training data is needed (effort for labelling)
- Classification problems: classify an example into a given set of categories
- Algorithm learns model from the labelled training set and applies this model on unknown data
- Used in real applications

Clustering

- Unsupervised Learning
- Find groups in the data, where the similarity within the group is high and the similarity between groups is low
- Hardly ever used in real text retrieval applications
Document Classification by k-Nearest-Neighbour (kNN) Algorithm

1. For each desired class: at least one document must be provided
2. Find $k$ nearest neighbours of a document:
   1. Take all terms of a document
   2. Combine them to a query (e.g. with a logical OR)
   3. Take the first $k$ results from an index
3. Assign the unknown document to the class with the most neighbours in it
4. Goto 2

Description

**kNN Algorithm**  Outer steps 1 to 4

**Adaptation to IR**  Inner steps 1 to 3
Recommender systems - Overview

- Recommender systems should provide usable suggestion to users
- Recommender systems should show new, unknown---but similar---documents
- Recommender systems can utilize different information sources
  - **Content-based Recommender**
    - Finds similar documents on the basis of document content
  - **Collaborative Filtering**
    - Employs user rating to find new and useful documents
  - **Knowledge-based Recommender**
    - Makes decisions based on pre-programmed rules
  - **Hybrid Recommender**
    - Combination of two or three other approaches
Types of Content-based Recommender

Document-to-Document Recommender

Document-to-User Recommender
Building a Recommender by Using an Index

Document-to-Document Recommender

1. Take the terms from a document
2. Construct a (weighted) query with them
3. Query the index
4. Results are possible candidates for recommending

Document-to-User Recommender

1. Create a user model from
   - Terms typed by the user
   - Documents viewed by the user
   - \( \Rightarrow \) Simplest user model is just a set of terms
2. Construct a (weighted) query with them
3. Query the index
4. Results are possible candidates for recommending
Solutions
Open-source and commercial IR solutions
Open Source Search Engine

Apache Lucene

- Very popular, high quality
- Backbone of many other search engines, e.g. Apache Solr, ElasticSearch
  - Lucene is the core search engine
  - Solr provides a service & configuration infrastructure (plus a few extra features)
- Implemented in Java, bindings for many other programming languages
Other Open-Source Search Engines

**Xapian**
- Support for many (programming) languages
- Used by many open-source projects

**Lemur & Indri**
- Language models
- Academic background

**Terrier**
- Many algorithms
- Academic background

**Whoosh**
- for Pythonistas

**Sphinx**
- Closely related to DBMSs
What customers want

Indexing pipeline

And also: user interface, access control mechanisms, logging, etc.
Commercial Search Engines

**Autonomy**
- Grantees not to truncate documents
- Offers a wide range on source connectivity

**Sinequa**
- Expert Search
- Creates new views, i.e. do not only show result list, but combine results to new unit

**Attivio**
- Use SQL statements in queries

**IntraFind**
- Lucene-based commercial search solution

And many, many more...
The End

Next: Preprocessing Platform & Practical Applications