HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

MSE | MASTER OF SCIENCE IN ENGINEERING

Recommender-System with Text Analysis for improved Geo-Discovery

Seminar Thesis

Master of Science in Engineering
Department of Informatics
HSR Hochschule für Technik Rapperswil

Author: Fabian Senn

Advisor: Prof. Stefan F. Keller

Date: December 20, 2013

Contents

1 Introduction

1.1 Problem Statement

Online platforms such as Geocat [1] provide a huge collection of geospatial metadata (geometadata). They deliver to their users geographic information descriptions about data published by federal offices and private enterprises. The metadata is stored according to a structured data model and communicated through a standardized exchange format. Due to the increasing amount of data available in the collection, finding documents in traditional vector space models as provided by Lucene [2] and Solr [3] becomes more and more difficult. Users must know before hand the keywords used in the requested documents. Related documents not including any of these words are not displayed, even if they are about the same subject. The reason for this is that tools like Lucene or Solr usually don't consider synonyms.

This work ist based on the ideas of Vockner et al. [4]. They suggest a system for supporting search on geospatial metadata collections. Next to the main search application, they included Easyrec as an external recommendation system for displaying search results other users have viewed. Thus, it is the goal to enhance accessibility and discovery of geospatial data (geodata).

Furthermore, Vockner et al. also discussed a way how a recommender system for documents can skip the long initial phase of data acquisition. They proposed a content-based filtering approach. For this, they computed the document similarities by a concept called Latent Semantic Indexing. Thereafter, the similarities were imported as external rules into the Easyrec recommendation system.

1.2 Structure of this work

This paper is divided into 3 main chapters. It starts with the explanation of the Latent Semantic Indexing algorithm. Besides describing its essential parts, there is also a section about building a working example. Next, two forms of recommendation systems called collaborative filtering and content-based filtering are explained. Finally, it will be showed how the mentioned components can be used along with a search system for enhancing user experience.

2 Latent Semantic Indexing

2.1 Prerequisite: Vector Space Model

Vector space model is a very common form of document indexing. It is used in popular libraries like Lucene [2].

2.1.1 Model Evaluation

As the name implies, the document corpus in vector space modelling transformed into a vector space. Each word occurring in the corpus is represented by a dimension. Documents can then be described as vectors from this space. Terms present in a document have a non-zero value in the corresponding vector element. All other elements are set to zero.

The concept can easily be showed in an example. It consists of the three documents listed below

- d_1:Head First Java
- d_2:Java in a Nutshell
- d_3:Java: An introduction into Java

This corpus will lead to the following document vectors d_1, d_2 and d_3. As a note, the words *in* and *into* as well as *a* and *An* are treated as a single word.

	d_1	d_2	d_3
Head	1	0	0
First	1	0	0
Java	1	1	2
in	0	1	1
a	0	1	1
Nutshell	0	1	0
introduction	0	0	1

In the simplest variation of the vector space model like in the example before, the weight of the words in a document vector is equal to their frequency in the document. A word appearing more often in a document is therefore more important. Although this assumption is true in the most cases, it does not consider that words widely used among the corpus do not help distinguishing the documents. A word that appears less frequently should therefore receive a higher weight. This problem addressed by the Tf-Idf weighting. The Tf-Idf weight is calculated for each word. It is a product of the following two values.

tf stands for term frequency. Its value is computed by counting the frequency of a term in a document.

idf stands for inverse document frequency. The document frequency is the number of documents in which the term is present. To penalise words which appear frequently, the value is inverted.

$$idf(t) = 1 + log\left(\frac{numDocs}{docFreq + 1}\right) \tag{2.1}$$

The same example as above produces the following Tf-Idf vectors. It can be seen that the word *Java* receives a lower weight. This is due to its appearance in all documents.

	d_1	d_2	d_3
Head	1.18	0	0
First	1.18	0	0
Java	0.88	0.88	1.75
in	0	1	1
a	0	1	1
Nutshell	0	1.18	0
introduction	0	0	1.18

2.1.2 Query functionality

Query strings are converted to vectors using the same method as for mapping the documents. Due to the manual creation of queries however, it is possible that a requested keyword is not known by the document corpus. This is the case if the the word is not present in any of the documents. Thus, the word must be deleted from the query string.

After computing the query vector, it is placed into the vector space. The system now calculates a similarity value for each document and returns the ones with highest score. The most common used method form of similarity is the cosine similarity. In this method, the angles between the query and the document vectors are computed. The smaller the angle between the vectors, the higher is the similarity value.

2.1.3 Limitations

The vector space model lacks one important feature - the semantic sensitivity. The model only recognizes terms and their occurrence in documents. However, it does not identify the subject of a document. Therefore, it is not able to recognize synonyms. As an example, if a user searches for documents about football but instead queries with the synonym "soccer", the search engine using a vector space model will only retrieve documents including the word "soccer".

Today, this can be bypassed by including synonym lists. With aid of these, terms in documents can be extended with all possible variations. Although this simple approach improves synonym handling, creating the list manually can be very time consuming.

2.2 Latent Semantic Indexing (LSI)

Latent semantic indexing, also named as latent semantic analysis (LSA), is an extension to the traditional vector based document representation. It tries to overcome the existing problems of the vector space model by identifying semantic association between words. This can improve synonym handling.

Given a document-term matrix A from a preprocessed document corpus and a value k, the LSI-algorithm evaluates a matrix A_k whose rank is equal to k. This process is called singular value decomposition. Section 2.2.1.3 explains the method more in detail.

More intuitively spoken, the algorithm tries to find semantic relationships between words. This means, terms that are used several times in the same context tend to be highly correlated to each other. Correlations found in this manner are subsequently collapsed into a single dimension in the vector space. This process is done when k dimensions are found.

2.2.1 Model Evaluation

The process for building the LSI index is shown in the following diagram:

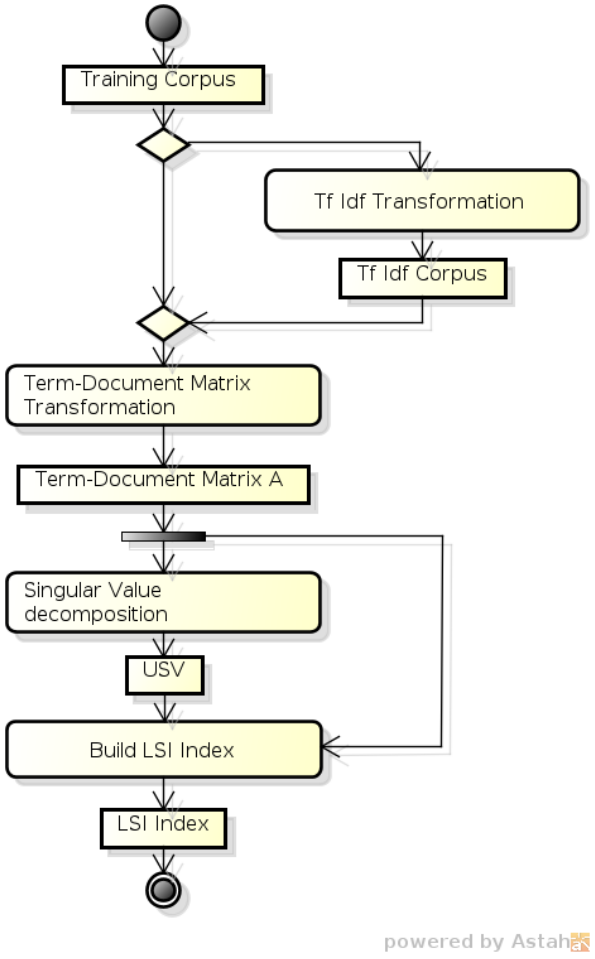


Figure 2.1: LSI Model Evaluation

2.2.1.1 Tf-Idf Transformation

In this step, the documents are transformed into the vector space model. This is done by introducing a vector space R^n whereat every term in the corpus represents an axis. For each document, a vector is crafted whose elements are the Tf-Idf weight of the terms included in the document.

This step is optional. It is also possible to simply use the term frequency values. However, converting the corpus into Tf-Idf space performs better, due to the superior weighting system.

2.2.1.2 Term-Document Matrix Transformation

The document corpus is now transformed into a term-document matrix A with $m \times n$ values. The m rows represent the unique terms used in the corpus. The n columns stand for each different document.

$$A = \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \dots & x_{m,n} \end{bmatrix}$$

2.2.1.3 Singular Value decomposition

With the aid of singular value decomposition (SVD), the number of rows and therefore the number of dimensions is reduced. This is achieved, by evaluating which words relate strongly to each other. Terms with high correlation share afterwards the same axis.

In the process of singular value decomposition the goal is to factorize the original matrix A into 3 subsequent matrices. The following formula expresses the factorization.

$$A = USV^T \tag{2.2}$$

The matrix U is called the term-concept matrix. It is of the size of $m \times r$ where r is the rank of the original matrix A. S is the matrix of singular values and satisfies following restrictions $S_{1,1} \geq S_{2,2} \geq ... \geq S_{r,r}\ and\ S_{i,j} = 0\ where\ i \neq j$. V is the computed $n \times r$ document-concept matrix. Both the matrices U and V are unitary. Meaning, multiplying them by their transposed matrix yields the identity matrix I_r ($U * U^T = I_r$ and $V * V^T = I_r$).

In a further step, both the ranks of the matrices S and U are reduced by k. This is done by removing columns and rows so that it reveals a $k \times k$ matrix S_k and a $m \times k$ matrix U_k.

2.2.1.4 Build LSI Index

After evaluation of S_k and U_k, the original term-document matrix A can be used for building a document index. For this, every single document vector of A is turned into a LSI vector. Responsible for that is the following formula:

$$D_k = A^T U_k S_k^{-1} \tag{2.3}$$

D_k a $n \times k$ matrix including the original n documents in their LSI representation as k dimensional vectors.

2.2.2 Updating the Index

If new documents are available, they can be added with a similar formula as presented above.

$$d_k = dU_k S_k^{-1} \tag{2.4}$$

The document vector in its Tf-Idf representation d is turned into a LSI vector d_k by multiplying it by $U_k S_k^{-1}$. After the transformation, the result is appended to the index D_k. This process is called *folding*.

While this works well if the document is similar to the ones in the index, problems arise if the data introduces new words. Words that were not present during the training phase are not recognised later. Therefore, the only solution is to ignore them.

Reevaluating the model with integration of the newly available data can resolve this problem. As such, the whole index must be changed. Thus, all previously calculated similarities must be updated.

2.2.3 Query process

Similar to the vector space model, queries in the LSI space are expressed as vectors. A query must therefore first be transformed into LSI space. This process is done in the same manner as stated in the equotation 2.4

$$q_k = qU_k S_k^{-1} \tag{2.5}$$

The query vector q must have the same rank as A. Thus, unknown terms are deleted.

2.2.4 Challenges

2.2.4.1 Performance

Calculating the singular value decomposition is very CPU intensive. This is due to the fact that a corpus can easily hold more than 10000 documents. Moreover, the number of different terms can reach a huge extent. As an example, the English Wikipedia comprises about 3.9 millions documents and 100000 different terms.

The computational complexity of the original algorithm as proposed by Golub and Reinsch [5] is:

$$\mathcal{O}(4m^2n + 8mn^2 + 9n^3) \tag{2.6}$$

Where m and n are the dimensions of the $m \times n$ Input matrix. Modern implementations reduce the complexity by calculating only the first k singular values.

2.2.4.2 Defining k

Another challenge is to find the optimal amount of topics possibly present in the document corpus. The fewer dimensions are requested, the broader become the identified concepts. On the other hand, a higher value for k will lead to more specific topics.

As a standard value, a value of 300 is proposed. If a huge amount of documents are present, setting k to 400 can be considered [6].

2.2.5 Limitation

In the contrary to the traditional indexing variants like the vector space model. The LSI is intended to be queried with documents. Single terms query are therefore not as effective as before. Also, features like boolean queries or negations of query words are not possible. For example, one could not query for documents which include the term *football* but not the term *american*.

2.2.6 Example

2.2.6.1 Building the model

In this example 10 books are to be indexed with the LSI algorithm. The books are chosen from 3 different topics - C#, Java and programming designs. The titles are below:

- d_1:**Head First Java**, 2nd **Edition**
- d_2:**Introduction** to Programming Using **Java**, Sixth **Edition**
- d_3:**Java** in a **Nutshell**
- d_4:**Head First C#**
- d_5:**C# 5.0** in a **Nutshell**: The definitive Reference
- d_6:Pro **C# 5.0** and the .NET 4.5 Framework
- d_7:**Head First Design Patterns**
- d_8:**Design Patterns**: Element of Reusable **Object-Oriented** Software
- d_9:Applying UML and **Patterns**: An **Introduction** to **Object-Oriented Analysis** and **Design** and Iterative Development
- d_{10}:**Head First Object-Oriented Analysis** and **Design**

Now, the term frequencies are calculated. For this, stop words and terms only occurring once are removed. The outcome of this is listed in the following table.

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}
Analysis	0	0	0	0	0	0	0	0	1	1
C#	0	0	0	1	1	1	0	0	0	0
Design	0	0	0	0	0	0	1	1	1	1
Edition	1	0	1	0	0	0	0	0	0	0
First	1	0	0	1	0	0	1	0	0	1
Head	1	0	0	1	0	0	1	0	0	1
Introduction	0	0	1	0	0	0	0	0	1	0
Java	1	1	1	0	0	0	0	0	0	0
Nutshell	0	1	0	0	1	0	0	0	0	0
Object	0	0	0	0	0	0	0	1	1	1
Oriented	0	0	0	0	0	0	0	1	1	1
Pattern	0	0	0	0	0	0	1	1	1	0
5.0	0	0	0	0	0	1	1	0	0	0

Next, the Tf-Idf weights are calculated. This is the term-document matrix A

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.1 & 1.1 \\ 0 & 0 & 0 & 0.83 & 0.83 & 0.83 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.64 & 0.64 & 0.64 & 0.64 \\ 1.1 & 0 & 1.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.64 & 0 & 0 & 0.64 & 0 & 0 & 0.64 & 0 & 0 & 0.64 \\ 0.64 & 0 & 0 & 0.64 & 0 & 0 & 0.64 & 0 & 0 & 0.64 \\ 0 & 0 & 1.1 & 0 & 0 & 0 & 0 & 0 & 1.1 & 0 \\ 0.83 & 0.83 & 0.83 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.1 & 0 & 0 & 1.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.83 & 0.83 & 0.83 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.83 & 0.83 & 0.83 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.83 & 0.83 & 0.83 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.1 & 1.1 & 0 & 0 & 0 \end{pmatrix}$$

The singular value decomposition factorizes A into three matrices U, S and V. Their relation to A is stated in the formula 2.2.

$$U = \begin{pmatrix} -0.41 & -0.15 & -0.13 & -0.1 & -0.22 & -0.34 & 0.57 & 0.03 & 0.15 & 0.47 \\ -0.05 & 0.18 & 0.44 & -0.34 & 0.05 & -0.59 & -0.44 & 0.04 & -0.12 & 0.12 \\ -0.38 & -0.11 & 0.1 & 0.04 & 0.04 & 0.23 & -0.01 & -0.04 & 0.2 & -0.5 \\ -0.11 & 0.58 & -0.26 & 0.25 & 0.02 & -0.05 & -0.25 & 0.23 & 0.62 & 0.13 \\ -0.22 & 0.22 & 0.28 & 0.25 & -0.38 & 0 & 0.03 & -0.28 & -0.08 & -0.09 \\ -0.22 & 0.22 & 0.28 & 0.25 & -0.38 & 0 & 0.03 & -0.28 & -0.08 & -0.09 \\ -0.29 & 0.19 & -0.39 & -0.06 & 0.46 & -0.38 & 0.12 & -0.41 & -0.14 & -0.34 \\ -0.09 & 0.55 & -0.19 & -0.08 & -0.03 & 0.28 & 0.12 & 0.25 & -0.63 & 0.13 \\ -0.02 & 0.25 & 0.13 & -0.79 & -0.09 & 0.32 & 0.18 & -0.14 & 0.3 & -0.1 \\ -0.41 & -0.19 & -0.1 & -0.11 & -0.11 & 0.03 & -0.23 & 0.34 & -0.09 & -0.11 \\ -0.41 & -0.19 & -0.1 & -0.11 & -0.11 & 0.03 & -0.23 & 0.34 & -0.09 & -0.11 \\ -0.35 & -0.1 & 0.1 & 0.04 & 0.38 & 0.4 & -0.31 & -0.39 & -0.02 & 0.55 \\ -0.13 & 0.11 & 0.56 & 0.18 & 0.52 & 0.02 & 0.39 & 0.39 & 0.04 & -0.06 \end{pmatrix}$$

$$S = \begin{pmatrix} 3.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.75 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.38 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.24 \end{pmatrix}$$

$$V = \begin{pmatrix} -0.15 & 0.61 & -0.04 & 0.3 & -0.31 & 0.15 & -0.15 & 0.15 & 0.19 & 0.56 \\ -0.03 & 0.33 & -0.01 & -0.55 & -0.08 & 0.47 & 0.33 & 0.07 & -0.5 & 0.01 \\ -0.17 & 0.58 & -0.43 & 0.08 & 0.34 & -0.2 & -0.05 & 0 & 0.02 & -0.53 \\ -0.1 & 0.19 & 0.35 & 0.01 & -0.29 & -0.4 & -0.37 & -0.42 & -0.52 & -0.08 \\ -0.02 & 0.19 & 0.25 & -0.67 & -0.04 & -0.12 & -0.19 & -0.16 & 0.61 & -0.03 \\ -0.06 & 0.12 & 0.48 & -0.05 & 0.41 & -0.38 & 0.08 & 0.62 & -0.14 & 0.15 \\ -0.3 & 0.1 & 0.55 & 0.33 & 0.3 & 0.4 & 0.24 & -0.36 & 0.16 & -0.16 \\ -0.39 & -0.21 & -0.01 & -0.07 & 0.11 & 0.41 & -0.71 & 0.29 & -0.09 & -0.13 \\ -0.64 & -0.18 & -0.29 & -0.18 & 0.28 & -0.23 & 0.14 & -0.28 & -0.07 & 0.46 \\ -0.53 & -0.12 & 0.06 & 0.03 & -0.59 & -0.15 & 0.32 & 0.3 & 0.13 & -0.35 \end{pmatrix}$$

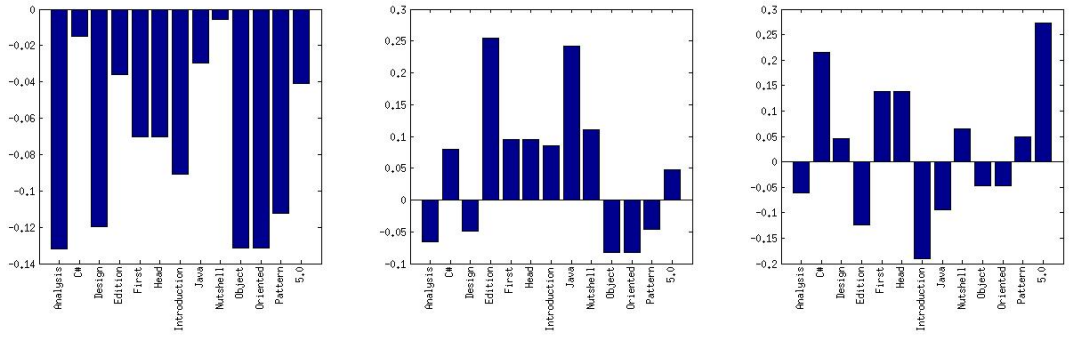
After computing the factors, the singular matrix S is reduced by k. This means, the k highest singular value are retained while the rest is removed. This gives us the $k \times k$ Matrix S_k

$$S_k = \begin{pmatrix} 3.1 & 0 & 0 \\ 0 & 2.3 & 0 \\ 0 & 0 & 2.1 \end{pmatrix}$$

Also, U is reduced by k. To achieve this, only the first k columns are retained.

$$U_k = \begin{pmatrix} -0.41 & -0.15 & -0.13 \\ -0.05 & 0.18 & 0.44 \\ -0.38 & -0.11 & 0.1 \\ -0.11 & 0.58 & -0.26 \\ -0.22 & 0.22 & 0.28 \\ -0.22 & 0.22 & 0.28 \\ -0.29 & 0.19 & -0.39 \\ -0.09 & 0.55 & -0.19 \\ -0.02 & 0.25 & 0.13 \\ -0.41 & -0.19 & -0.1 \\ -0.41 & -0.19 & -0.1 \\ -0.35 & -0.1 & 0.1 \\ -0.13 & 0.11 & 0.56 \end{pmatrix}$$

With the matrices U_k and S_k the model is completely built. The model is aware of k different concepts. Each one of the these is expressed as a composition of all words and their influence. The concepts are visible in the histograms below.



(a) **(b)** **(c)**

Figure 2.2: The three concepts calculated by the LSI algorithm

It can be seen that the different concepts correlate slightly with the topics of the books. In concept 1 the words Analysis, Design, Object, Oriented and Pattern have the most influence. Concept 2 distinguishes strongly the words Java and Edition. Although the word Edition has no connection to Java, the LSI algorithm has detected some similarities because two of the three Java books include both words in the title. Finally, concept 3 weights C# and 5.0 the most.

2.2.6.2 Building the index

With the two matrices S_k and U_k the initial corpus can now be transformed into the semantic space. The formula used for this process is 2.3.

The outcome is seen below. Each document is represented as a three dimensional vector. Every dimension stands for a concept.

	1	2	3
d_1	-0.15	0.61	-0.04
d_2	-0.03	0.33	-0.01
d_3	-0.17	0.58	-0.43
d_4	-0.1	0.19	0.35
d_5	-0.02	0.19	0.25
d_6	-0.06	0.12	0.48
d_7	-0.3	0.1	0.55
d_8	-0.39	-0.21	-0.01
d_9	-0.64	-0.18	-0.29
d_{10}	-0.53	-0.12	0.06

2.2.6.3 Querying the index

Given the index and the matrices S_k and U_k, it is now possible to query the documents. As input a Tf-Idf weighted query document q is crafted. The vector is then transformed into a semantic representation the same way as previously done while indexing.

By calculating the cosine similarity between the query vector q_k and the documents in the index, it is now possible to find out which of the document in the corpus is most similar.

Initiating, a possible query document.

*Thinking in **Java***

It can be seen, that the title covers the topic Java. Therefore, it should be similar to the books of this category.

$$q = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.83 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Using the equation 2.5 the Tf-Idf vector is transformed into LSI representation q_k.

$$q_k = \begin{pmatrix} -0.23 & 0.42 & -0.15 \end{pmatrix}$$

After transformation, the vector can be compared to the index documents. This is done by calculating the cosine similarity. The higher the similarity score, the more similar is the query to the document.

The similarity between the query and the documents in the index is shown in the following bar diagram. As expected, the title correlates strongly with the Java books.

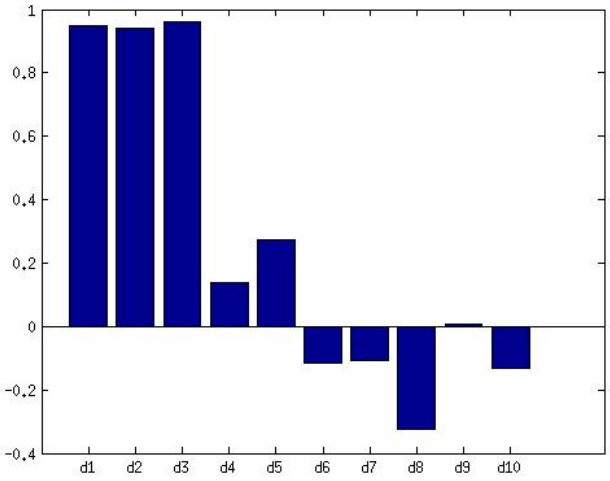


Figure 2.3: Document similarity

3 Recommendation System

3.1 Approaches

Recommendations are mainly generated by two different techniques - collaborative and content-based filtering.

3.1.1 Collaborative Filtering

Collaborative filtering is a technique for item recommendation by including information about user interactions like clicks, ratings and purchases of items. It is used by many online retailers - among Amazon.com is the most popular.

The main advantage of the collaborative filtering approach is that it is not needed to understand the items in detail. This is important in fields where analyzing items is impossible or tied to a lot of effort.

3.1.1.1 Association Rule Learning Method

Many of todays systems make use of the association rule learning method introduced by Rakesh Agrawal in 1994. It is an implementation of a collaborative filtering method. Its goal is to find item association by inspecting transactions previously done by users. Simply spoken, it proposes items in the manner of "user who bought/viewed item X also bought/viewed item Y".

In the context of recommendation systems, a rule stands for an association between two or more items. The association is simply expressed as a score. The higher the score the more likely the items are viewed or bought together.

For calculating the association between items, two values are important - support and confidence:

support: The support value stands for the proportional amount of transactions in which the items are bought together. As an example, out of 800 transactions in total 40 include the item set X, the support value $supp(X)$ is therefore $40/800 = 0.05$.

The support factor is mostly used as filtering criteria. Item sets that do not satisfy a specific support value are mostly not considered for creating rules.

confidence: The confidence value describes the conditional probability of buying Y when item set X is bought. It is calculated with the formula $\mathrm{conf}(X \Rightarrow Y) = \mathrm{supp}(X \cup Y)/\mathrm{supp}(X)$. The factor $\mathrm{supp}(X \cup Y)$ is the support value for transactions in which item sets X and Y were bought together.

In the contrary to the support value, the confidence is used as a sorting value. This means, the stronger the confidence the more associated is X to Y and therefore the more likely it is recommended if a user buys one of the items.

3.1.1.2 Apriori Algorithm

The number of possible item sets is the power set over all items. Due the power set grows in exponential manner, it is impossible to consider all possible item combinations. An approach for limitting the amount of sets is given by the Apriori algorithm.

The algorithm strongly relates on a minimal support value. The Apriori algorithm then incrementally increases the size of item sets and checks if they satisfy the given support value. If not, these sets are not considered for further examination. This way, the number of created sets is lowered.

Example The process is explained in the following example. For the reason of simplification, the support value is expressed as the absolut number of transactions.

A user has defined a minimal support value of 3. Since the start of the recommendation system, following transaction were recorded: *{beer, tomato, wine, spaghetti}, {beer, tomato}, {tomato, wine, spaghetti}, {tomato, wine}, {beer, tomato, spaghetti}, {wine, spaghetti}, and {tomato, spaghetti}*

First, the Apriori algorithm creates all 1-item sets and calculates their support value.

Item Set	Support
{beer}	3
{tomato}	6
{wine}	4
{spaghetti}	5

All the given items satisfy the minimum support value of at least 3. Therefore, for generating the 2-items sets all 1-item sets are considered.

Item Set	Support
{beer, tomato}	3
{beer, wine}	1
{beer, spaghetti}	2
{tomato, wine}	3
{tomato, spaghetti}	4
{wine, spaghetti}	3

Again, the algorithm checks if any of the sets has a support value higher than the specified minima. This is true for the sets *{beer, tomato}, {tomato, wine}, {tomato, spaghetti} and {wine, spaghetti}*.

Next, all possible 3-item sets are created. It can be seen that the only 3-item set is achieved by combining the sets *{tomato, wine}, {tomato, spaghetti} and {wine, spaghetti}*.

Item Set	Support
{tomato, wine, spaghetti}	2

The support value of the resulting set is lower than 3, hence it is discarded. As there are now no item sets available, the algorithm stops.

In conclusion, with the use of this method only 11 item sets had to be checked against the minimal support value. If initially all combinations were made, the amount of comparisons would have been $\frac{4!}{(4-1)!1!} + \frac{4!}{(4-2)!2!} + \frac{4!}{(4-3)!3!} \frac{4!}{(4-4)!4!} = 4 + 6 + 4 + 1 = 15$. This shows that the algorithm helps in decreasing the number of inspections needed.

The main challenge in successfully using the Apriori algorithm is finding a good setting for the support value. If it is set too high, too few rules will be found. On the other hand, a low value will end up in considering all possible item combinations.

3.1.2 Content-based Filtering

To the contrary of collaborative filtering, the content-based filtering method does not imply item association by analysing the user behaviour. Rather, it examines the characteristics of the users or the items. The first type is achieved by calculating the users interests based on their likes and dislikes. Afterwards, it is possible to suggest items similar users were interested in. For the second approach, item similarity measurements are utilized. If a user buys a specific item, the system can then suggest correlated items.

In this work, a content-based filtering method is implemented by the Latent Semantic Indexing algorithm.

3.2 Easyrec

Easyrec is a provider of a remote recommendation system. It is freely available for commercial and personal usage [7].

The main idea behind easyrec is to provide a simple ready-to-use recommender system. It is published as web archive and can be deployed on own server instances. For access, easyrec includes a REST and a JavaScript API.

3.2.1 System Overview

The easyrec is implemented as a client server application. The structure is as follows

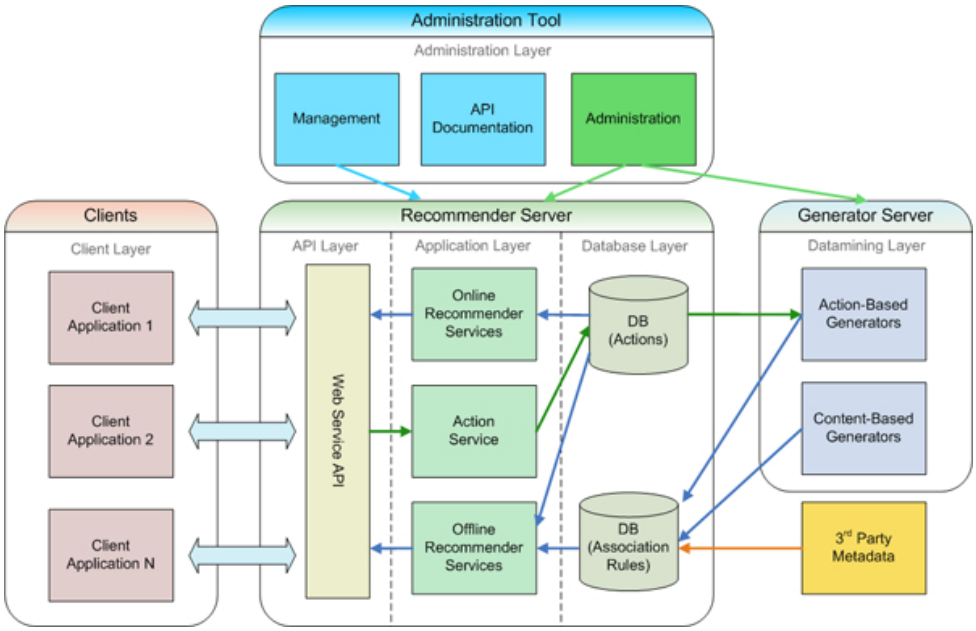


Figure 3.1: easyrec architecture
Source: wikipedia.org

Association Rules DB: The association rules DB holds all rules generated by the rule generators. Furthermore, it is also possible to import rules from external sources

Action Service: This service is being requested if a user has interacted with a specific item. Easyrec supports three different action types - view, buy and rate action

Recommender Service: Easyrec includes both an online and an offline recommender system. Compared to the online version, the offline recommender service generates recommendations calculated by the meaning of collaborative filtering methods. This includes suggestions of items other user also viewed/bought. It is called offline because the recommendations presented by the service are not up-to-date with the user actions. This is due to the long execution time of the Apriori algorithm

The online recommender system is updated with every action executed by a user. Recommendations from this service are mainly item rankings. To these belong overviews of the most bought or best rated items

Action-Based Rule Generator: The action based rule generator is the implementation of the collaborative filtering method. It is responsible for creating recommendations based on the three acrion types. As method, the Apriori algorithm is used

Item-Based Rule Generator: Easyrec also supports a method for content-based filtering. For this, it calculates the textual similarities between item profiles

3.2.2 Actions

The easyrec application differs between three different user actions - view, buy and rate. An action is always bound to a session identified by a session ID. If the ID is unique and lasts as long as a user stays on the website, transactions can be simulated. This is done by assigning actions with the same session ID and generated in a given time frame to the same transaction. For example, a user who buys two items during the stay on the website will generate a buy transaction including both items.

3.2.3 Recommendations

Easyrec provides a fair amount of recommendations. Among, the recommendations of most importance are the ones generated by analyzing the transactions made by users. They are listed below

- other users also viewed
- other users also bought
- items rated good by other users

All of the presented recommendations are products of the three different user interactions.

4 Concept

The basic concept is seen in the following diagram.

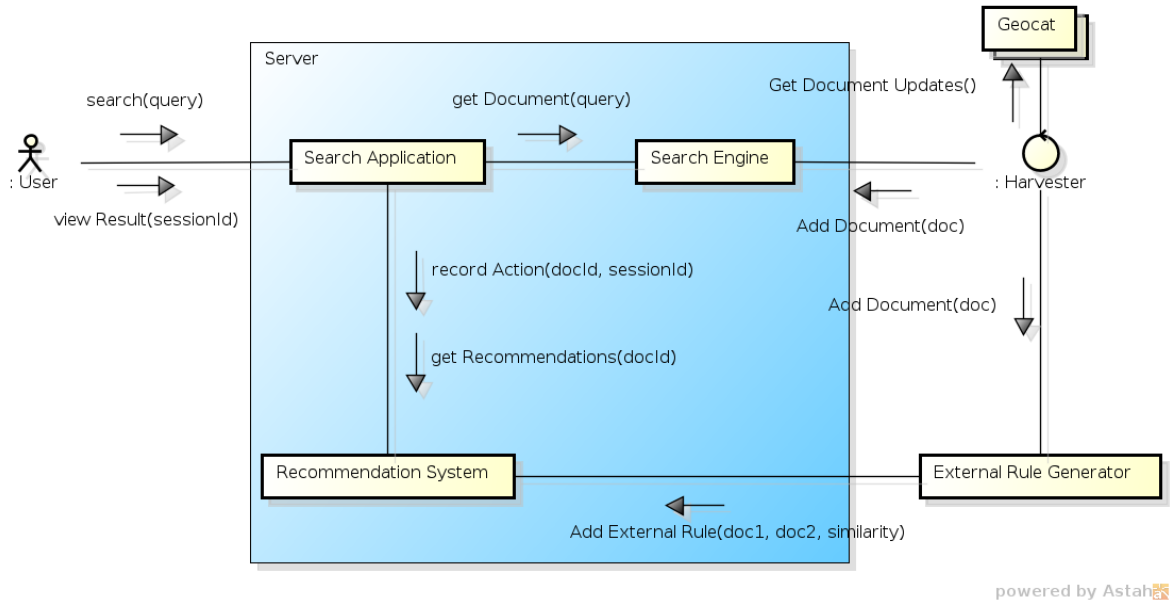


Figure 4.1: System Overview

In the sections below, the particular components are described more in detail

4.1 Recommendation System

The recommendation System is responsible for making document suggestions. It is designed to use both collaborative and content-based filtering. The first filtering method is calculated within the recommender system. It is fed from the search platform by recording user interactions. The second method is assigned to an external rule generator which computes content-based item relationships.

4.2 Search Portal

The search platform is the actual interface the user interacts with. It contains a search field in which the users can place their query string. After confirmation by the user, the query is sent as a request to the

search platform. The platform then retrieves the documents over its connection to the search engine. Finally, the search results are presented to the user in a scrollable window.

Over the connection to the recommendation system, recommendations for currently viewable search result are retrieved. Also, the search platform collects user actions like downloading or clicking on search results and sends it to the recommendation service. With these informations, recommendations can be adapted to the user interests.

Easyrec provides 3 possible user actions - view, buy and rate. Due to no rating functionality is planned, the third action type is not included. The remaining two are mapped to user interactions as showed in the follow image.

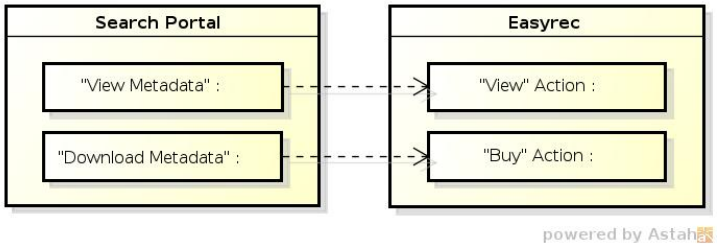


Figure 4.2: Relation between search portal and Easyrec actions

There are two different user interactions - view and download metadata.

View Metadata: The "View Metadata" action is executed if a user clicks on a search result or on a recommendation. The search portal then displays the detail view of the metadata

Download Metadata: The downloading functionality is only accessible through the detail view of metadata. The action is generated if the user clicks on the download button

4.3 Search Engine

The search engine handles query requests from users. It is bound to a document index where the geospatial metadata is stored. In this work, the search functionality is taken over by a Apache Solr server.

The query requests mainly refer to the textual content of the metadata. This means, users must craft query strings, containing keywords available in the metadata. In a further extent, localization or region based filtering methods can be applied.

4.3.1 Metadata Structure

Metainformation for geospecific data is specified under the ISO standard 19115. Furthermore, with ISO 19139 there is also a norm for metadata in XML-format.

The standard allows implementations of country or department specific profiles. The Swiss profile available is known under the name of GM03.

4.3.2 Text in Metadata

As the goal of this work is to extend text based search engines for geodata, it is the task to extract textual content out of metadata. In a first attempt the following text fields of the GM03 standard are considered. Later, the support of further fields are possible.

- *title*: Title of the cited resource
- *source owner*: Responsible party for the metadata
- *abstract*: A short description of the content data

4.3.3 Language

Proper language support is a very difficult task due the GM03 standard defines 4 different languages - *German*, *English*, *French* and *Italian*. The problem is that not all metadata are translated in every language. This means, documents for which no translation exists can only be found in its primary language.

Since no search system is given, proper support of multilingual documents is not intended to be dealt with. This is due to the tremendous effort which would exceed the scope of this work. As most of the data provided by Geocat is available in German, German is used as the default document language.

4.4 Harvester

The harvester is responsible for crawling the data sources for newly created documents. This is done by periodically sending requests to the geodata information systems. The sources can either be local or remote. The Documents are then added to both the search engine and the external rule generator.

The Harvester is not necessarily a component of the server application. Rather, it can be implemented as an external process communicating frequently with the search engine.

4.5 External Rule Generator

The main function of the rule generator is to calculate the item to item association for content-based filtering. This is done by computing document similarity with the help of the LSI algorithm. The similarity scores and the IDs of the documents are then imported as external rules into the recommendation system.

Similar to the harvester, the external rule generator is implemented as independent component. In a first attempt, the rule generator is written in python programming language, due to its useful topic modelling library - gensim.

4.5.1 Gensim

For this project, the equotations needed for building the document index and the similarity calculations are done by the python library called gensim [8]. It is a popular and easy-to-use toolkit created by Radim Řehůřek.

The library supports the required vector space models like Tf-Idf and LSI. In addition, there is also a handy application called simserver available. With its help, the effort for building and updating the document indices can be reduced.

4.5.2 Process

The diagram below shows the initial process for importing document associations into Easyrec.

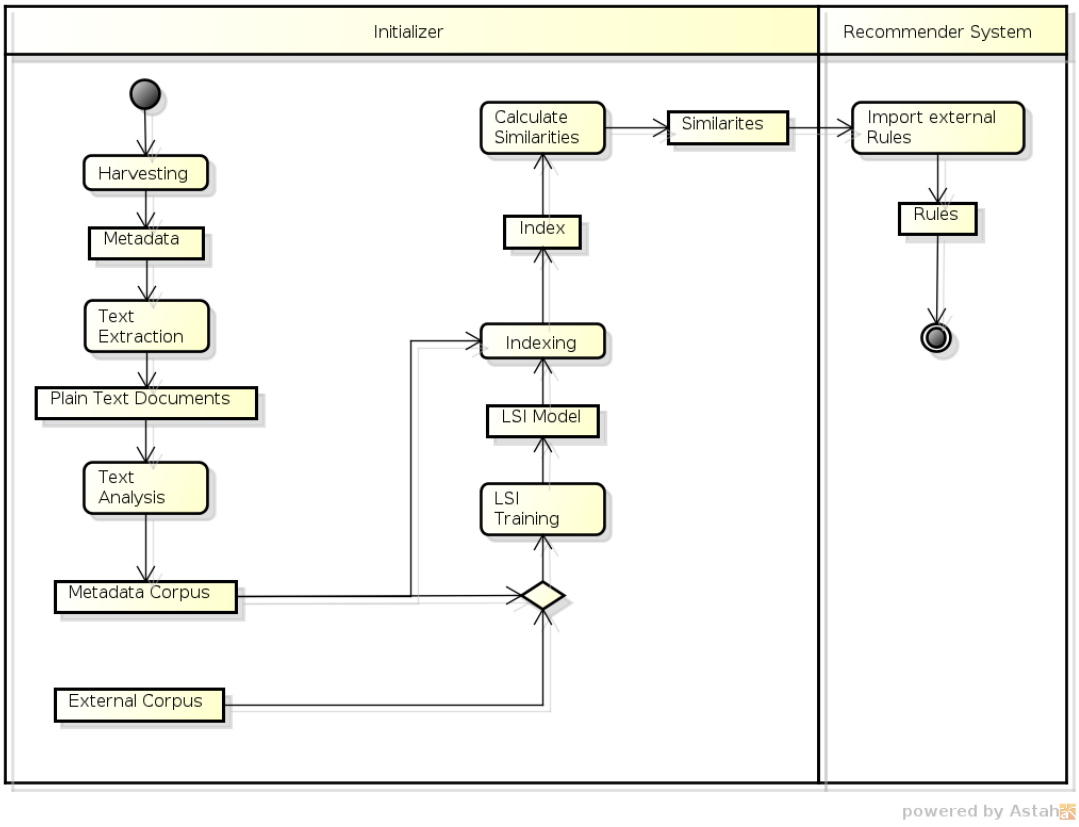


Figure 4.3: Process of importing document associations

4.5.2.1 Harvesting

A harvester is responsible for retrieving the metadata from external sources like databases, internet platforms or folders. The metadata is then available in different formats, for example PDF, XML or plain text.

4.5.2.2 Text Extraction

The LSI algorithm requires simple plain text data. In this step the text from gathered metadata is parsed and extracted. Due to the different formats of metadata available, a series of varying parsers must be implemented.

4.5.2.3 Text Analysis

Text analysis is the process of separate text into words. This includes lower casing, removing stop words and word stemming. This step is language dependent. Therefore, the language used for documenting the metadata must be known or recognized.

This is an important step for the LSI performance because it helps reducing the dimensionality of the problem. Otherwise, for example "Data" and "data" are considered as different words

The result is a corpus of preprocessed documents identified by their unique ids.

4.5.2.4 LSI Training

In the next step, the LSI model is evaluated. As input, a preprocessed corpus is used. Mainly, there are two different inputs possible.

External Corpus For generating the model, a corpus independent to the metadata can be used. Potential candidates are articles from Wikipedia or DBPedia. Due to the size of the corpus, a wide variety of words and topics are available. This allows learning many different word correlations. Synonym and polysemy handling can therefore be improved.

For maxing out the performance, the external corpus must be pre-processed the same way as the metadata-corpus.

One of the downfalls of this approach is the fact that the corpus of huge databases cover a lot of different subjects whereas geodata is specialized only in a few topics. Thus, the metadata is considered to be very alike. Additionally, the training of such immense data sources can take a long time.

Metadata Corpus Instead of using an external corpus for model generation, the already preprocessed metadata can be used. Problems arise, if the metadata has very sparse text information. This preconditions makes it difficult to recognise synonyms and polysems. On the other hand, the topics covered in the data is exactly the same as learned by the LSI algorithm.

4.5.2.5 Indexing

After model generation, documents can be added to an LSI index. For this, the pre-processed documents are placed into the LSI-based vector space model. Afterwards the coordinates are saved in an index. This process will be continued for every document in the corpus.

4.5.2.6 Calculating Similarities

Same as the other document indexes, the LSI index is used for querying documents. As input, a query is crafted and put into the vector space. With usage of the cosine similarity, for each document, a similarity score can be computed. The score ranges from 0.0 describing no correlation to 1.0 describing an exact match.

Since the similarity among the documents is requested, each document is now once used as a query. This means, for an index with n documents $fracn(n-1)2$ similarities will be calculated.

4.5.2.7 Import External Rules

Finally, the similarity scores are imported as rules into the Easyrec application. The rules are then stored in the Association Rule DB.

5 Conclusion

5.1 Results

The work has shown that the inclusion of recommender system can improve the discovery of geospatial metadata in geographic information systems. The system successfully combines content-based and collaborative filtering methods for document recommendations. The recommendations are displayed once a user examines a search result. This can help finding documents that were not previously visible.

Recommendations are a result of two different sources. First one is an implementation of a content-based filtering method. It is based on similarity calculation with the usage of latent semantic indexing. Besides enhancing the precision of recommendations, it has been shown that the method is also very useful for overcoming the long initial phase. The second source is a collaborative filtering method. This technique is based on user interactions. Once enough data is acquired, it is considered to be very valuable.

5.2 Improvements

5.2.1 Handle Duplicates

Text is not the main content in metadata. For this reason, the descriptions are usually very short and it occurs that the same title or even the same description is used for different documents. This is due to the fact that similar geospatial data of the same region. Therefore, the schema is simply copied and reused.

For duplicates, the external rule generator computes a high similarity value. This will lead to strong relationships between those documents. Although this behaviour is correct, it is often not very useful. This is due to the fact that recommendations should help finding documents that were not explicitly queried. Furthermore, diversity among recommendations will boost the searching experience. The recommendation system should therefore suggest documents not covered by the initial search query of the user.

To overcome this issue, documents with cosine similarity higher as a specified threshold could be cut off. With this extension, one could prevent suggesting copies of the same document.

The problem can also occur, if a document is unique, but it relates best to a set of near duplicates. Also in this case, the algorithm only suggests documents with the same text template. A possible solution could be provided by considering the similarity only to one of the copies. All relations to the other documents are not imported as rules. The question occurs, which relation should be retained. Also, how does the user get access to the other copies.

5.2.2 Include other Metadata fields

In this work, only the title and description field of the metadata are used for computing the similarity values. As it turns out, the text content is mostly very sparse and in many cases highly generic. These circumstances make it difficult to distinguish between documents. Thus, the similarity is not very accurate. Therefore, it is required to include more metadata fields. As an example, the distance and the form of the geographic objects could be included. Other characteristics like information about the source owner and the region could also be considered.

5.2.3 Support Multilanguage

Multilanguage comes into play if metadata is written in different languages. This is most likely the case in Switzerland. Although, the metadata format GM03 supports different languages with the usage of locales, not all contributors make the effort to translate the titles and descriptions into different languages. Treating all data the same makes it impossible for the LSI algorithm to identify similarities. This is due to the fact that the algorithm would recognize the different languages as different concepts. Rather, one would want the algorithm to calculate similarities independed of the document language. For example, a document about forests in German should correlate to one written in French.

As a possible solution, machine Translation could be included. Although the translation is in most cases not very precise, accuracy is not necessarily requested. The reason for this is the fact that translated texts are only used for creating the LSI model.

6 Nomenclature

Expression	Description
Collaborative Filtering	Collaborative filtering is a technique used in recommendation systems for calculating item similarities. The calculations are based on user interactions like buying and viewing items. Two items often viewed or bought together are considered to be very alike or supplementary.
Content-based filtering	Content-based filtering is a technique used in recommendation systems for calculating item similarities. The calculations are based on the characteristics of particular items. Items with high correlation in their profile are assumed to be similar to each other.
Geodata	Geodata is also called geospatial data. It holds information about geographic objects like locations and areas. The information is usually stored as coordinates or planes on earth's surface.
Geometadata	Geometadata is also called geospatial metadata. It is a type of metadata describing geospatial data. It usually contains textual information about the data and the collaborator
Latent Semantic Indexing	Latent semantic indexing is a adaption of the vector space model. It tries to reduce the number dimensions by merging highly correlated words into a topic.
LSI	Acronym for latent semantic indexing.
Recommendation System	A recommendation systems is used in retailer system for suggesting similar items to users.
Rule	A rule stands for an association between two or more items. The association is simply expressed as a score.

Table 6.1: Glossar

List of Figures

List of Tables

Bibliography

[1] Geoinformation Federal Office of Topography Coordination and Services. geocat - geographic catalogue. http://www.geocat.ch/geonetwork/srv/eng/geocat, 10 2013.

[2] The Apache Software Foundation. Lucene. http://lucene.apache.org/core/, 10 2013.

[3] The Apache Software Foundation. Apache solr. http://lucene.apache.org/solr/, 10 2013.

[4] Mittlböck M. Vockner B. Verwendung von textanalysekomponenten zur verbesserung der geo-discovery mit recommender-systemen. Thesis, Reseach Studios Austria, 2013.

[5] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[6] R. Bradford. An empirical study of required dimensionality for large-scale latent semantic indexing applications. http://en.wikipedia.org/wiki/Latent_semantic_indexing, 2008.

[7] Research Studios Austria Forschungsgesellschaft mbH. Easyrec. http://easyrec.org/, 10 2013.

[8] Radim Řehůřek. gensim - topic modelling for humans. http://radimrehurek.com/gensim/, 10 2013.

[9] Schütze H. Manning C., Raghavan P. *Introduction to Information Retrieval*, volume 1. Cambridge University Press, 2008.

[10] Media Wiki. Latent semantic indexing. http://en.wikipedia.org/wiki/Latent_semantic_indexing, 9 2013.

[11] Media Wiki. Singular value decomposition. http://en.wikipedia.org/wiki/Singular_Value_Decomposition, 10 2013.

[12] Puffinware LLC. Latent semantic analysis (lsa) tutorial. http://www.puffinwarellc.com/index.php/news-and-articles/articles/33-latent-semantic-analysis-tutorial.html, 1 2010.

[13] Media Wiki. Recommender system. http://en.wikipedia.org/wiki/Recommender_system, 10 2013.

[14] The Apache Software Foundation. Apache mahout. http://mahout.apache.org/, 10 2013.

[15] Donald J. Patterson. Using matlab for lsa. http://www.ics.uci.edu/~lopes/teaching/inf141W10/slides/Lecture19.pdf, 1 2010.