Leopold-Franzens-University Innsbruck

Institute of Computer Science
Databases and Information Systems

De-duplication and error correction of OCR-processed index cards using optimized string metrics

Master Thesis

Markus Ruepp

supervised by
Prof. Dr. Günther Specht and Robert Binna

Innsbruck, October 12, 2011

Abstract

This thesis presents an approach of error correcting torn texts in an automatic way. The drafted system beyond uses record linkage, where multilingual error prone *optical character recognized* (OCR) full-texts are linked with similar, error corrected records in a web catalog. In general, web catalog are accessed using queries. Hence, we use *Query Formulation* techniques to generate keyword based queries from the OCR texts. The keywords may be error corrected using fuzzy search engines, before executed against a catalog retrieval system. Several well known information retrieval mechanisms, like *thesaurus expansion* and *query reformulation* are used to avoid bad retrieval performance and to react in a proper way.
After retrieving a set of records from a catalog, a string metric sorts those records according to their particular similarity with the full-text. The similarity measure uses an adaption of the n-gram overlap to extract relevant features of the full-texts. Error robust character based Levenshtein calculates the string similarity on feature level. The resulting feature-based similarities are weighted and combined.
The similarity measures are classification into similar, dissimilar and undecidebale, indicating, whether the most similar catalog record should or should not be linked with the full-text.

Contents

Chapter 1

Introduction

In large database systems with mutliple actors, record-redundancies are common. Those redundancies are mostly unwanted and lead to extra costs. De-duplication and its theory beyond, record linking [FS69], identifies those redundant/similar records and triggers the removal of duplicates. Current record linking systems perform record comparisons on grounds of their particular feature-vectors, often including a preliminary statistical analysis. In classical record linking, all records are available and accessable in a direct way. In more complex retrieval systems (RS), records are accessed using discrete queries, that specify particular properties of a target record in fragmentary repository.
The process of record deduplication in a system that requires queries, can be drafted as follows:

- Retrieve a record r
- Query for potentially similar records r_s
- Measure for each retrieved record in r_s the probability of equality with r
- Apply deduplication actions on similar records

Measuring equality of two records is tricky, especially if the record structure is complex. In many cases equality expresses semantics not syntax, which is notably dure to measure concerning texts, because there are several valid syntax for one meaning.
De-duplication of textual records use (string-)similarity metrics to identify redundant candidates for deletion. General speaking, with increasing heterogenity of texts, string similarity measures loose confidence.
Measures on texts resulting from an OCR-process, that show high error level rates, are markedly difficult and therefore rarely applied without

preliminary error correction. Marciniszyn et al. [MA04] tried to reconstruct texts from shreddered pieces from the Stasi[1] regime, former GDR[2]. The so-called E-jigsaw algorithm exploits shape analysis of the snippets and creates neighbor hyphothesis for other snippets. There has been plenty of research under this domain with various publications [KDS10, SBP05, NS06, Tyb04, NDH08].

Given a set of two million index cards, we perform advanced record linking of error prone OCR text with records from a web catalog. OCR error correction is tricky so current error resolution algorithms are rarely able to correct, according to [TE96] more than 60% of the errors. To achieve a theoretical hundred percent correctness, we perform advanced record linking with a web-catalog. The web-catalog we used, contains more than 90% of the index cards (p. 6), which are error corrected and semantically annotated (p. 8).

The paper focusses on the design of a prototype system that tries to retrieve a set of potentially similar records from a web catalog for an index card, using queries. In a subsequent de-duplication step, each record in the search results is scored based on the particular similarity with the index card. The card text are results of an OCR process.

The suggested record-linking algorithm is made up from two modules. First, the query formulation part (chapter 2), that acquires a potential set of similar records from the web-catalog, second, the similarity measures that score the retrieved records according to their similarity with the index card (chapter 3).

1.1 Motivation

Shortly after the year 2000, strong digitizing efforts at the University of Innsbruck lead to the processing of millions of index cards (refer to p. 6). The index card technology had become outdated and replaced by more powerful digital approaches like databases and search engines. Such a card carries relevant, human readable information to identify a document in the archive and references its analog place. This textual information, a records' abstract, may be reused in a digital approach, in order to save funds. The default way to digitize texts is the widely used technology *optical character recognition* (OCR) [MNY99], that translates scanned images to an electronic, machine-encoded text.

Unfortunately it is not that easy. The results of an OCR process become worse with decreasing quality of the input data. In the case of University of Innsbruck, this is a significant problem. The build-in error corrective

[1] http://en.wikipedia.org/wiki/Stasi
[2] http://en.wikipedia.org/wiki/East_Germany

mechanisms (dictionary-based term comparisons), the OCR engine[3] is shipped with, which should minimize the impact of bad input quality, faint. Two card features that influence the error correction in a bad manner are: the correction process expects a monolongual text, though many cards cover multilingual texts (1) and the dictionaries beyond are relatively weak, because highly technical terms and abbreviations are hardly covered (2). In addition, the degree of destruction can be too high, that a correction/resolution of a torn word to its propper spelling may be untrustworthy.

1.2 Statement of the problem

Coming with the decision of moving from an analog index system (index cards) to a digital system, old evolved index cards had to be digitized using Optical Character Recognition [**?**]. Due to the heterogenic character of the cards, featuring several fonts, the OCR process produced error prone OCR texts. After all, two million card text have been produced. The decision maker at the University wanted an error free set of texts, to simplify post processing, though manual revision has not been arguable due to shortness of money. Simple error correcting systems failed, because of highly technical terms, abbreviations and a widespread mixture of different languages, even on one card at the same time.
Hence, they headed toward a system, that is able to error correct and annotate (refer to 8) most texts in a satisfyable with within reasonable time.
An approach that meets those requirements, uses record linking (refer to p. 9). Record linkage (RL) basically associates two or more objects on basis on a fullfilled measurable criteria. If the criteria is similarity, there are numerous synonyms for this problem [4]. Objects of the analog catalog, represented through errornous full-texts are linked with similar records from web catalogs, which hold mostly sighted and annotated texts. A successfull linkage premises that a similar record exists. If so, an index card could be resolved as a correct and granular annotated record.
However, the problem setting prevents the classic application of record linking. In simple cases record linkage is performed on records with a homogenic and comparable structure. More complex approaches try to link records that are unstructered and errornous. In this thesis we link errorprone short full-texts with annotated, verbose records from a different source.
Both data structures have to be converted to a comparable corpus. At

[3]http://www.abbyy.com/

[4]merge-purge, de-duplication or entity resolution [HYKL08]

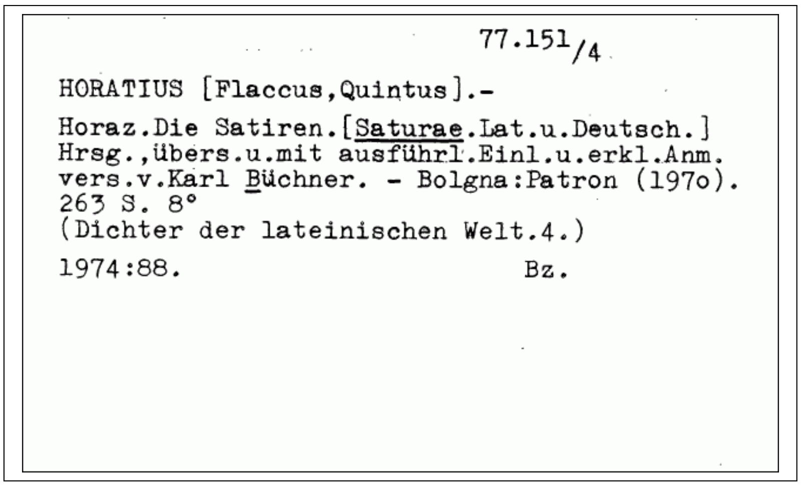
a closer look, there are some additional challenges that come with this record linking method. It is hardly possible to perform statistical analysis of the terms in the records, like TF-IDF [SB88], because web catalogs are usually accessed via queries.

1.3 Terminology

Throughout this thesis, some terms are of utter importance and require a definition.

1.3.1 Index card

An Index Card (see figure 1.1) is usually made of heavy paper of standardized size, that has been a common technology to administer large amounts of data, like books in libraries. They carry predefined pieces of information, representing the *key* in a key-value based index. To simplify card handling, an index card may follow a well defined structure, featuring predefined fields, and are hold in an alphabetical order (e.g. title or the authors surname). Cards usally carry an internal index number, which marks the place of the indexed object, the value.
Digging deeper, the index cards at the University Innsbruck, usually carry the following pieces of meta information: title, short description, the authors, publisher, year, type, place and reference value.

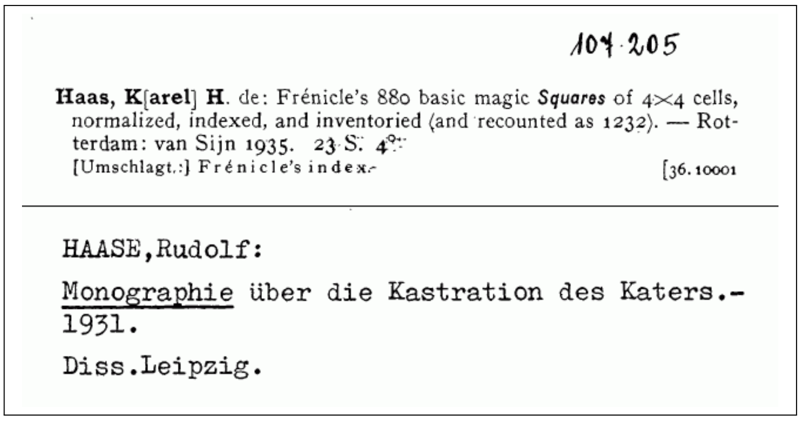


77.151/4

HORATIUS [Flaccus,Quintus].-
Horaz.Die Satiren.[Saturae.Lat.u.Deutsch.]
Hrsg.,übers.u.mit ausführl.Einl.u.erkl.Anm.
vers.v.Karl Büchner. - Bolgna:Patron (197o).
263 S. 8°
(Dichter der lateinischen Welt.4.)
1974:88. Bz.

Figure 1.1: A typewritten index card from the catalog at the University Innsbruck.

An index system, no matter if digital or analog, may be a constantly evolving object, that will be extended if e.g. a book is added to the library. When the University Innsbruck initiated the catalog, index cards had been typewritten. The ongoing technical progress made typewriters oldfashioned and induced their replacement by printers. In figure 1.2 shows a degenerated text, originated over time. The significant impact on the readability can cause troubles in the OCR process and lead to word errors.

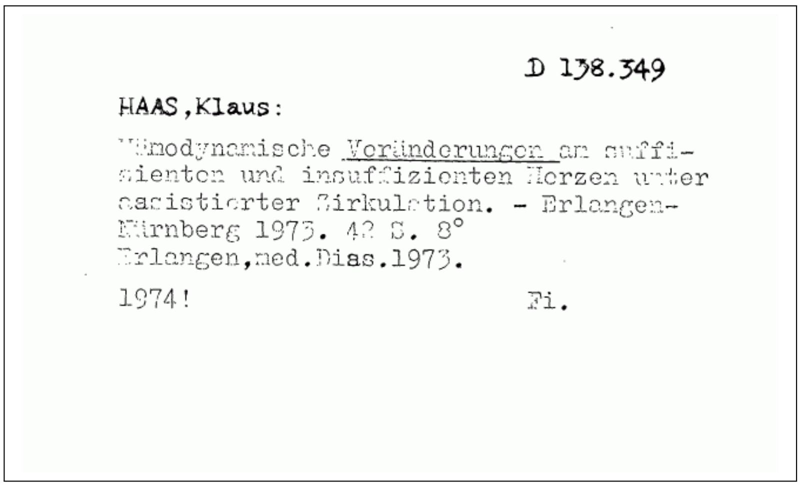

107.205

Haas, K[arel] H. de: Frénicle's 880 basic magic *Squares* of 4×4 cells, normalized, indexed, and inventoried (and recounted as 1232). — Rotterdam: van Sijn 1935. 23 S. 4°.
[Umschlagt.:] Frénicle's index. [36.10001

HAASE,Rudolf:
Monographie über die Kastration des Katers.-
1931.
Diss.Leipzig.

Figure 1.2: Index cards and the evolution of typing. The card on top is printed with several different fonts. The index card on bottom is solely typewritten.

Time, technology and storage can lead to a degeneration of readability, so even humans may have problems with reading those cards (figure 1.3).


D 138.349

HAAS,Klaus:
Hämodynamische Veränderungen an suffi-
zienten und insuffizienten Herzen unter
assistierter Zirkulation. - Erlangen-
Nürnberg 1973. 42 S. 8°
Erlangen,med.Diss.1973.
1974! Fi.

Figure 1.3: Degeneration of quality and readability.

1.3.2 OCR text

An OCR text is the result of a text extraction of a image. It is generated from scanned images, where certain areas in the image are classified by exploiting graphical characteristics as a specific character. This classification is based on a threshold and therefore strongly dependent on the image quality. Most OCR-texts are reclassifications, based on dictionary knowledge to improve the text quality. The final OCR text should be human readable. Still, despite the usage of dictionaries OCR texts may contain, faulty words like *Mile* instead of *Mike*. Both are syntactically correct, in the particualar context only one may be valid.

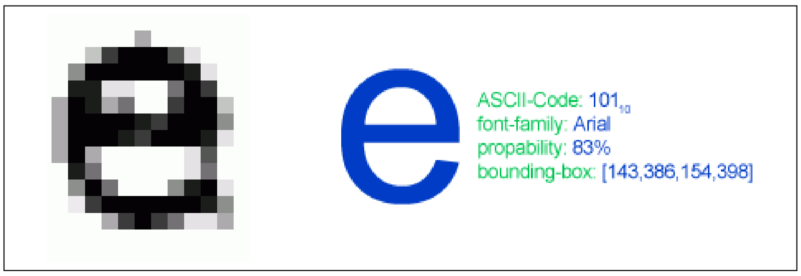
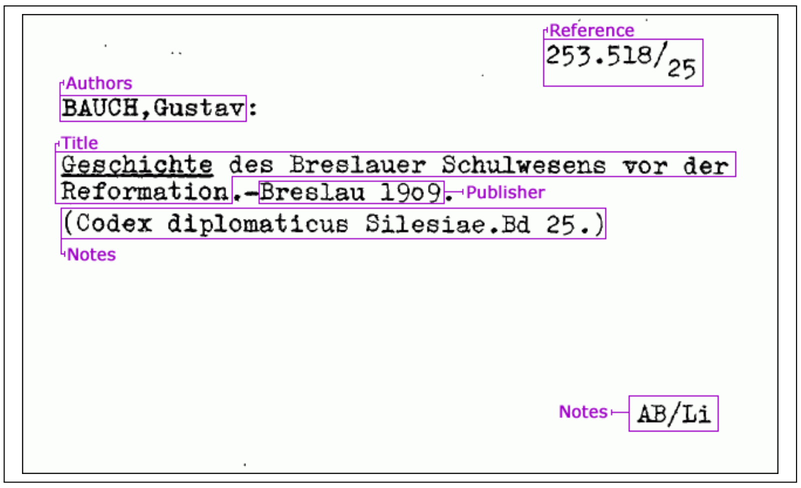


Figure 1.4: Illustration of the OCR process.

1.3.3 Semantic Annotation

In this thesis we will frequently mention the keyword *annotation* and mean *semantical annotations*. Semantic annotation is the association of semantical knowledge with an arbitrary object. Often, words are use to express the semantics. Figure 1.5 shows an example of a annotated index card. This process requires appropriate knowledge, for example to identify the name of an author in a text.


253.518/25

BAUCH,Gustav:

Geschichte des Breslauer Schulwesens vor der Reformation.-Breslau 19o9.

(Codex diplomaticus Silesiae.Bd 25.)

AB/Li

Reference
Authors
Title
Publisher
Notes
Notes

Figure 1.5: An index card with semantical annotation.

1.3.4 Web Catalog

A *web catalog* is a digital, accessable and searchable unification of collections of libraries. In this thesis we use the worlds' largest catalog worldcat[5]. A member of the Online Computer Library Center (OCLC) is entitled to access the entire data. Worldcat stores the vast majority of the index cards. Those cards are spell checked and annotated. However, any web catalog, which contains the required card, can be used.

1.3.5 Record Linking

Record linkage (RL) is the crucial part in this thesis. In general, RL is a method where two or more records are linked on the basis of the existance of a particular characteristic c. In the case of our problem setting, we examine the bidirectional textual similarity property of two texts.
In mathematical terms, RL is a technique of resolving a bipartitie graph [Wes00]. A bipartite graph $G = (V, E)$ is a model for the relationship of its nodes V and edges E, such that for any edge $v, w \in E | c \in A, w \in B$ holds. V can be splitted up into the two groups A and B, as shown in figure 1.6.

[5]www.worldcat.org

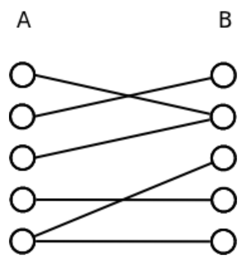


Figure 1.6: Record Linkage is similar to the mathematical problem of resolving a bipartite graph G.

Chapter 2

Query Formulation

In information retrieval, a query is used to acquire a set of records from an information system. A query is a precise, computer readable request to such a system, that satisfies a predefined syntax, namely the query language. A *query language* is a formal description, describing the properties and structure of a query. The quality of a query is measured by precision and recall (refer to p. 42). The set of valid query languages is defined by the retrieval system, in our case the web catalog worldcat.

Regarding the problem to retrieve potentially similar records from a retrieval system using queries for an index card, we use query formulation. *Query formulation* is the process of translating a text into a query, by extracting qualified keywords. The decision whether a word is qualified to be used in a query is based on a decision model. The decision model is generated in a preliminary step using a training set and approximates the retrieval potential for every word using a set of features.

Query formulation may encounter difficulties if the text is errornous. Since only valid written words should be used in a query, error prone texts should be corrected. Given an OCR-text (p. 8), error correction and word validation is absolutely necessary. In such a case the decision model may also evaluate the spelling of the word.

In this thesis, we use the web catalog worldcat (retrieval system), which supports the query language CQL, with a keyword-based query syntax. The retrieval system will return a set of records that satisfy the query. Since the language follows a logic-like syntax, it is possible to generate *unsatisfiable* queries. In section 2.2.4 we will discuss methods to deal with query unsatisfiability.

A successful query will return at least one record, that is similar to the index card. In case a query returns no results (failed query), we apply some query expansion and reformulation techniques to improve the retrieval performance (query quality).

Figure 2.1 illustrates a basic problem in a query based retrieval process.

Since a query is always an abstraction of an object, it has to be as accurate as necessary to guarantee a manageable amount of hits and as imprecise as possible at the same time, to allow small differences between similar objects. Otherwise valid, favored objects are excluded or the set of result is too large with just a few valid objects.

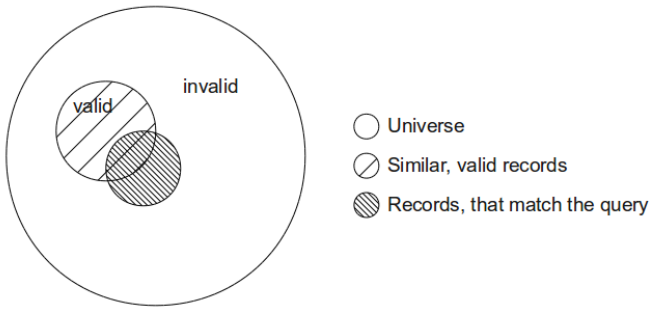


Figure 2.1: Query based retrieval: The performance can be measured using precision and recall (refer to definitions 2.1 and 2.2).

For sake of completeness we want to note, that there is no guarantee, that there is a similar record in the catalog for every index card.

2.1 Related work

We can distinguished several branches of related work: word error correction, term-based query formulation and query expansion.

Word error correction strategies can be be classified into *non-word error detection*, *isolated-word error correction* and *context-depended word correction* [Kuk92]. Non-word error detection is the application of term validity (correct spelling) problem. Basic spell checking engines are based on dictionaries. Hence, the existence of a term in a dictionary proofs its validity.
In order to correct a misspelling, GNU Aspell[1] uses a combination of Metaphone [HA03] and Levenshtein distance [Lev66] to match a misspelled word to a set of words from its dictionary. An alternative approach is the use of n-gram [CT94]. The search-framework Apache Lucene[2] provides a fuzzy-search based on n-grams. Lucene allows to create more complex dictionary records than Aspell. Such a lucene record, called document, can be extended with statistical information. [MI03]

[1]http://aspell.net/
[2]http://lucene.apache.org

et al. suggest the Porter stemming algorithm in a preprocessing step to enlarge the hit-set (recall).
Complex isolated-word error correction systems exploit statistical analysis like *statistical language modeling* (SLM) [TE96]. [TZMFE96] uses spelling error patterns and letter n-gram. The resulting system, extended by character confusion probabilities and a maximum likelihood sequence estimation (viterbi algorithm [Vit03]) enhances the error-reduction rates up to 60%.
A totally different approach to isolated-word error correction in detail, or fuzzy string search problem in general, proposes the prefix-tree structure trie [Fre60] to store strings. Within a trie, every word has a well defined position, whereas similar words are stored close together. Words with same prefixes are placed in the same subtree. The search function returns the value of the closest node, that is stored in the trie. Recent Trie-algorithms are fast and efficient for string search [AS07, HZW02], even space-optimized tries exist (PATRICIA[3] [Mor68]). However, the error resolution capabilities of a trie are limited, because the weights for a misspelled character decreases by the number of possible children of a node per position.
Following, we want turn towards term-based query formulation. Assuming a prior proper error resolution of a text, in order to generate a term-based query, most relevant terms have to be identified. This common information retrieval problem is known as keyword extraction [Tse98, MI03] or term relevance [HR01, Spi94], among others[4]. These term relevance approaches can be classified into two groups: approaches that require global statistical knowledge from a preprocessing step (A) and rest (B). Tf-idf[5] [Spa88], part of group A, defines a term relevance by its inverse global term frequency in the domain-depended corpus. This corpus has to be created in a preprocessing analysis.
There are cases, in which a frequency-corpus cannot be build due to a lack of information. [MI03] solves that drawback. They suggest a χ^2 measure, that shows comparable performance as tf-idf without the need for a corpus (Group B). In both cases term relevance is based merely on the term frequency.
If a term-system provides more extractable features than frequency, its best practice to build a term weighting system by equipping every term t_i with an appropriate weight w_i. Finding the optimal weights w_i (optimization problem) can be solved using search algorithms like genetic programming (GP) [KP05]. GP will be explained and discussed in chapter *Similarity metrics* (chapter 3).

[3]http://www.csse.monash.edu.au/ lloyd/tildeAlgDS/Tree/PATRICIA/

[4]similar notations for *keyword extraction*: term importance [MI03], document surrogate [Jo03]

[5]term frequencyinverse document frequency

Fixing spelling errors and weighting terms, are crucial techniques of query expansion [Eft96] (QE). Another QE-technique is to extend a term with morphological similar words (synonyms). This approach reduces the word mismatch problem [XC96]. Morphologically similar words are found in a lexical thesaurus, which can be, according to [CY92], automatically generated.

2.2 Method

The designed query formulation/execution approach necessitates the following two data structures:

- *database*, containing full-texts
- *web-catalog* that provides corrected and annotated records, accessed via term-based queries.

The algorithm starts with the retrieval of card-texts from the database. There are no requirements to the full-text. Based on the set of delimiters, the text is splitted into its tokens (term), spell checked and, if required and possible, error corrected. Each term of the resulting term-set is ranked (term score), according to its particular estimated retrieval performance in a query. The retrieval performance represents the quality of a query, which will be discussed later. Terms that promise a high retrieval performance achieve a high score. The query is executed against the web-catalog and interactively improved until a termination condition is fulfilled.
The crucial part in the process is to measure and define the query quality. The query quality is geared to the following statements

- avoid or dampen the impact of logical exclusions in query (Section 2.2.4)
- be as restrictive as possible to retrieve a manageable number of hits.
- retrieve at least one matching record.

In the following sections we will describe methods to improve the query quality.

2.2.1 Training Set and Quality Assurance

In artificial intelligence, algorithms can be evaluated using training sets, to state, whether they meet their requirements or not. Hence, the considered algorithm is executed in a well defined environment with defined output. In information retrieval there are two quality measures for a retrieval process (query)

$$precision := \frac{\{relevant\ records\} \cap \{retrieved\ records\}}{\{retrieved\ records\}} \tag{2.1}$$

and

$$recall := \frac{\{relevant\ records\} \cap \{retrieved\ records\}}{\{relevant\ records\}} \tag{2.2}$$

which can be seen as redrafts of the quality gears defined above.
We use a training set *ts*, to evaluate the query by calculating *precision* and *recall*. Training set *ts* consists of 200 tuples of full texts from index cards and their particular matching set of records in the web catalog. The index cards have been randomly chosen from a total amount of approximately two billion.

$$entry_i := (fulltext,\ query,\ records)$$

where

$$records := \begin{pmatrix} record_1 \\ \ldots \\ record_n \end{pmatrix}$$

These sets of relevant records have been generated manually. For learning purposes every training set entry is extended with a manually formulated query for worldcat, that will will retrieve at least one similar record of the record set aside some dissimilar records.

2.2.2 Text analysis

Unless there are no agreements of structure or composition of the full texts, we try to analyze it. Before splitting a text into its logical components, which are called terms, we perform a comprehensive analysis of the card-texts to detect problem-specific text features. The analysis consists of two individual steps that analyze

- inter-term and
- intra-term features.

The inter-term analysis tries to infer rules from the hidden, concealed structure, like the order of the field-values. We discovered that on most index cards are structured in the order author, title, description. Since we don't know the limits of those fields, a card text is just a bunch of words, though well structured.
In the second step we focus on isolated terms and their anomalies. It shows frequent displacement of blanks (blanks-displacement problem), which is a result of the OCR process. This frequently results in two well known word errors,

- Separation of one term into two substrings (i.e. separated-term error), e.g. *not ification* instead of *notification*. In most cases the resulting word fragments are nonsense.

- Run-together problem[6], where two or more words are treated as one, because no blanks are inserted, e.g. $badnotification$.

The correction of those errors is rather complex. Separated-term errors can be resolved by validating concatenated proximate terms against a dictionary. This doubles the worst-case time complexity to validate a set of n terms from $\mathcal{O}(n)$ to $\mathcal{O}(n * (n-1)) = \mathcal{O}(n^2)$.

The fusioned-terms problem can be solved in two ways. The previously introduced ordered prefix tree trie [Fre60] is able to derive the distinct terms in an iterative manner. A trie extends the behavior of a dictionary, by grouping syntactically related strings.

A mitigated method to correct the fusioned-term problem is rule based. The according rule-set assumes a switch of case (`HelloPeter`) or type (`1860München`) to derive a missing separator.

Statistics

A manual full-text analysis of the training set (2.2.1) revealed that 3.5% of all parsed terms are *fusioned-terms*. The primitive rule-based approach is able to resolve 85% of fusioned terms into their particular components.

The reason for the satisfiing performance of the rule-based approach is that for most fusioned terms at least one part is very short, e.g. `8München`. Moreover many index cards are written in German, where nouns start with an upper case. The histogram in figure 2.2 shows the distribution of term length focusing on the shorter term in a term fusion. It states for the majority of fusioned terms, at least one term is shorter in length of 4. This indicates that other resolution strategies, like fuzzy search, may be successful, if there is a dominant part in the fusioned term. Hence a term short invalid pre/suffixes will be resolved.

[6]notation according to GNU Aspell http://aspell.net/man-html/Controlling-the-Behavior-of-Run_002dtogether-Words.html

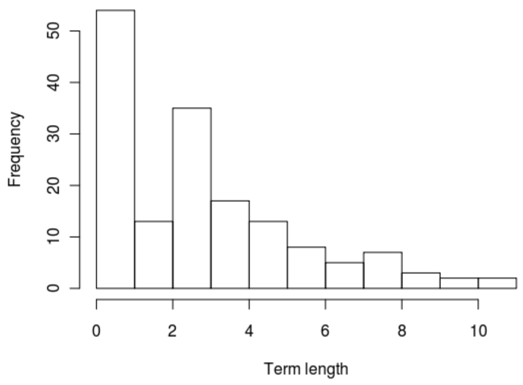


Figure 2.2: Histogram of term length of the shortest term in a term-fusion.

Results and Alternatives

Still, 15% of all fusioned terms have been unresolved, which seems quite unsatisfying.
The alternative of using a the previously introduced prefix tree (trie), is able to resolves up to 97% of this particular training set. To match words, a trie requires a relatively exact spelling, which may no be given. Moreover it has to cover a huge amount of words in different languages, with about the same capacity of the dictionary we use later. Such a huge trie is inefficient and not essential for the post-processing.
To a certain degree, fuzzy search can be used to resolve the *Separated-term problem*, which will presented in the next section. The fuzzy approach allows us to recover the dominant part in a fusioned term.

2.2.3 Spell checking and correction

In the tense *term analysis* from above we suggested methods to resolve some problem related errors in the full texts. Since the designed prototype has to handle error prone OCR texts from we will focus on general spell-correction mechanisms.
Confident spell checking and spell correction methods for multilingual texts afford powerful dictionaries. We build a dictionary using Apache Lucene, which is known to provide high-performance even for large amounts of full-text data. The dictionary is accessed using the HTTP front-end of the "fully-featured search server" Apache solr[7]. A Lucene index is document based, whereas a document represents a single word extended with its general frequency (statistical knowledge).

[7]http://lucene.apache.org/solr/

The data for the dictionary comes from a German-wikipedia dump of all articles, which is freely available. Such a dump features rare technical terms, abbreviations and, to a certain degree, words from other languages like Dutch, English, French and others.
The fuzzy-search ability achieves *non-word error detection* (spell checking) and *isolated-word error correction* rolled into one. To limit the number of documents that are returned from the solr engine after running a term against it, we specify a minimal edit distance of 90% for potential documents. The corresponding solr query looks like this:

$$value :< errornous\_keyword >\sim 0.9$$

Testing a terms spelling-validity is unsure, since a dump contains misspelled terms too and fuzzy-search may resolve any erroneous-word to a word in the dictionary. Statistical knowledge, like term-frequency in the wikipedia-dump, which is treated as an approximation to the global frequency in the universe, becomes a crucial indicator for correctness. Other term-specific parameter that influence a terms relevance are discussed in section *term relevance* (section 2.2.6).

2.2.4 Query design

In a query based retrieval process, the retrieval performance is totally dependent of the power of the retrieval system beyond. In this context the performance is limited by a query language. Hence, a proper query design, that exploits the entire potential of the query language is crucial. The two ways to access the catalogs' record collection[8] require either SRU (Search and Retrieve via URL) or OpenSearch[9] specification. Both solutions provide RESTful services which use XML as response format. SRU is a retrieval standard for libraries, build on top of Contextual Query Language, that extends the feature set of OpenSearch essentially.

Language

The Contextual Query Language (CQL) [Tay08] features a boolean-logic-like syntax. A CQL-Query is generated from a set of operators (AND, OR and NOT) and a selection of qualified terms. A basic query q with n terms t_i is non-nested and looks like

$$q := t_1\ AND\ t_2\ AND\ \ldots\ AND\ t_n \tag{2.3}$$

SRU returns *MARC-XMLs*[10] results, with a great, granular choice of field-types, called tags[11]. These tags can be considered as semantic

[8] worldcat.org
[9] http://www.opensearch.org/Specifications/OpenSearch/1.1
[10] http://www.loc.gov/standards/marcxml/
[11] http://www.lib.auburn.edu/catalog/docs/marctaglist.html

annotations.

Satisfiability

In complexity theory, the satisfiability problem (SAT) is a decision problem of logical formulas, that describes which variable assignment (instance) of a boolean expression e evaluates to TRUE. Hence, expression e is called satisfiable, if there is at least one variable configuration that results to TRUE. Since the query formula is a conjunction, all sub-statements t_i have to be TRUE in order to satisfy the entire formula e. Expression t_i is true - under the assumption of non-nestedness -, if the value t_i is featured by a designated record-text. A query with misspelled terms will lead, in all likelihood, to a *practically* unsatisfiable formula (logical exclusion).
The challenge for a query formulation strategy is to choose those terms in the card text, that will not raise an unsatisfiable query formula. This probability can be measured and is described by the so-called *term-intersection* problem (p. 19). Some terms should not be used in a query, cause it would be unsatisfiable.
Based on the term intersection we build a model that calculates the probability for a term to be inside the term intersection.

2.2.5 Term intersection

Card-analysis showed a remarkable discrepancy between a card-text and a similar catalog record, considering their terms. Despite the well designed field structure and their homogeneous spelling, there is only a small subset of terms which is covered by both objects, a index card and a particular similar catalog record. This observation is called *term intersection*, defined as

$$term\ intersection:\ terms(index\_card_i)\ \cap\ terms(record_j) \tag{2.4}$$

Regarding the limitations of the query language CQL (e.g. conjunction), a term used in a query must be inside this term intersection, to avoid unsatisfiability.
Unfortunately term intersection can only be measured after a index card is linked to a record, which is not possible. In an extended analysis we identified some extractable features, that allow us to predict the probability for a term to be in the term intersection. This approximation is called *term relevance* and will be explained in the next section. Using the knowledge of *term relevance* we build a model which is basically a scoring function (p. 27). Terms with a high probability achieve a higher

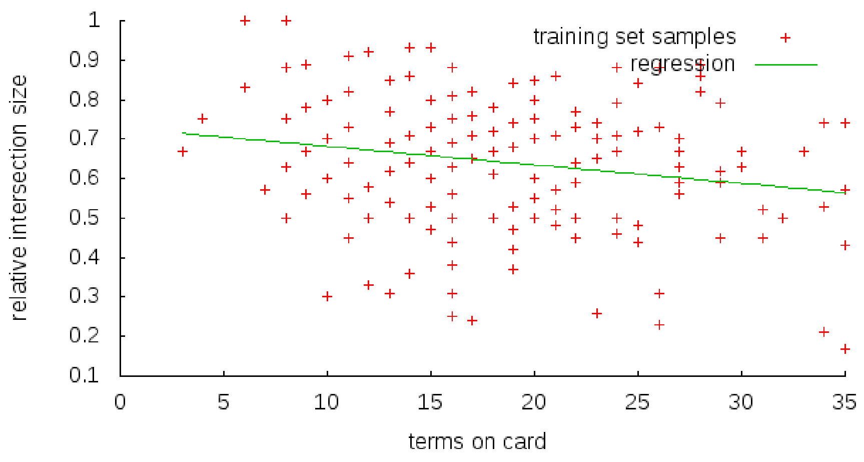


Figure 2.3: The regression line (green) shows that the relative number of terms that can be found on the index card and the record decrease with increasing verbosity of the card-texts.

score than others.

Our estimations for a web-catalog-record to be verbose and an OCR-text to be minimalistic, lead to the following term-relation[12]

$$terms(index\_card_i) \cap terms(record_j) \subset terms(index\_card_i) \quad (2.5)$$

Consequently only a subset of the card-terms are qualified to be used in a query. Terms outside of the intersection cause a logical contradiction and return no hits. The query formula is called *unsatisfiable*.
The intersection size is measured by counting the terms that occur on both sides of a manually linked set of record tuples (index-card and catalog record). Relevant for the intersection measure on catalog-side is the set of indexed fields, the other fields won't affect the query behavior in any way.

Statistics

The average intersection size in the training set is 68%. We recognized a correlation between increase of card-text length and decrease of intersection size (figure 2.3). The reason for that behavior is that the well defined equally written features like title and author are limited.

[12]first-order logic

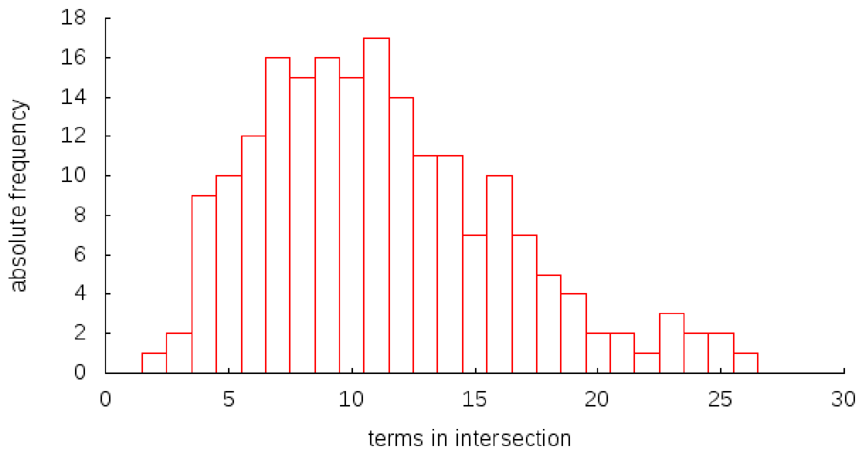


Figure 2.4: The distribution of number of terms in intersection appears to be scattered. More than 95% are above the 5 term-limit, which is a crucial number for *Query formulation* (p. 29). Figure 2.10 shows the same data combined with the total number of parsed terms.

Moreover, in some manual analysis we observed that some terms, that appear to be important, based on the calculated score, though are likely to be invalid are called *untrusted terms*. This observation is highly problem specific. Affected terms are

- Abbreviations inside the publisher field, e.g. Diss, Univ, Forts or Verl
- Notes on an index card, which have been merged into the original text during the OCR process, e.g. Standort.

We suggest a score readjustment for those candidates, to dampen their relevance.

$$score_{untrusted} = 0.95 * score \tag{2.6}$$

2.2.6 Term relevance

A term that is in of the intersection has a high relevance for query-usage. Term relevance is an approximation of the term intersection, based on the following three term features

- probability of correct spellings (independent of the problem setting),
- position in text and
- type of word (both problem dependent)

A high term-relevance postulates a high probability to be inside of the intersection for term t.

In the following two sections we will focus on those features and describe their characteristics.

Independent features

As a general, problem independent relevance-feature we identified the *probability of correct spelling* (1). Due to the fuzzy-word-resolution of the dictionary, plain term-validity checks are not possible at all. Hence, the *proper spelling probability* is derived from extractable features

- *global term frequency*,
- *term-length* and
- similarity of query-term t_{raw} and resolution-term t_{res}. Term t_{raw} is resolved by the dictionary to t_{res} using fuzzy-search. t_{res} is the term with the largest similarity to t_{raw} available in the dictionary.

The global term frequency-value is approximated from term frequency in the wikipedia-dump, during dictionary generation (p. 17). This significant figure is stored in the dictionary as attribute to every term.

In general, term- or word-length are Zipfian distributed [Zip49]. Zipfs' fundamental statement applied on natural language states that longer words are less frequently used than shorter ones.

Consequently, word length and valid spelling are linked. Figure 2.5 shows the correlation of a general power law expression $\frac{1}{x}$ and the distribution of word-length found in the wikipedia dump, mentioned before. Zipf's law is a power law probability distribution, that states that "the frequency of any word is inversely proportional to its rank in the frequency table".

In other words, the probability for a misspelled word to represent by chance a different valid word is much lower if the word is longer. Hence, Term-length is an essential indicator for valid spelling.

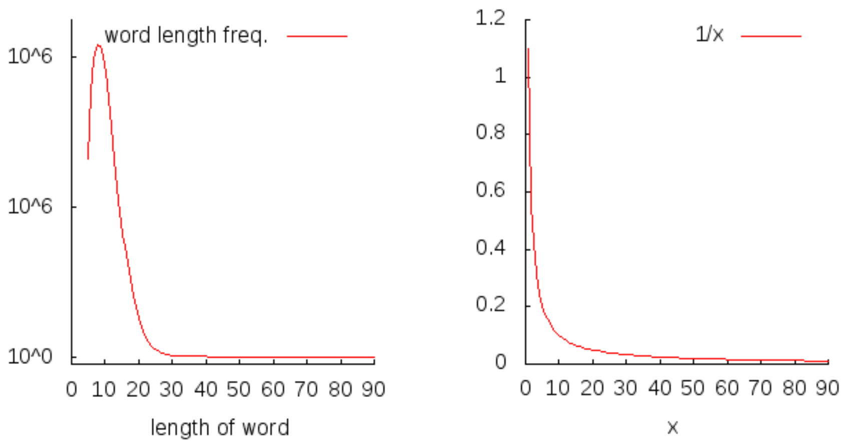


Figure 2.5: The distribution of the word-length in natural languages is zipifan distributed (Zeta distribution). A comparison of word-length distribution in the wikipedia dump (left) and a general power law expression (right), which is similar to Zipf's law.

However, since we use those features in combination with weights, which are optimized, the mentioned independence of the problem setting will be dimished. The final solution will be problem depended.

Dependent features

Two problem related (dependent) features that are used to model the term intersection are *position in text* (2) and *type of word* (3). Both features require an elaborate analysis of the data.

The relevance of *position in text* is determined through the field-order on the card. For example on most index cards the fields title and author, are leading and their terms are likely inside the term intersection (well defined spelling). According to the field-order, the publisher field is, if it exists, always the final one. Content of this particular field is rarely indexed on worldcat, so those terms will not be part of the term intersection. Hence, a leading position of a term term in the card text indicates to be part of the indexed fields title, author or description. Figure 2.6 illustrates the significance of the position feature, based on data from the training set (see p. 3.2.1). Leading terms are in the term intersection.

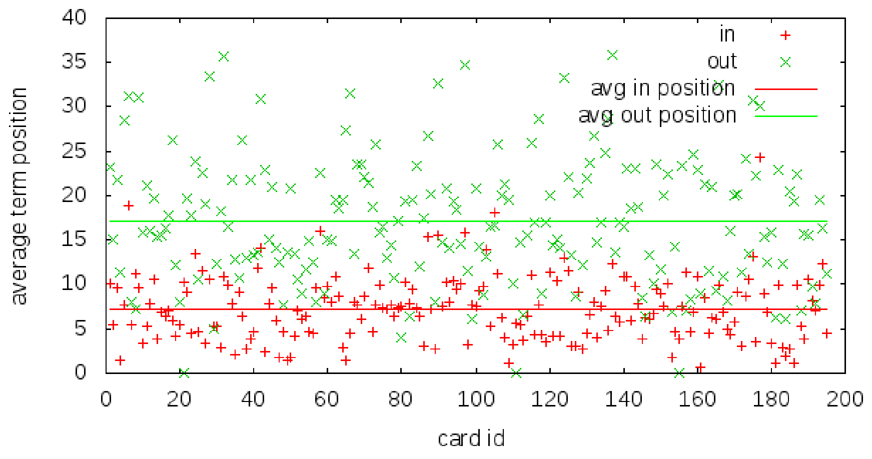


Figure 2.6: The relevance of the problem-dependent feature *position* can be diagnosed by comparing the average position-values of terms inside the intersection (red) and terms outside (green).

The analysis of the feature *type of word* states something similar. Since usage of content of the publisher field is bad practice, terms that are frequently featured in that field, like names of cities, may be downgraded.

2.2.7 Query expansion

Query expansion (QE) [Spi94, Eft96] is a technique to improve the retrieval performance of a query. QE uses several methods to cope with *word mismatch* [WHT$^+$07], defined in citation 2.2.7, a fundamental information retrieval problem.

> "In general, word mismatch refers to the phenomenon in which a concept is described by different terms in user queries and in source documents. Query expansion represents a promising avenue to address such problems." [WHT$^+$07]

The classical QE-approach is interactive, so it requires user interaction. A fully automated QE-adaptation takes the decision of query-improvement based on a probabilistic heuristic.
QE-techniques can be classified into term-related and query-related. Term-related methods improve a query on term-level by suggesting alternate spellings (discussed above) or synonyms, in contrast to query-related methods, with the classical purpose of reweighing the query-terms.
In this thesis we design a fully automated query expansion and reformulation method. All error checked and corrected terms are expanded

with alternate morphological variants using a thesaurus. *Term scoring* orders all terms with respect to their theoretical retrieval performance. Regarding the challenges imposed by term-intersection and query language, we vote the best terms, probably a subset, into a query-relevant *term pool*. This pool supplies a formulation method, that selects terms for query usage, based on a heuristic.
After every query execution a feedback f_i is generated. This feedback is an evaluation of the performance the query achieved. f_i is used by the heuristic to select better terms from the pool in a next query (reformulation p. 32) and learn from failure.
QE is a collection of techniques to increase a query's robustness against failure and hence maximize recall (equation 2.2).

Thesaurus expansion

Several words with almost identical or similar semantic meaning are synonyms. A lexical-semantic thesaurus is a dictionary that maps a term t to its synonyms syn_i. In query expansion, a term may be extended with its synonyms to produce a less-restrictive query q_{weaker}. The fact that q_{weaker} can easier be satisfied induces a reduced probability of logical exclusions (query failure). Thus, the CQL query q looses its flat, non-nested structure

$$q := \ldots\ t_{i-1}\ AND\ t_i\ AND\ t_{i+1}\ AND\ \ldots$$

Term t_i is substituted by the disjunction of t_i and its synonyms. The ordinary query q is modified to

$$q' := \ldots\ t_{i-1}\ AND\ (\ t_i\ OR\ syn_1\ OR\ syn_2\ OR\ \ldots\ OR\ syn_n\ )\ AND\ t_{i+1}\ \ldots$$

This process is called *thesaurus expansion*. Thesaurus expansion is a solution to the *word mismatch* problem, mentioned before (p. 24). Thesauri are of great importance in case of user-generated queries. Thesaurus generation is rather complex, though automatable [CY92].
In context of automated query formulation of documents with a well-known feature set, the possible thesaurus challenges are clear and can be explored. A comprehensive term-intersection analysis pointed out that the major use case is strongly related to the German linguistics. In the German alphabet exist some special characters, e.g. Germanic umlauts, with two almost similar spellings e.g. *ä* as *ae* (escaped). Other accumulations haven't been recognized. This fact allows us to derive a simple rule-based thesaurus on character-level. The case-independent rule-set $(\{ä, ae\}, \{ö, oe\}, \{ü, ue\}, \{ß, ss\})$ defines groups of similar characters. Synonyms syn_i are dynamically generated.

No rule without an exception: The rule based approach tends to fail in some cases, especially in German where ae, oe or ue is not an umlaut per se. *michael* will be altered to *michäl*, which is nonsense.

Thesaurus expansion helps diminish the word mismatch problem [XC96].

Approximate query terms

Two other techniques, aside Thesaurus expansion, that weaken the word mismatch problem are *Approximate string matching* (ASM) and *Common prefix* (CP). Again, their usage in a retrieval process necessitates the support of the query language beyond. The language CQL (p. 18) supports common prefixes.
Approximate string matching (ASM) is a technique to match strings that are close, but not necessarily identical. Especially in context of query based information retrieval, approximate string matching allows a certain error level. [GP06] et al. suggest a so-called k-Approximate String Match, that allows an maximum edit distance of k.

In query-context CP is used to match t_{prefix} and t, whereas t_{prefix} is a prefix of t. The targets are abbreviations on the card text, which are often simply prefixes of the desired word.

On a text of an index card, which is basically a document abstract, abbreviations are common. On catalog-record side, due to its verbosity, they are rare. An abbreviation is a kind of *data truncation* that matches its super-word merely applying fuzzy search mechanisms.

Fuzzy behavior is not implicitly available in CQL, it can be accomplished by appending the asterisk-wildcard $*$ to the term t. The resulting fuzzy subquery $t*$ matches all strings which have t as prefix. Asterisk $*$ represents any string of arbitrary length.

An approximation of query terms results in a better *recall* at a price of a larger number of hits.

Figure 2.7 illustrates the improvements of CP applied on the cards of the training set. For 28% of them we experienced an improvement of term robustness. For those cards, the average increase of the intersection size is 15%, which is about 1.6 terms (average). The charts in that figure show significant improvements for some cards.

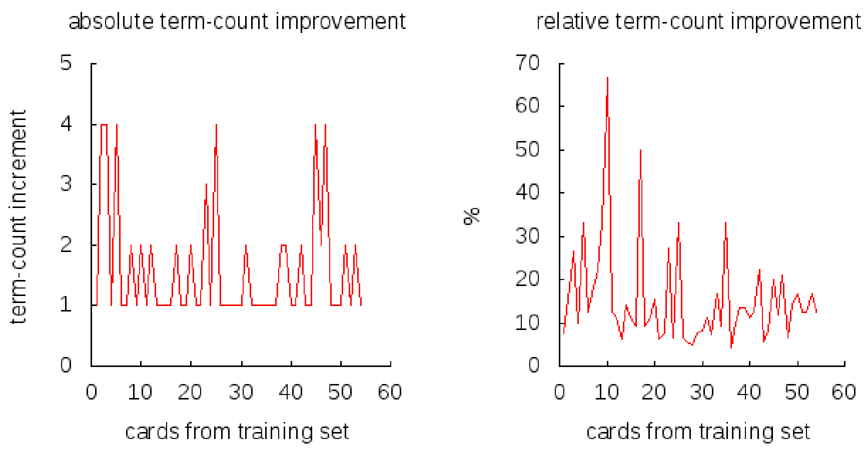


Figure 2.7: Absolute and relative improvements with Common prefixes.

Term scoring and normalization

The approximation of the *term intersection* is finalized by scoring the term relevance features in an appropriate way. The so-called *term scoring* orders the terms of a card text based on a calculated score that represents their particular retrieval potential. The problem of scoring can be solved in a similar way as suggested for *Ranking Retrieval Systems* [SNC01], though with a different purpose.
Scoring is related to the machine learning meta-algorithm called *boosting* [Sch02], in which several weak learners create a single strong learner. In this context, relevance features can be considered as weak learners, the score model as their resulting strong learner.

A combined score is derived from the four relevance features (compare term relevance p. 21)

- *frequency* f
- *length* l
- *position* p and
- *similarity to query keyword* s

which have to be *normalized*, in order to combine them in a balanced way. Every single feature provides an independent sub-score. [WCB06] suggests score normalization, to achieve comparability. f and l are mapped into a [0; 10] interval, using

$$f' = log_{10}(f)$$

and

$$l' = ln(l)$$

f' and l' are the normalizations of f and l. The desired interval for p and s is [0; 1]. p-values range from 0 to about 40. Since p can be assigned with any value that is smaller than the maximum number of terms on a card, a normalization function should be able to handle card texts that are longer than fourty words. Figure 2.10 on page 31 shows the measured term counts in the training set. To make p fit into the desired interval, position values are normalized to p' by

$$p' = \frac{1}{\sqrt[6]{e^p}}$$

This formula, illustrated in figure 2.8, upgrades leading terms, which are to a great likelihood in the intersection. The reasons for the high relevance of position may be reviewed on page 24. Rearmost terms become irrelevant for this sub-score.
Term resolution similarity s is not adapted, since $0 \leq s \leq 1$,

$$s' = s$$

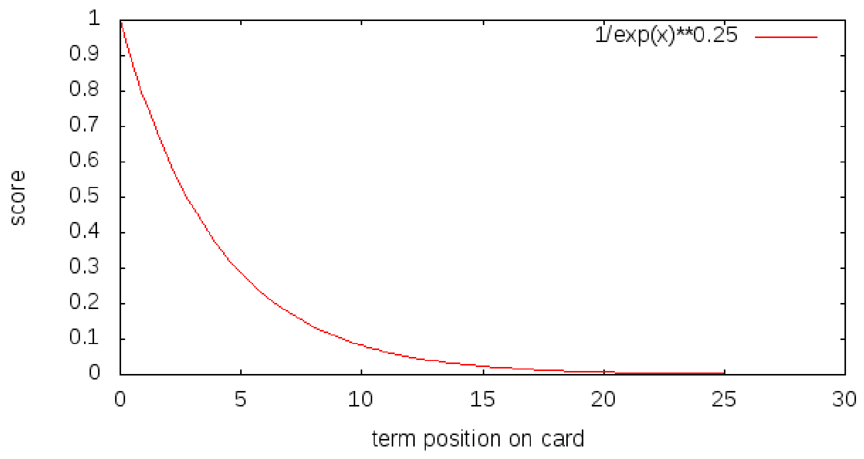


Figure 2.8: The position of a term in the text is crucial for its probability to outperform in a query. Leading position are good indicator for a good retrieval performance (figure 2.6), which drops rapidly.

Both interval groups result in vectors $i = (f', l')$ and $k = (p', s')$ which are normalized again to $\hat{\imath} = (f'', l'')$ and $\hat{k} = (p'', s'')$ (unit vectors, figure 2.7), to dampen system variations. This second normalization is necessary, because the interval for f' and l' is ten times larger than of p' and s'. The transformation to unit vectors of i and k solves this problem.

$$\hat{i} = \frac{i}{|i|} \tag{2.7}$$

Finally, a terms' score can be calculated by summing up the particular values. Since p'' is the core relevance feature, it is amplifyed by 1.5.

$$score := f'' + l'' + 1.5 * p'' + s'' \tag{2.8}$$

If a *score* has been calculated for a term t, t is called *scored term.*

Query formulation

Query formulation (QF) forms a query string (equation 2.3), that satisfies the query design (2.2.4) aspects, using a set of scored terms.

The quality requirements concerning a query are an *acceptable precision* and robustness against terms, that are not in the term intersection (*robustness against invalid terms*).

Precision (definition 2.1) describes the relation between relevant and irrelevant records that have been retrieved. In an ad-hoc system the precision measure can only be estimated by a different extractable characteristic, that correlates with it. We recognized an intuitionally strong correlation between precision and the number of terms in a query. Figure 2.9 shows the average number of hits obtained by a successful query q and the number of terms used in q. The critical lower term-count in q for an acceptable precision is 3, other queries contradict the basic idea of queries. The evaluated queries have been manually formulated for the training set.

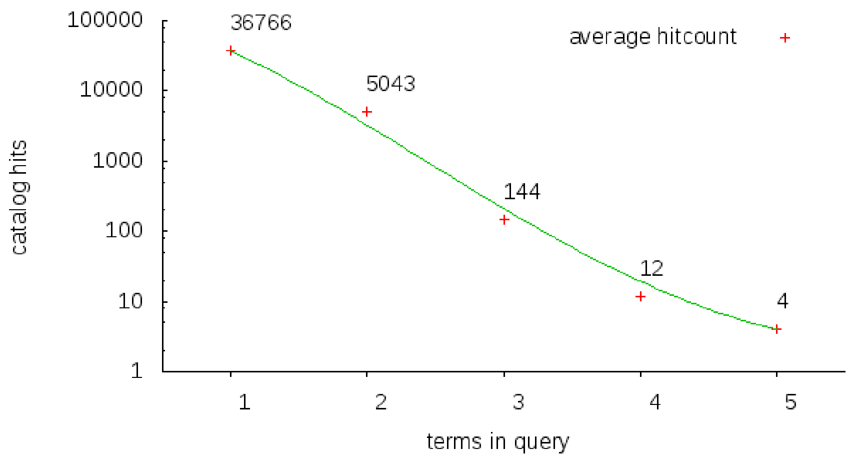


Figure 2.9: A survey of the queries in the training set to detect the minimal average number of terms in a query, that guarantee a acceptable result-set size. In general, at least 3 terms are necessary to meet the query expectations (Figure 2.1)

Robustness against invalid terms reduces the risk that one or more invalid terms in a query lead to a failure of the retrieval process. This is necessary, since we cannot ensure if the term is invalid or not (based on the terms' score) and the query design requires every query term to be valid (in the term intersection of card and record). The designed method formulates queries in an iterative manner (query reformulation), using feedbacks from all previous queries of that particular index card, to detect invalid terms.

Query reformulation chooses terms from a set of optimal-scored terms, called *term pool* (pool). We found out, that a term pool size of *six* is close-to-optimum (Figure 2.10), because it maximizes the query robustness against at most two invalid terms in the query pool. Hence, the worst case assumption of two invalid terms in a pool still leads to an acceptable precision using the residual 4 to 5 terms (compare figure 2.2.7).

For every executed query q, the algorithm calculates a feedback, which symbolizes q's retrieval performance. Based on that feedback, the terms in the query pool are re-scored (compare *term relevance feedback* p. 32). For the initial query, the algorithm chooses a random set of terms, because all terms in the term pool are equally scored.

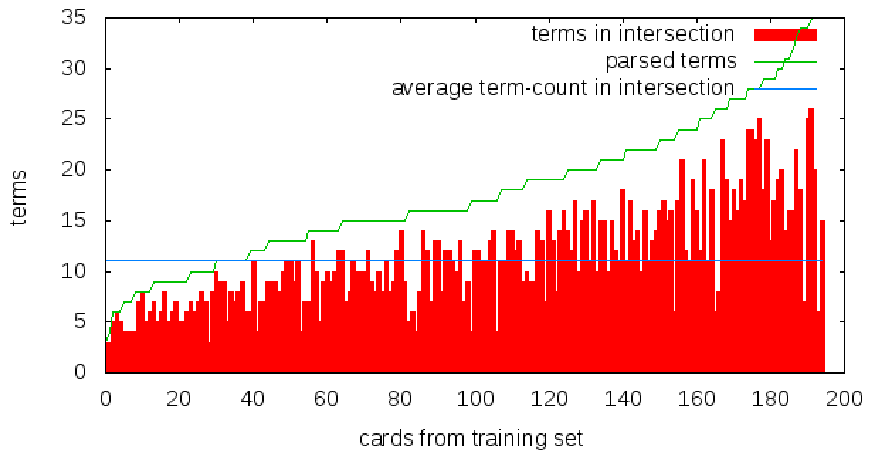


Figure 2.10: Term intersection at one view. Independent of the number of parsed terms (green), an intersection size of *six* is the least common denominator every card can provide. The number of outliers lies at an acceptable rate of 3% (training set), that afford an manual refactoring of the OCR text.

In figure 2.11 we illustrate the measured effects for the cards of the training set, with a term-pool size of 6.

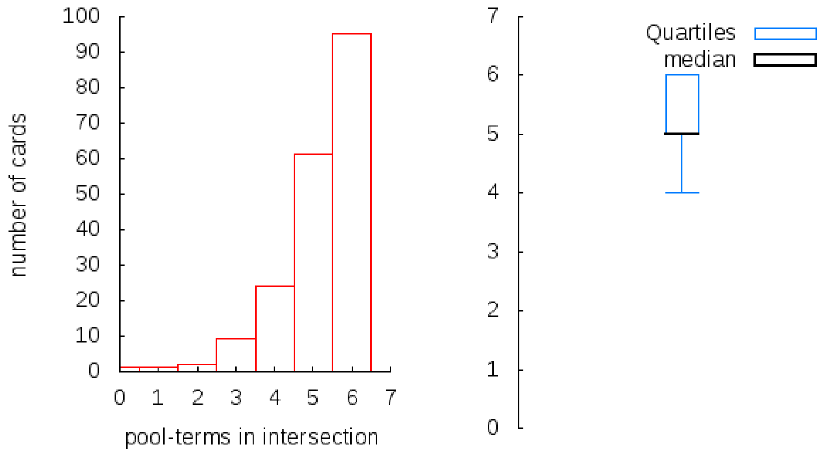


Figure 2.11: Considering a query pool size of 6. Based on the drafted term scoring from above, 80% of all cards (training set) feature a term intersection of *five* or more (Figure left). The Box-Whisker-Plot on the right shows that $\frac{3}{4}$ cards achieve at least *four*.

The *term pool* provides the framework for a reformulation method, which is needed to react on failed queries.
To design a heuristic, that identifies the most promising query (combination of terms) for a predefined query-length k, is not trivial and premises the knowledge of all possible combinations that can be used. This *stochastics problem* is better known as *combinatorics* and can be solved using the *binomial coefficient*, that formally counts the k-combinations of a set S, which is a subset of k distinct elements of S. If S contains n elements, the number of k-combinations is

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

A k-combination is a combination with k elements. For the optimal assignment $k = 3$ (terms in query) and $n = 6$ (pool-size) there exist 20 combinations. Every combination represents a query. The order of the terms in a combination is of no relevance. To find the best performing query we suggest a randomized selection of a query q in the set of the calculated queries. Features like position and term length should not be used to identity the residual bad terms, so the best outcome will be reached by chance.
The query selection strategy is pessimistic and will always try queries with minimal length first. At this point pessimistic means, the algorithm expects a high number of invalid query terms in the pool. In the next section we will introduce a query reformulation strategy, that tries to identify the best performing query in the pool, based on term relevance feedbacks from previous queries.

Term relevance feedback and reformulation strategy

The possibility of bad retrieval performance (recently referred to query failure) requires a query improvement strategy. This reformulation strategy assumes that every index card C has their correspondent similar records R in the web catalog, that can be retrieved by a query generated from terms in the query pool. For the sake of completeness, this assumption does not hold for all cards, just for an assumed 90% of them (in worldcat).
Consequently, we can state, that at least one term combination using terms of the query pool will lead to records R. The suggested reformulation strategy will choose that query (combination of terms) with the highest probability to retrieve R. At this point term relevance feedback is used to speed up the process of searching a satisfying query.
Every executed query returns a feedback about its *retrieval performance.* This *performance* is evaluated using the classic information retrieval

quality measures, precision (definition 2.1 p. 15) and recall (definition 2.2 p. 15). The extractable, relevant query feedback is classified into

- *bad precision*: query retrieves *unmanageable large* number of hits.
- *bad recall*: query retrieves no records, that fulfill the desired criteria e.g. similarity. The decision whether a record is meets the similarity requirements is based on the training set, which contains a set of records ids for every card, that are similar or a similarity measure in combination with a threshold (introduced in chapter *String metrics* p. 39)

Hence, *term relevance feedback* is not directly available (indirect relevance feedback[HR01]) it has to be derived from a query feedback.
Robertson and Sparck Jones [RSJ88] suggest a weighting formula 2.9, that allows us to identify invalid terms in a set (e.g. term-pool), just by interpreting the classified query feedback.

$$w = log\frac{p(1-q)}{q(1-p)} \tag{2.9}$$

and

$$0 < p, q < 1$$

where p is the probability that a document contains the term, given that it is *relevant*, and q is the same probability, given that it is *not relevant.* We apply this formula during the iterative retrieval process, whereas *bad precision* is evaluated as relevance indication for the used n terms (p) and *bad recall* as not-relevance, q. Weight w is calculated for every term in the pool.
Let n be the number of terms used for query $q := \{t_1, \ldots, t_n\}$. Relevance w_i of term $t_i \in q$ is influenced by its relative impact in the query, $\frac{1}{n}$. The particular p and q is recalculated and normalized.

$$p' = \frac{p + \frac{1}{n}}{p+q},\ q' = q \tag{2.10}$$

if and only if q's feedback confirms bad precision.

$$p' = p,\ q' = \frac{q + \frac{1}{n}}{p+q} \tag{2.11}$$

in case of bad recall. p' and q' are the re-assignments for p, q.
In order to achieve the best retrieval potential and determinism, we choose that k-combination C, where $2 < k < 7$, that scores the maximum score-sum T, using the particular combination terms' score w_i

(Robertson Jones weighting, eq. 2.9).

$$T = \sum_{i=1}^{k} w_i \tag{2.12}$$

where

$$w_i \in C$$

Minimize set of potential queries

Reformulation alters the query composition, by exploiting the respective relevances of the pooled terms (automatic query expansion [Eft96]). R. reacts differently on the two kinds of feedback.
Bad precision is interpreted as positive feedback - affects p - which indicates, that a current query is too tolerant. The number of terms n used in a query is increased by one. Moreover, all k-combinations C_{i-}, that are a subset of the current query C_i, $C_{i-} \subset C_i$ are dropped, because the won't provide any benefit.
Bad recall states that at least one of the used query terms is invalid. All terms in this query are downgraded by increasing q (equation 2.12). n is assigned with 3 (minimal query length), to formulate most tolerant queries. Negative feedback for query q triggers an indirect negative feedback for query q', where $q' \subset q$. We can infer, that all super-k-combinations C_{i+} will be invalid too, so they are dropped, $C_i \subset C_{i+}$.

2.3 Implementation and Architecture

The entire query formulation module is implemented in Java[13]. The basic program flow is shown in figure 2.12.

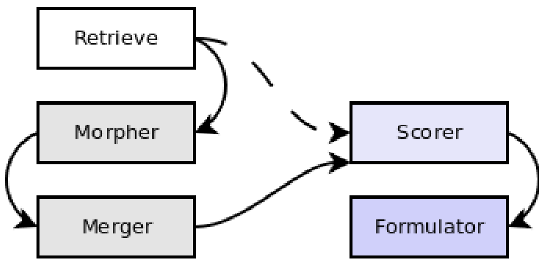


Figure 2.12: Basic program-flow. Since subprocess *Merger* and *Morpher* are optional, there is a dashed line from *Parser* to *Scorer*. This may be appropriate some data, in order to achieve an additional speedup.

In the following text we will describe each component on the flow chart.

[13]www.java.com

2.3.1 Retrieval

OCR documents are accessed via a database connection, as illustrated in figure 2.13 as *Unified Modeling Language* (UML) diagram.

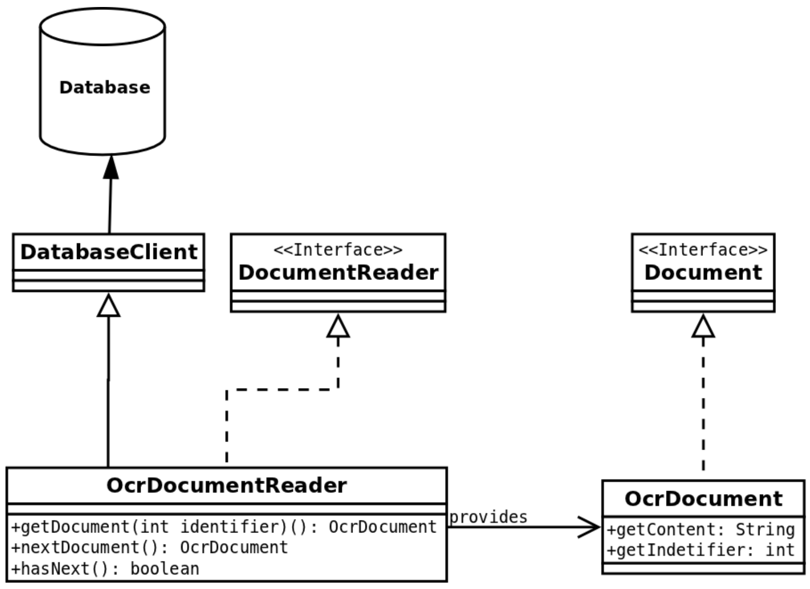


Figure 2.13: Class diagram of the OCR document retrieval architecture using a database as backend.

2.3.2 Parser

The term parser component extracts words using the *java.util.StringTokenizer* class based on a minimalistic set of delimiters,

$$\backslash n \backslash t \backslash r.,;:" - +* \, !? <> ()[]\{\} \tag{2.13}$$

These delimiters mark the beginning and the end of a word (in this context: token). The fuzzy search ability of Lucene resolves spelling errors to a certain degree and performs spell-checking and correction methods (2.2.3).

2.3.3 Morpher and Merger

The morpher module has the purpose to extend parsed words with spelling alternatives. This minimizes the problem of logical exclusion in a query (word mismatch). Those spelling alternativess are the result of the applied thesaurus expansion (2.2.7) or simple substitutions.
In the merging step all terms that are syntactically similar like `Thübungen` and `Thuebingen` are grouped to one query term. The merger component

reduces the total number of terms and ensures that only distinct terms are used in the query-pool, respectively the query.
The execution of morpher and merpher is not obligatory, though it will increase the retrieval performance for cards that are verbose, error prone or very short.

2.3.4 Scorer

The scorer module, introduced in 2.2.7, uses statistical, card-specific knowledge to predict the retrieval potential of a term on an index card. The scorer is deterministic and unable to learn from scoring feedback. The scoring algorithm is derived from a initial feature analysis of the training set. Those n terms with the highest score result in the so-called query-pool (see p. 29). In our tests n is assigned with *six*.

2.3.5 Executor

A query is a permutation generated from subset of term in the query-pool. The class diagram in figure 2.14 gives an insight into the query generation. A *CQLQueryParser* instance generates a query from an OCR document. The query is used by the *searcher*, which has access to the designated catalog.

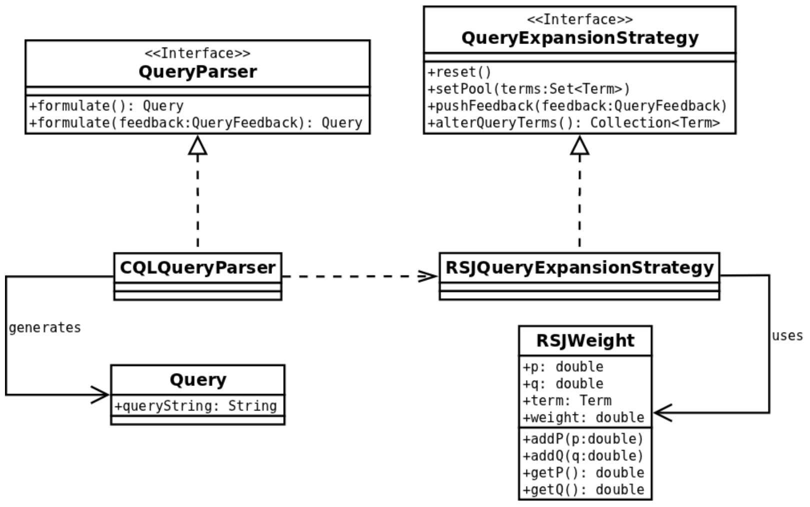


Figure 2.14: Class diagram for QueryParser and query expansion.

The searcher component will execute such a query q and return a feedback (see p. 24), that represents the q's retrieval performance. This feedback is returned to the CQLQueryParser, consequently to the query expansion mechanism, which will rescore the terms in the query-pool (see p. 32).

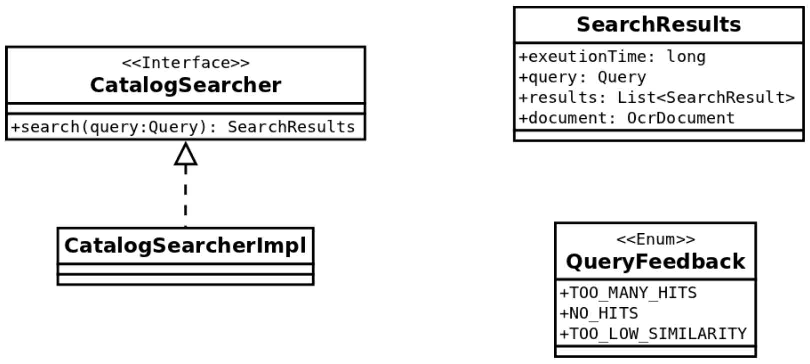


Figure 2.15: The simple classes used in the searcher.

The searcher will stop, if the result set contains matches or the query-pool cannot offer a new, unflagged term permutation. A query q, which is a specific term permutation, is flagged, if the result set is empty or contains just dissimilar records. By flagging a query, all child-permutations q_{child} of q will be flagged too, whereas $terms(q) \subset terms(q_{child})$. Hence, q is less restrictive as q_{child}.
For unmarshalling purposes of the XML-based search results, the popular JDOM[14] API is used (Object/XML mapping). There is a huge load of potential good XML-APIs in the web, though personal experience and the fact that JDOM satisfies all requirements, other solutions like the more lighweight and possibly more efficient XOM[15] have not been considered.

2.4 Results

The suggested method is evaluated and tested for retrieval quality using the 200 cards from the training set. Retrieval quality is determined through *recall* and *precision* (figures 2.2 and 2.1). Additionally the number of queries needed to retrieve at least one similar record should be small. In the predefined training set, every card has a particular set of manually chosen similar catalog records, which are needed to infer whether a record is similar or not (compare p. 42).
From previous analysis we infer a *pool size* with 6 and minimal query length with 3 as optimal. We introduced a *reformulation limit* with value 8, that terminates the reformulation process if its value is reached,

[14]http://jdom.org
[15]http://www.xom.nu/

independently of the retrieved results. We found out that this limit of 8 is sufficient for the vast majority of cards to retrieve at least one similar catalog record, even if two (or less) illegal terms are in the term pool. The reformulation limit is related to the pool size, because it determines the number of k-combinations.
The *reformulation limit* is not obligatory during the execution, though it achieves a good speed-up and meets the performance agreements by disclaiming on the linkage of a few cards.
The suggested query formulation algorithm, is able to retrieve 92% of the training quests correctly. The residual 8% failed either because of their particular bad text quality or an adversarial configuration of the term pool.
The probability for a correct retrieval of a card is, 92%. Since we assume an optimal training set, that features the universe of all cards in an optimal way, a different card set will perform in the same way.

2.5 Discussion

The drafted retrieval approach works satisfying. For the residual 8% the card complexity is to large, no generated query is able to provide a similar record. The reason is always a few number of search terms, which is directly linked to the text quality/length. The term extraction uses the fuzzy search ability of Lucene, so even heavily torn text can resolved to a satisfying bunch of terms. This error-gentle term resolution is a great performance issue, tough for the current set of bad-quality cards, absolutely necessary. In order to correct even those torn 8%, the OCR process should be improved.
Generally speaking, performance improvements can be gained by reducing the costly fuzzy-search method to those terms, which necessitate this treatment.

Chapter 3

String metrics

In mathematics, a *metric* is a function d that defines the distance[1] of two objects X and Y. Hence, a *string metric* is a distance function for character strings.

$$d : X \times X \to \mathbb{R}$$

and

$$d(x,y) \left| \begin{array}{ll} d(x,y) \geq 0 & \text{non-negativity;} \\ d(x,y) = 0 & \text{indeterminable;} \\ d(x,y) = d(y,x) & \text{symmetry;} \\ nd(x,z) \leq d(x,y) + d(y,z) & \text{triangle inequality;} \end{array} \right.$$

The inverse function of distance d^{-1} is called similarity. A similarity measure is always based on the comparison of one or more features (n-dimensional vector). A feature is a measurable condition of an object, in this context a text t, e.g. word w is a feature of t, if $w \in t$. Every relevant characteristic of an arbitrary object o - e.g. field-value or word - may be used as a feature f. In general, f has to be extractable, comparable and of relevance for the distance measure (focus on the feature). The remaining information, that is not used as a feature, degenerates to noise and is immaterial for the metric.
Complex, nested objects can be abstracted by their particular feature-vectors. A text document with an implicit hierarchical structure is an example for a nested object o.
Object o is represented by a feature-vector of length n, whereas n is the number of features. The similarity of two complex objects may be derived from particular similarity measures of their particular feature-vectors.

[1]synonyms: difference and dissimilarity

This chapter solves the similarity-measure-problem of two text-records (objects), whereas one record suffers from a cloaked, not-extractable feature-set. The text-records show the following characteristics

- record one is a full-text from an OCR-processed index card (card-text), which is basically a document abstract, known to be brief, unstructured and erroneous (record abstract)
- the other record is a XML-document from a web-catalog[2] that provides a verbose, granular annotation of text

In order to design a confident similarity metric, all features of interest have to be extracted (uncloaked), cause otherwise two heterogeneous documents cannot be compared in a confident way. We use techniques of plagiarism detection to reveal the designated area of a field value on the index card.
The purpose of the designed record-similarity-metric is record-linkage [FS69]. The decision whether a card-text is linked to a catalog-record depends on their record distance d and a well chosen threshold. The drafted metric is robust against spelling errors and balances differences in length as well as structural disparities. It assumes, that at least one object has extractable features. By identifying qualified matching areas in the card-text, the similarity measures are confident.

3.1 Related work

Measuring the similarity of two texts is a well-known problem, first discussed by [Lev66] in a character-based approach. The Levenshtein-distance (LD) [Lev66, Gus97] counts the minimal number of edit operations needed to transform text A into text B. Valid operations are *insertion*, *deletion* and *substitution*. LD is probably the best-known member of the edit distance (ED) group. [ByJKK04] suggests improved ED-approaches, that work with a time-complexity of $\mathcal{O}(n^2)$ [NW70, WF74] and in linear space [Hir75], which is better than the original ED approach.
Longest common substring (LCS) [BHR00, Gus97], as a special case of ED, restricts the set of edit operations to *match*, *insert* and *deletion*. Since we can delete missing characters, this approach can be faster than the general edit distance approach (LD).
The previously described ED algorithms suffer from displaced-word problem, where the order of the words is changed from text A to text B. A LD approach would charge in worst case $lengthOf(w)$ edit operations for a displaced word w. This results in unreliable distance measures.

[2]http://www.worldcat.org

The Greedy String-Tiling (GST) [Wis93] algorithm solves this displaced-word problem by introducing a block-shifting operation. GST is similar to the Levenshtein distance, except the fact that a bunch of characters (block) can be shifted for costs of 1.
ED-based distance measures become inefficient for longer texts (exponential growth, time-complexity of $\mathcal{O}(n^2)$). Lyon et al. [LMD01] propose a more efficient, statistical method of fingerprinting texts, in order to detect short passages of similar text in large document collections. The fingerprint approach exploits the fact that two different texts on the same subject contain only a small portion of word-trigrams. Shingle words in English and other languages are Zipifan distributed [Mel06], which means that a small number of words are used very often, other appear seldom [Sha51]. Fingerprints are hashes, constrained to the property, that similar text have similar hashes (locality sensitive hashing). The interpretation and comparison of fingerprints can be optimized [Ste05]. The idea of fingerprints has mainly been used for copy or plagiarism detection ([SGm95, Bro97]).
[PW00] states in a similar approach, that a text can be identified on the basis of no more than 5 words. This content based signature is very similar to the fingerprint approach and originally invented to create robust hyperlinks. The system, developed by [Hei96], limits the fingerprint to a 35 to 40 characters and takes us even more closer to the common local-sensitive-hashing approach from above.
[CGPW02] analyzed the issue of text reuse and derivation in context of newspaper and news agencies. Their work MEasuring TExt Reuse (METER project) is build on top of the previously invented meter corpus [GFW$^+$01]. The METER project explored the degree in which newspaper copy text from news agencies and newswire sources. A newspaper journalist could derive a text wholly, partially or not. The similarity measures used include n-gram overlap, greedy string-tiling and sentence alignment. The METER corpus is a supervised learning framework, whereas the different metrics are trained on a training set. N-gram overlap is the straightforward approach of measuring the common words of two texts. Documents are treated as a collection of n-grams. Hapax legomena, which is the same as 1-grams, "has been shown to discriminate plagiarized texts from those written independently even when lexical overlap between texts is already high" ([CGPW02]). Sentence alignment, as abstraction of character alignment which is used in some edit distance approaches [Gus97, Chu93], treats the derived text as a "translation" of the first. These algorithms have been used to map translation equivalents across different languages [D.00]. Brown et al. [BLM91, BE03] apply the sequence alignment approach on monolingual texts to derive their degree of relationship. Alignment is based on word-level-n-grams - a n-gram is a set of n words. The algorithm calculates

an approximate n-gram overlap that optimizes global and local similarity measures. This approach of sentence alignment is very weak against intra word errors, like typos, cause of the word-level n-grams.
The linear algebra based *Vector Space Model* (VSM), introduced by [SWY75], supports similarity and distance measures in same way. The n-dimensional space (n-space) stores an arbitrary object o at a designated position, depending on o's feature-vector. Object features are extractable, in case of a text a feature can be a word. Such a feature vector is mapped to a well-defined point in the space (Cartesian coordinate system). A continuous degree of similarity or distance of two space-object can be computed using cosine similarity or euclidean distance. The vector space model has been criticized for being ad hoc [RW99].
Distance measure algorithms can be parameterized by multiplying the n-dimensional feature vector with a weight-vector $w \rightarrow \mathbb{R}^n$ to achieve better results. Such a optimization assumes the existence of a optimum. This heuristic optimization problem is solved using search strategies like simulated annealing[Kir84], ant colonies[DS04] or genetic algorithms. Pseudo-randomness (indeterminism) is used achieve a robustness against local minima.

3.2 Method

The confident record similarity measure is based on a choice of feature that are given on both records types (compare record characteristics 40), the fields title, authors and description. This feature set reduces the impact of noise of perturbation like verbosity-differences.
The challenge to extract hidden features of card-records is solved using an n-gram overlap approach, similar to the algorithm used in the METER project [GFW$^+$01]. An appropriate distance measure of the feature sets f_{card} and $f_{catalogrecord}$ is weighted to achieve better results. The card's feature vector (title, authors, description) is a fingerprint, hence sufficient to identify a card. Unfortunately there is no guarantee, that all those listed fields will occur. In such a case, a fallback mechanism will be executed.

3.2.1 Training set

In artificial intelligence methods training sets are used to evaluate and optimize (training) algorithms. These sets contain a small representative choice of entries of the universe with the same probability distribution for all relevant features. In supervised learning, a training set evolves to a function, that maps input data to a well-defined, desired result. Training set-based optimizations may suffer from over-fitting, that occurs, if the training set statistical characteristics differs from the universe in a significant way. The result is an algorithm that behaves perfectly on the training set, though underperforms for real data. In our case, the minimal observed requirements to an optimal training to resist overfitting are:

- change of feature order
- change of word order in a feature
- feature position on card
- feature absence

We use a set of 200 entries, to evaluate different distance approaches, which is small compared to the the two million index cards (universe). Therefore, overfitting is likely to occur. The reason for its slim size is found in the complex and time-consuming manual creation. Every entry is maps an index card with its correspondent query to a list of classified catalog ids. The binary classification defines whether a catalog id is similar to the index card ($records_p$), used as input, or not ($records_n$).

$$entry_i := (ID_{card},\ query,\ records_p,\ records_n)$$

whereas

$$records_* := \begin{pmatrix} ID_{catalog_1} \\ ID_{catalog_2} \\ \dots \\ ID_{catalog_n} \end{pmatrix}, records_n \cap records_d = \emptyset$$

An optimal distance approach, calculates confident distance measures, that allow us to derive similarity and dissimilarity in a confident way. The final classification is based on a threshold.
The similarity measures has to score at least one valid record first, elsewise it's a false-positive. As additional challenge, the average-ratio of valid to invalid records is 6.5:10.

3.2.2 Cloaked feature extraction

The essential challenge of the entire similarity measure is to extract the field-values (features) of the card texts. Initially we tried to detect field-ranges on the card using statistical knowledge. This approach requires a elaborate analysis of a representative card selection (training set), which makes it a problem specific solution. However, we found out, that despite consistent field-order of the card-majority, statements about field-value areas on the card are unconfident with chaotic metric results.

Text-feature extraction can basically be solved using the n-gram overlap method, suggest by [GFW+01]. This method facilitates approximate string searching of a text-snipped *needle* in a larger text, the *haystack*, with robustness against shifted text blocks. For that purpose both texts are translated into their particular n-gram representation, a collection of n-grams. In an additional bitmask *match* of size $sizeof(ngrams(haystack))$, $match[i]$ is set TRUE, if the n-gram in the *haystack* at index i exists in *needle*, elsewise FALSE. The best overlap region is determined through most number of TRUE-entries in *match*. Code snippet 3.1 shows the basic steps to generate the *match*-vector.

```
needle = ngrams(needle_text)
haystack = ngrams(haystack_text)
match = new boolean[sizeof(haystack)]

num index = 0
while index < sizeof(haystack)
  match[i] = contains(needle, haystack[i])
  index = index + 1
end

# get overlap-regions by analysing bitmask match
regions = overlap_regions(match)

# region with most TRUE-entries is best
return best_region(regions)
```

Figure 3.1: Pseudocode of *n-gram overlap*, similar to [GFW+01].

Function `overlap_regions` uses the bitmap `match` to detect overlap-regions, with sufficient density of TRUE entries. The size of an *overlap region* is determined by the length of the needle and a jitter parameter (jitter threshold), that allows an overlap to be shorter or longer,

$$overlap\_len_{min} := length(needle) - jitter\_threshold$$

and

$$overlap\_len_{max} := length(needle) + jitter\_threshold$$

The relative jitter-parameter depends on the quality of the cards and the probability of changes in word-order. This may distinguish from field to field.

Overlap regions are detected, by shifting a window (sliding window) of size $overlap_{max}$ through the *match*-array, until the first or the last element in the sliding window is TRUE. If so, the marked region is checked, whether it fulfills the similarity requirements (similarity threshold).

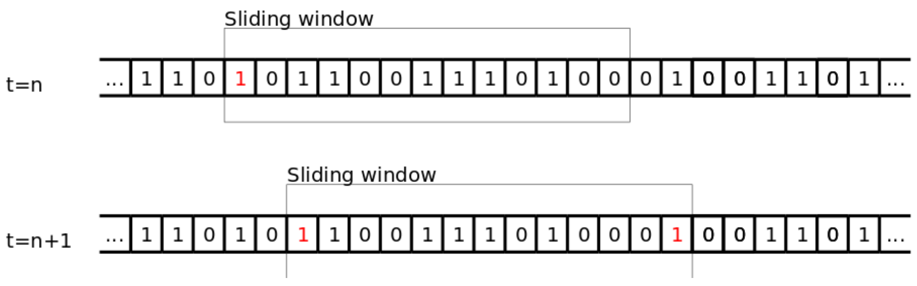


Figure 3.2: sliding window.

Those overlap candidates are optimized to score a better similarity with the needle. Hence, the algorithm looks for smaller overlaps $overlap_i$ inside the candidates, under condition of

$$overlap\_len_{min} < length(overlap_i) < overlap\_len_{max}$$

that either score a higher similarity and are shorter or same similarity and longer. Figure 3.3 demonstrates this optimization process.
Function `best_regions` chooses the best text overlap (text snippet). Here we could use a different similarity-measure approach than n-grams, e.g. character based methods.
In order to gain spelling-error robustness during the overlap calculation, we use n-grams on character level (groups of n consecutive characters). In this context n-grams are normally on word-level. Character n-grams are less efficient.
Our application of the n-gram overlap assumes that potential catalog-record and the index card are similar. Consequently, every common feature-value of the record - title, author, description - an overlap region on the card is calculated.

3.2.3 Feature comparison

All features are plain character strings, suited for classical edit distance-related approach. Since word order is inconsistent, especially for the author-field ('Fredkin Edward' vs 'Edward Fredkin'), metric robustness against *word disorder* is crucial.

$$d_i := distance(f_{card_i}, f_{record_i})$$

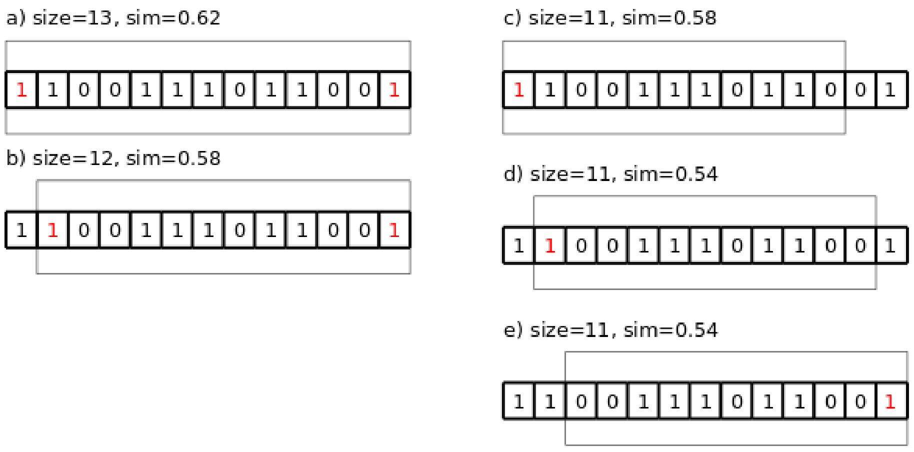


Figure 3.3: jitter threshold is set with 15

In *plagiarism detection* [Clo00], a popular approach to solve the word disorder problem is Greedy String-Tiling[Wis93] (GST). It extends the Levenshtein distance-functionality by a block-shifting operation.
Anyhow, we use the n-gram distance approach, which is - in our case - more efficient than GST, cause the requested n-gram vectors are available from the overlap calculation, performed previously. In the drafted n-gram method, the n-gram order is neglected, merely frequency is relevant. The worst-case charge for a displaced character group is n.
So if d_i is low, this could mean

- *on index cards' side:* card and record are dissimilar, card quality is bad or feature i not available on *card* at all
- *on record side:* feature i not available on *record*

Card-sided conclusions are irrelevant, cause they influence all measures in the same way. Yet, record-sided ones may distortion the metrics' performance, if charged with the calculated n-gram distance of 100% for a missing field. We found out, that a catalog record may lack of a field, withal it is correct and verbose. Let's assume two records r_1, r_2 of the same author in an akin topic.
On record-side missing features can easily been detected, during XML parsing of the catalog response. We suggest a missing-feature distance correction from 100% to 80%.

3.2.4 Feature weights

Since the features f_i have a different significance for the similarity measure, we attach them with individual, optimized problem-specific weights.

This weighting achieves better, more confident similarity results.

$$distance_w := \sum_{i=1}^{n} w_i * d_i \tag{3.1}$$

Missing feature

Missing feature problem describes the problem, that occurs, if a feature is not available on a record. In those cases the maximum distance will be returned for this feature.
In general the availability of the chosen feature set should be reliable. Still, there are some reasons, that may cause the absence of a obligatory field.

- Absence by accident: the assumption for the web catalog to be correct and complete may only hold for a major subset of cards
- Absence by in-verbosity

If a feature f_{r_i} of record r_i is absent, a distance measure with the target feature of the card will be the maximum of 100%. Hence, this calculated dissimilarity may be significantly higher than measures for clear dissimilar features.
The potential danger may cause a serious distance distortion for a minority of records, whereas the score of similar records is drastically degraded just because of a missing feature.
We consider a counter strategy, that charges a default missing-feature-dissimilarity x, assigned with a value between 20 and 60 percent, depending on the OCR-text quality and feature homogeneity. Thereby we hinder a strong downgrading of similar records and at the same time still guarantee a low feature similarity for clearly dissimilar records, because the other features will still show a large dissimilarity.
For completeness, feature-weight optimization won't solve the *missing feature problem* because it is too rare.

Similar features

Another issue, that addresses the correctness of records, occurs if two or more features are similar or identical. On some records we observe that features are assigned with the value of a second feature, probably to avoid a *NULL* value. Since the feature values are identical or even equal, the calculated n-gram overlap will be the same, hence hence their particular feature-similarity.
The effect of the *Similar feature problem* is an amplification of a record towards dissimilarity or similarity with a particular card. An amplified similarity can be problematic, because the record quality is actually not

as good as the scored similarity may indicate. Records with identical features are, in most cases, not intended. Again, this problem will only effect the performance of the algorithm in a negative way, if the card quality is bad or the compared features show variances.

Our suggested solution for the *similar feature problem* exploits the fact, that all card features normally are distinct from each other. Consider c to be a chosen subset of features of record j, that are affected by the similar feature problem. Function $distance'_w$ is the improved weighted distance measure $distance_w$ (definition 3.1) for that problem, defined as,

$$distance'_w := mean(distance(c^2)) * distance_w$$

where c^2 is the Cartesian product $c \times c$. A qualified feature selection for c may be the features $\{author, title\}$, because the *similar feature problem* frequently appeared among those record fields. For some record data, a *threshold* t may be considered in the function $distance'_w$.

$$distance'_w := \begin{cases} distance_w & \text{if } distance_w < t, \\ mean(distance(c^2)) * distance_w & \text{elsewise.} \end{cases}$$

Such a threshold will dampen negative effects of the suggested *similar feature problem* modifications, that will evaluate records with pretty similar feature characteristics (e.g. auto-biography).

3.2.5 Optimization

Algorithms based on weighted features are suitable for further automatic optimizations, in order to achieve better results. Optimization of a function f identifies - on condition that there is an optimum - those input alternatives $A := (a_1, \ldots, a_n)$ that map the best values. Hence, optimization is a search problem in the domain of

$$f : A \to \mathbb{R}$$

In most cases the optimum is also the global maximum/minimum of f. A classification functions g outputs a vector C of classified records. Since $C \notin \mathbb{R}$, it is mapped into $\mathbb{R}$ by an additional function f. An adequate mapping rule for f is the number of false-positives occurred during the classification.

A false-positive is a failed classification, that indicates, that a false record is mistakenly classified as good (positive). Since the metric orders a bunch of catalog records according to their particular similarity to an index card, whereas the most similar one is first, only the leading record of the ordered list is relevant for false-positive evaluation.

Parameterized function

Our goal is the search of the optimum in the domain of a weighted/parameterized function. f is extended through the input-vector W that adjusts particular elements in A, known from section *feature weights* (p. 3.2.4). Consequently W influences C and the calculated classification quality $q \in \mathbb{R}$.

$$f : (A \times W \to C) \to q$$

whereas

$$A := \begin{pmatrix} a_1 \\ \dots \\ a_n \end{pmatrix}, W := \begin{pmatrix} w_1 \\ \dots \\ w_n \end{pmatrix}, C := \begin{pmatrix} c_1 \\ \dots \\ c_n \end{pmatrix}$$

This kind of optimization, which counts the number of false-positives, requires a training set T_{op} (p. 42) that allows us to evaluate a calculated classification with the optimal one. Since A is fixed and the similarity measure for C is deterministic, f will calculated the optimal weights W_{op} according to the optimum defined in T_{op}.

Genetic programming

One outstanding search strategy, that can be used to find an optimal weighting, is genetic programming (GP) [KP05]. Based on well-known methods from evolution, like mutation, recombination and selection, this strategy is able to overcome local optima, even if the input possibilities may be infinite, $|A \times W| = \infty$. A distinct set of genes is called individual. GP uses a fitness function, that maps the quality of the particular gene-set from an individual I to $\mathbb{R}$ (fitness landscape). A simple but powerful fitness function is counting the number of false-positives, generated by a specific W. The better I's fitness, the higher its chances for reproduction.
A GP-based optimization will return the best adapted individual I, which represents the optimal occurrence of W in T_{op}.

3.3 Implementation

The entire *string metrics* module is implemented in Java. In the domain approximate string matching, there is a huge load of open-source libraries. Our designed similarity measure m is an improved approach of the n-gram overlap (view p. 43) from the METER project[GFW$^+$01]. The mentioned improvements lead to a higher error robustness.
During the weight-optimization, m is treated as a black box, that expects weights for every field as input.

3.3.1 Fitness function

We use *JGap* [3] as *genetic programming* framework, which is quite easy to use. JGap requires a fitness function that maps an individual, defined as a set of genes G, to an integer, whereas a large number signals high fitness,

$$fitness : G \rightarrow \mathbb{N}$$

Since we want to optimize the weights of the field-similarity measures, a gene represents a particular weight w_i for feature i. Consequently, a individual I is a vector of those genes,

$$I := \begin{pmatrix} weight_1 \\ \dots \\ weight_k \end{pmatrix}$$

We utilize a fitness function according to the suggestions of [Dye03], who recommends to use the number correct classification on a given training set as fitness value. Since an entry of the training set T_{op} is constrained to feature at least one true-positive and one true-negative record, we extend Dyers' fitness function to evaluate the order of the results and check if all *negatives* are scored after *positives*. The optimal classification C'_i, defined for every $entry_i \in T_{op}$, orders all true-positives (TP) previous to true-negatives. Similarity thresholds are not used.

$$C'_i := \begin{pmatrix} positive_1 \\ \dots \\ positive_n \\ negative_1 \\ \dots \\ negaitve_m \end{pmatrix}$$

The resulting fitness function charges a fee for true-negatives, that are out in the expected order.

$$fitness(i) = correct(i)^{boost} - \sum_{c \in C} misplaced(c)$$

where

$$misplaced(c) \left| \begin{array}{l} \text{0 iff c is true-positive in } C'_i \\ \text{number of TPs ranked after c, elsewise} \end{array} \right.$$

Boosting value *boost* for TPs is assigned with 3, though this depends on the size of C'_i. There exist no surveys considering an assignement of *boost*-value.

[3]http://jgap.sourceforge.net/ Java framework for genetic programming

3.3.2 Caching

Searching for the optimum of a function is time consuming, because many cycles with different parameter (individuals) have to be processed. During the optimization, there are numerous redundant calculations, that can be accelerated though caching. The cache system is composed from two different caches-layer, with different purpose

- Similarity measure cache and
- Web response cache / Permanent cache (3.3.2)

A Cache contains several entries, called cache blocks. Those entries are key-data tuples, whereas a key is used to access and store a designated piece of data. Since some cache frameworks constraint the key length, all keys are *md5*-hashed. In Java, md5 [Riv92] hashes can be generated using the standard message digest algorithm provided by the Java package *java.security.MessageDigest*.
Caching is only necessary, if events occur in a predicable manner (e.g. optimization), that justify its application. Furthermore, the storage of the considered data should be cheaper than a recalculation.

Metric cache

Metric cache is designed to store and retrieve small amounts of data for fast accesses on frequently used data.
This layer is realized as a *java.util.HashMap*. The *Java Virtual Machine* (JVM) beyond stores all data in the Random-Access Memory (RAM), therefore fast.
This metric cache stores all calculated distance measures for every card in the training set, for every field.

Web response cache

This cache layer, persists the response XMLs of the catalog. Those XMLs can become quite large, a RAM-based cache layer may consider memory problems. The access frequency is relatively low, therefore access speed is not so important.
We use *memcached* [Fit04], a distributed memory caching system. Memcached is based on a client-server architecture with TCP-based messages. Data compression combination is used to reduce storage consumption and therefore disk accesses.
Again, the cache blocks are key-data based, whereas a key is a md5-hash of the request url, to fulfill the limited-length constraint for keys in a memcached system.

Beside speedup considerations, caching of web responses is recommended from the legal point of view. The current catalog-license is shipped with a query limit per day.

Speedup

The expected speedup can be estimated through the formula used in parallel computing,

$$S_p = \frac{T_1}{T_p}$$

whereas T_1 is the execution time of the sequential algorithm and T_p the consumed time in a multi-processor system. Adapted to our problem setting, T_1 represents the execution time of the first cycle during the optimization, when no cached data is available. Hence T_p are cycles with cached information.
Considering the training set (figure 3.2.1) speedup potential is gained from caching, not from parallelization. The first cycle, where all cache layers have been empty, takes about 48 seconds ($T_{cycle=1}$). All following cycles show an average execution time of approximately 2 seconds ($T_{cycle>1}$).

$$S_{cache} = \frac{T_{cycle=1}}{T_{cycle>1}} \approx \frac{48}{2} = 24$$

Caching gives us a speedup S_{cache} of approximately 24. The reasons for the huge speedup are

- Slow retrieval of records from catalog or file system used for permanent caching (page 52)
- Complex distance measures

Permanent caching

Since a catalog is an evolving system, with evolving result sets for identical queries, executed at a different time, we use a *permanent caching* layer. Considering the training set from above (p. 42), which provides a classification for all records in the query response of positive and negative, a request-response invariance is crucial. This cache layer persists the MARC XMLs on local file system.

3.4 Architecture

We now present the architectural view of the similarity metrics module. The system is designed to be generic and parts of it can be exchanged, depending on the individual purpose. In the previous chapter we illustrated component views of the query formulation module. Query formulation is designed to acquire a high quality result set. Now, we introduce a module the uses those result sets and evaluates them. We will present diagrams that show how those two modules plug together and consequently how the metrics is designed.

Figure 3.4 shows the major classes involved in the evaluation process.

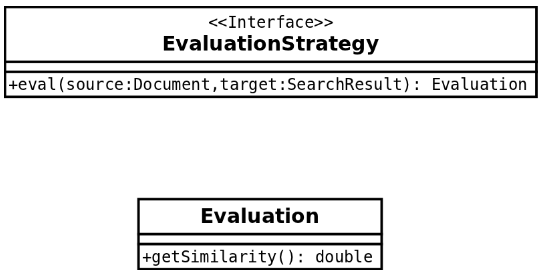
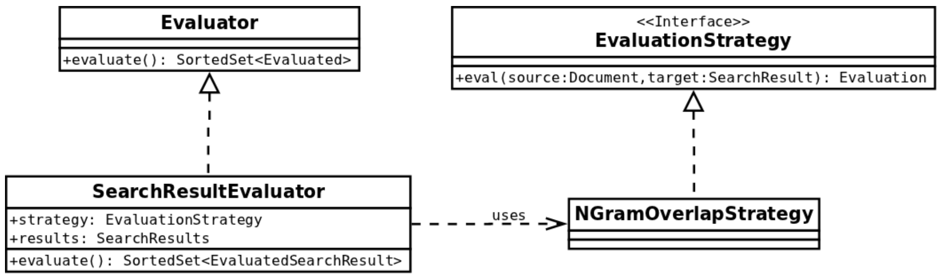


Figure 3.4: The two essential classes in an evaluation.

Class *EvaluationStrategy* is the delivery point for the retrieved result sets, therefore the gateway between the modules *query formulation* and *similarity metrics*. During runtime we use the *NGramEvaluationStrategy* as implementation of *EvaluationStrategy* (Figure 3.5). We use interfaces for controller, to allow simple adaptation and custom implementations of EvaluationStrategy.



Evaluator
+evaluate(): SortedSet<Evaluated>
<<Interface>>
EvaluationStrategy
+eval(source:Document,target:SearchResult): Evaluation
SearchResultEvaluator
+strategy: EvaluationStrategy
+results: SearchResults
+evaluate(): SortedSet<EvaluatedSearchResult>
uses
NGramOverlapStrategy

Figure 3.5: NGramStrategy measures the similarity of the entries in SearchResults and an OCR document.

Class *SearchResultEvaluator* uses the *NGramEvaluationStrategy* as dependecy. *NGramEvaluationStrategy* calculates the n-gram overlaps on the OCR texts, followed by the similarity measures (Figure 3.6). Later, the retrieved search results are ordered according to their similarity with the OCR text. *Evaluation* is a wrapper for the calculated *similarity*.

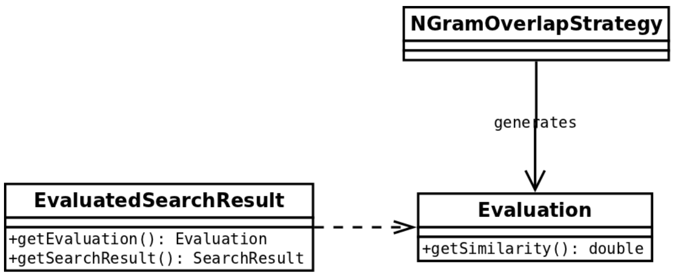


Figure 3.6: An Evaluation carries the crucial similarity measure as double.

3.5 Results

The evaluation of the 200 cards in the training set (p. 3.2.1) are promising. The n-gram overlap approach achieves a false-positive count of 5. Those false-positives occur for two reasons.

- bad quality of full-text (3 cards)

- missing features (compare p. 43), that are essential for the current similarity measure (2 cards), e.g. the author field is not available

The probability for a valid record linking, according to the training set is 2.5%.

The similarity measures are pretty confident, as figure 3.7 shows.

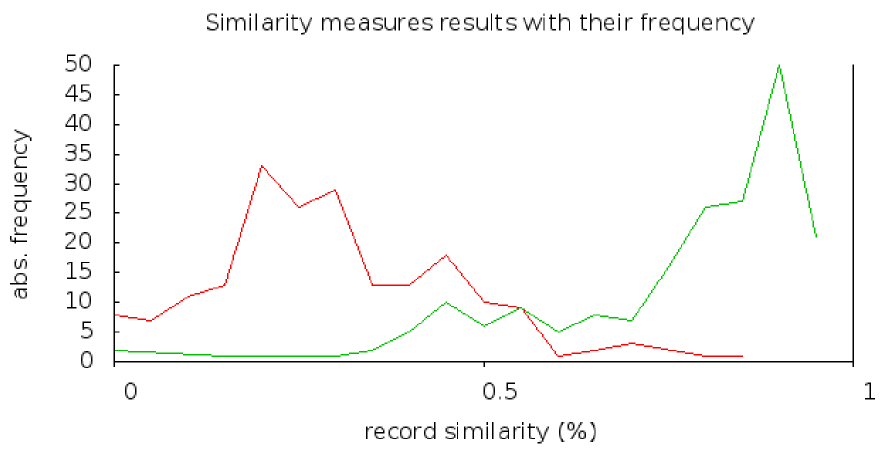


Figure 3.7: Distribution of true positives (green) and true negatives (red). Using the ordinary feature choice, introduced above, records that show a similarity between 40% and 70% cannot be classified and have to be reviewed. Above that range, we can derive, that a record is identical with almost 100% certainty.

3.6 Discussion

The n-gram approach works great to re-locate cloaked features in a torn text. It is heavily robust against misspelling and word disorder.

Regarding similarity measures, this method allows one to use the optimal string similarity measure, depending on the particular purpose. In some cases, it may be appropriate to use a character based approach. Since we are heading towards efficiency, the n-gram based similarity measure comes for (almost) free.

3.6.1 Multiple features

We observed a rare unwanted problem, that leads to awkward results. Some catalog records show the same value for two or more features, and we are not talking about auto-biographies. In any case, the multiple occurrence creates a boosting effect for the similarity measure. Due to the fact, that it is very unlikely to occur we did not handle it in the code. Anyhow we want to suggest a solution to it. similarity of two docs * dissimilarity of all fields: to avoid, that one doc with 2 fields, that are similar s ranked high

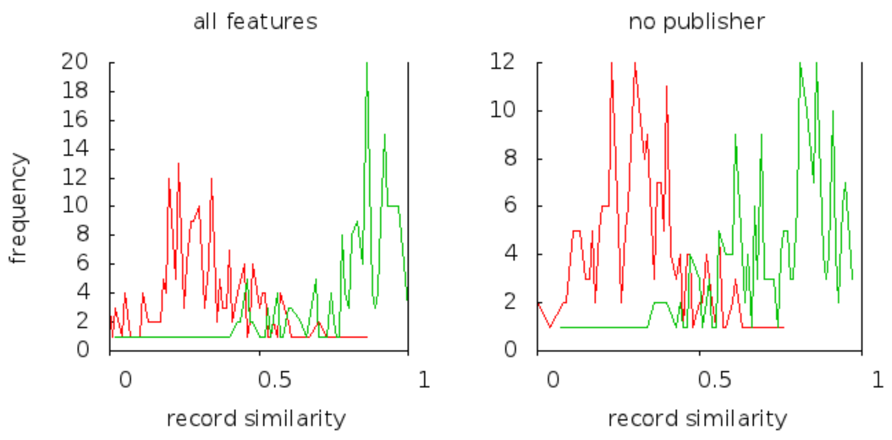


Figure 3.8: Comparing two similarity measures based on different feature sets.

3.6.2 Feature choice

The choice of features is crucial for a success. When dropping the feature *publisher*, the number of false-positives stays the same. Figure 3.8 compares those similarities measures with measures based on all features. As we can see, the overlap size is smaller in the title-author-feature approach. The publisher feature suffers from a great inhomogeneity, there is no persistent writing. This fact disqualifies it for usage in a similarity measure, cause the output is almost random.

Chapter 4

Experiments

In a final experiment we want to explore and measure the behavior of the entire system, where query formulation and similarity metrics are plugged together (i.e. 1.2). The final record linking engine will perform a probabilistic classification of the calculated similarity figures to decide whether an index card and the best retrieved catalog record are *similar*, *probably similar* and *dissimilar*.
The classification uses thresholds obtained from the supervised experiments with the training set (section 4.2.1).
The similarity class *probably similar* can be manually resolved to similar or dissimilar by using the visualizer.

4.1 Implementation and Architecture

In addition to the independent module-implementations, query formulation and similarity metrics, we implemented a third module, the visualizer (Section 4.1.1). The visualizer outputs simple HTML markups using Java Server Pages[1] (JSP) which run in an Apache Tomcat[2] servlet container.
The dependencies are handled using Apache Maven 2[3].
The *system* is standalone and single threaded.

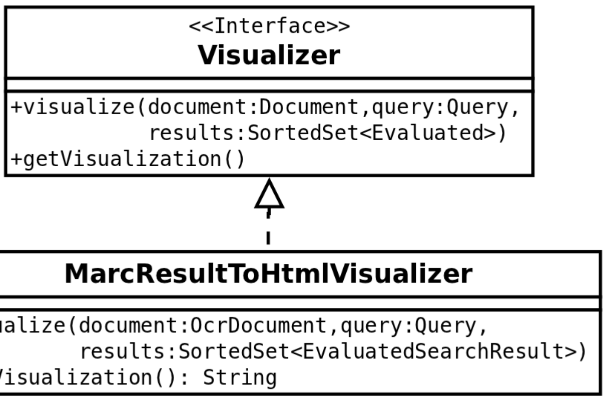
4.1.1 Visualization

The Visualization module is designed for the two purposes

- Supervision of uncertain classifications (probably similar) and
- Demonstration of the algorithm, to provide users an insight into the algorithm.

[1]http://www.oracle.com/technetwork/java/javaee/jsp/index.html
[2]http://tomcat.apache.org/
[3]http://maven.apache.org/

It is web based and visualizes one card result at an instant of time. According to figure 4.1, we build a HTML visualizer, that outputs a valid HTML markup as visualization, that can be viewed in the browser.

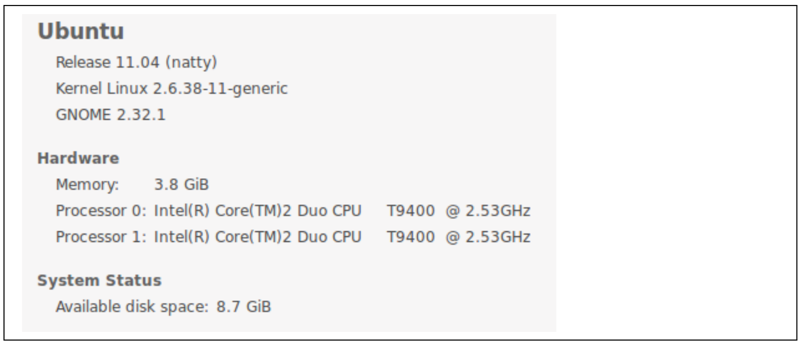


<<Interface>>
Visualizer
+visualize(document:Document,query:Query,
results:SortedSet<Evaluated>)
+getVisualization()
MarcResultToHtmlVisualizer
+visualize(document:OcrDocument,query:Query,
results:SortedSet<EvaluatedSearchResult>)
+getVisualization(): String

Figure 4.1: *MarcResultToHtmlVisualizer* as the default implementation of *Visualizer* generates from OCR document, the query and the evaluated search results an HTML.

4.2 Results

The experiments are run on a set of approximately 280 index cards (experimental set), that differ from those of the training set. The performance of the algorithm is measured through a manual supervision. The execution time measures have been carried out on a system shown in figure 4.2 with no load.



Ubuntu
Release 11.04 (natty)
Kernel Linux 2.6.38-11-generic
GNOME 2.32.1

Hardware
Memory: 3.8 GiB
Processor 0: Intel(R) Core(TM)2 Duo CPU T9400 @ 2.53GHz
Processor 1: Intel(R) Core(TM)2 Duo CPU T9400 @ 2.53GHz

System Status
Available disk space: 8.7 GiB

Figure 4.2: Hardware of the test infrastructure, used in the experiments.

Initially we want to compare those data sets, the training set and the index cards for the experiment, to see, whether they show a similar distribution of the measures. This is crucial, because we use a lot of statistical information of the OCR-texts for the query formulation (Section 2.2.7). This statistical data is obtained from the training set, which should be a small subset of the card universe, represents all major attributes. Figure 4.3 illustrates those sets, whereas all measures are manually classified into similar (green) and not similar (red).

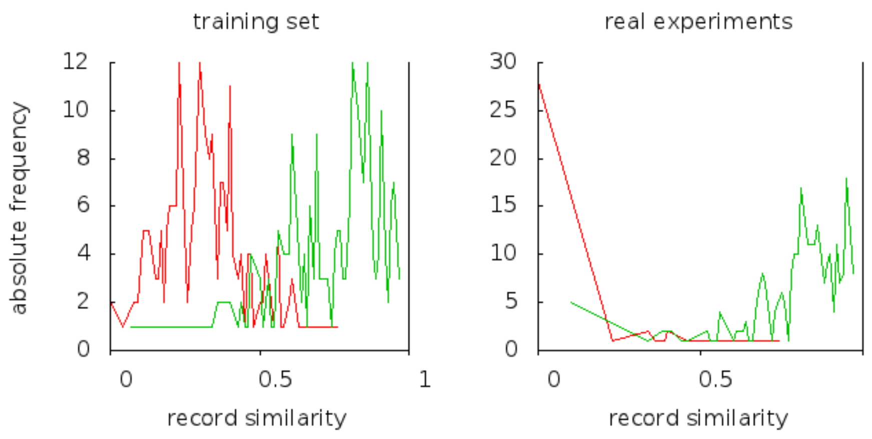


Figure 4.3: Similarity histogram comparing the trainig set and the real experiments.

The figure shows a much better performance during the experiment with an accumulation around 0. This indicates that for those cards the query formulation failed. The reasons for the failed query formulation can be either

- bad quality of the full-text
- worldcat does not contain these cards.

4.2.1 Classification thresholds

The crucial point for a working classification method is its confidentiality. We differentiate three different classes of a similarity measure

- similar,
- probably similar and
- dissimilar.

On the similarity measures in figure 3.8 we recognize a large intersection of measures, that show dissimilar and similar records. This range of intersection goes from a similarity of 7% to 76%, which is a dominant range. Based on that critical area, we try to define two thresholds, that mark a *certainly positive* and *certainly negative* measure. The threshold *certainly positive* is defined by the highest similarity of a true negative in training set, which is 76%. Consequently, lowest similarity of a true positive of 7% corresponds with the *certainly negative* threshold. Nevertheless, a similarity of 7% is that minor, that one may even speak of dissimilarity for even higher similarities. We reassign *certainly negative* threshold with 30%, which is just an approximated value from the illustration (3.8). Similarity measures that lie inside range between those thresholds will be classified as *probably similar.* These records afford a manual supervision, probably using the visualizer, introduced before,

$$sim(card_i, record_j) \left| \begin{array}{ll} \geq 76 & \text{similar} \\ \leq 30 & \text{dissimilar} \\ else & \text{undecidable} \end{array} \right.$$

The bar chart 4.4 illustrates the relative distribution of measures after being classified. Hence 10% of the 280 cards used in the experiment could not be linked to a catalog record, 30% are doubtful and 60 are definitely valid, according to the thresholds.

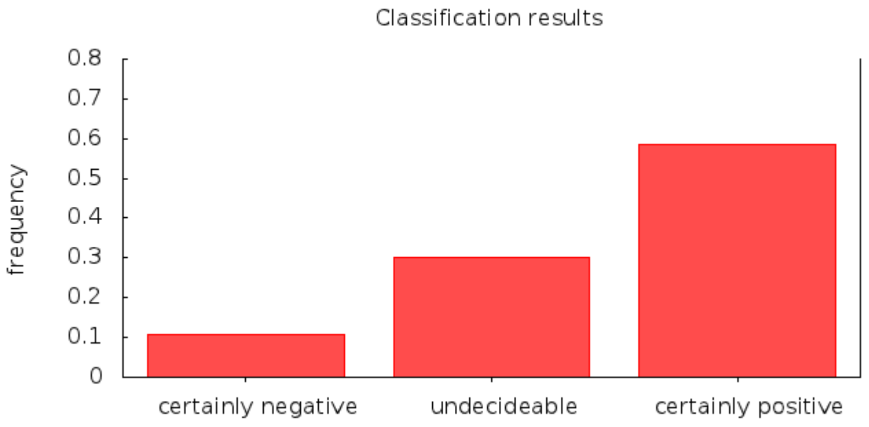


Figure 4.4: Histogram of the classified similarities using the threshold limits defined above.

We configured an termination query count of *seven*, the reformulation process stops after seven tries, or earlier. The algorithm will stop searching for records, if there are no combinations left to formulate a query, or if at least one search results is *certainly positive.* In the case of seven retries, the average number of queries to terminate is *3.3*. Figure 4.5 shows a detailed histogram about the query count.

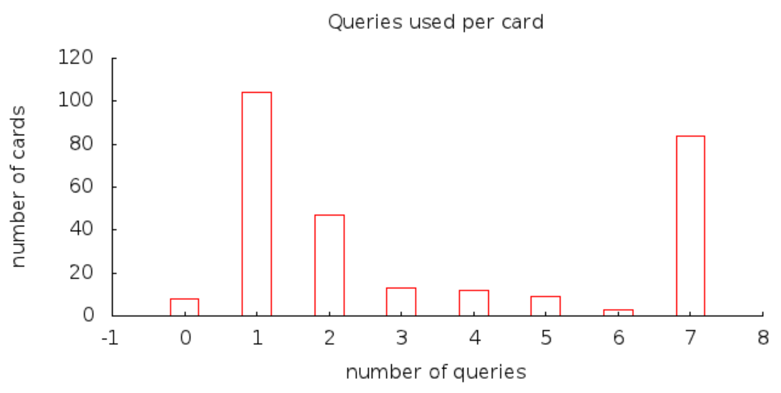


Figure 4.5: Histogram of the number of queries per card until the process terminates. Accepted number of retries was six.

As we can see, the number of retries is crucial, if the thresholds are relatively weak. Consequently, the query-time, which stands for the time until the query process terminates. In average, the algorithm needs 14 seconds to classify a card.

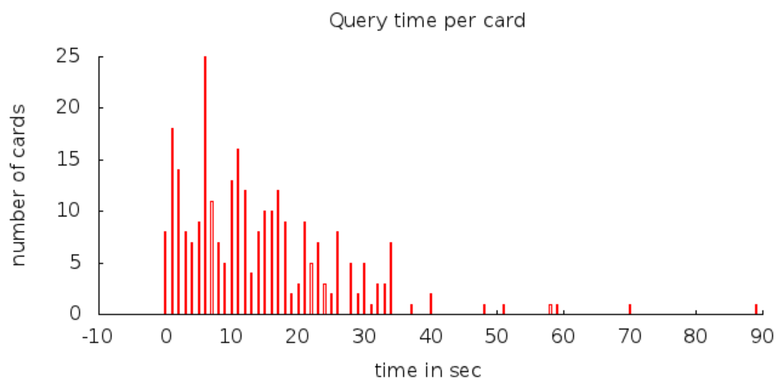


Figure 4.6: Distribution of query-times, needed until the retrieval process stops.

4.3 Conclusion

The results of the algorithm after performing record linking on a set of 280 cards is promising. The query formulation, as the most tricky part of the process, was able to retrieve a huge number of similar records, even if the OCR text beyond was pretty error prone. A manual review of the cards the did not succeed in the query formulation process featured mostly a bad OCR text. Concerning the string similarity metrics, we want to state that taking the decision whether an index card and a record are similar is even hard to take for humans. Hence, a proper definition of the threshold limits is crucial.

Appendix

A.1 Wikipedia Dump Parser

The Wikipedia dump has been used to generate a powerful, for the most part error free dictionary. Those dumps can become sizes of 10 GB and more. Inside a dump, the data is encoded as XML. Current desktop computers or laptop are not able to process such amount of data, because the process requires much more memory. The machine we worked on has four GB memory. Normaly, when you are going to parse a XML document, you can use either

- DOM or
- SAX parsers.

Nevertheless, both of them will fail, for good reasons.
The solution to that problem are so-called *StreamReader*, that can iterate through a XML document (wikipedia dump) and trigger events like

START_ELEMENT, *END_ELEMENT* or *CHARACTERS*.

In our case we used the *XMLStreamReader* provided by a *XMLInputFactory* part of the *javax.xml.stream* package. The experiences are very positive all our expectations have been satisfied.

Bibliography

[AS07] N. Askitis and R. Sinha: *HAT-trie: a cache-conscious trie-based data structure for strings*, *Proceedings of the thirtieth Australasian conference on Computer science - Volume 62*, ACSC '07, Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pages 97–105.

[BE03] R. Barzilay and N. Elhadad: *Sentence alignment for monolingual comparable corpora*, *Proceedings of the 2003 conference on Empirical methods in natural language processing*, Association for Computational Linguistics, Morristown, NJ, USA, pages 25–32.

[BG96] S. A. M. M. Buckley C. and S. G.: *New Retrieval Approaches Using SMART: TREC 4*, (1996), pages 25–48.

[BHR00] L. Bergroth, H. Hakonen and T. Raita: *A Survey of Longest Common Subsequence Algorithms*, *SPIRE '00: Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00)*, IEEE Computer Society, Washington, DC, USA, page 39.

[BLM91] P. F. Brown, J. C. Lai and R. L. Mercer: *Aligning sentences in parallel corpora*, *Proceedings of the 29th annual meeting on Association for Computational Linguistics*, Association for Computational Linguistics, Morristown, NJ, USA, pages 169–176.

[BM03] M. Bilenko and R. J. Mooney: *Adaptive duplicate detection using learnable string similarity measures*, *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, New York, NY, USA, pages 39–48.

[BMC$^+$03] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar and S. Fienberg: *Adaptive Name Matching in Information Integration*, IEEE Intelligent Systems, volume 18, (2003), pages 16–23.

[Bro97] A. Z. Broder: *On the Resemblance and Containment of Documents*, *In Compression and Complexity of Sequences (SEQUENCES97*, IEEE Computer Society, pages 21–29.

[BS93] T. Bäck and H.-P. Schwefel: *An Overview of Evolutionary Algorithms for Parameter Optimization*, Evolutionary Computation, volume 1(1), (1993), pages 1–23, URL `http://www.mitpressjournals.org/doi/abs/10.1162/evco.1993.1.1.1`.

[ByJKK04] Z. Bar-yossef, T. S. Jayram, R. Krauthgamer and R. Kumar: *Approximating edit distance efficiently*, *In Proc. FOCS 2004*, IEEE, pages 550–559.

[CGPW02] P. Clough, R. Gaizauskas, S. S. L. Piao and Y. Wilks: *METER: MEasuring TExt Reuse*, *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, Association for Computational Linguistics, Morristown, NJ, USA, pages 152–159.

[Chu93] K. W. Church: *Char_align: a program for aligning parallel texts at the character level*, *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, Association for Computational Linguistics, Morristown, NJ, USA, pages 1–8.

[Clo00] P. Clough: *Plagiarism in natural and programming languages: an overview of current tools and technologies*, 2000.

[CT94] W. B. Cavnar and J. M. Trenkle: *N-grambased text categorization*, *In Proc. of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175.

[CY92] C. J. Crouch and B. Yang: *Experiments in automatic statistical thesaurus construction*, *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '92, ACM, New York, NY, USA, pages 77–88.

[D.00] W. D.: *Alignment.*

[Dam64] F. J. Damerau: *A technique for computer detection and correction of spelling errors*, Commun. ACM, volume 7(3), (1964), pages 171–176.

[Doe09] R. Doe: *Carl Linnaeus Invented The Index Card @ONLINE*, 2009, URL `http://www.sciencedaily.com/releases/2009/06/090616080137.htm`.

[DS04] M. Dorigo and T. Stützle: *Ant Colony Optimization*, Bradford Company, Scituate, MA, USA, 2004.

[DS10] M. Diem and R. Sablatnig: *Recognizing characters of ancient manuscripts*, 2010, URL `http://link.aip.org/link/?PSI/7531/753106/1`.

[Dun94] T. Dunning: *Statistical Identification of Language*, Technical report, 1994.

[Dye03] C. R. Dyer: *Genetic Algorithms (Chapter 4.1.4)*.

[eae08] K. A. et. al. et. al.: *GNU Aspell, A library providing spell checking @ONLINE*, 2008, URL `http://aspell.net/`.

[Eft96] E. N. Efthimiadis: *Query Expansion*, *Annual Review of Information Systems and Technology (ARIST), v31*, pages 121–187.

[Fit04] B. Fitzpatrick: *Distributed Caching with Memcached*, Linux Journal, (124), (2004), pages 72–74,76,78, URL `http://www.linuxjournal.com/article/7451`.

[Fre60] E. Fredkin: *Trie memory*, Commun. ACM, volume 3, (1960), pages 490–499, URL `http://doi.acm.org/10.1145/367390.367400`.

[FS69] I. P. Fellegi and A. B. Sunter: *A Theory for Record Linkage*, Journal of the American Statistical Association, volume 64(328), (1969), pages 1183–1210.

[GFW$^+$01] R. Gaizauskas, J. Foster, Y. Wilks, J. Arundel, P. Clough and S. Piao: *The meter corpus: A corpus for analysing journalistic text reuse*, *In*, pages 214–223.

[GP06] S. Gollapudi and R. Panigrahy: *A dictionary for approximate string search and longest prefix search*, *Proceedings of the 15th ACM international conference on Information and knowledge management*, CIKM '06, ACM, New York, NY, USA, pages 768–775, URL `http://doi.acm.org/10.1145/1183614.1183723`.

[Gus97] D. Gusfield: *Algorithms on strings, trees, and sequences: computer science and computational biology*, Cambridge University Press, New York, NY, USA, 1997.

[HA03] V. J. Hodge and J. Austin: *A Comparison of Standard Spell Checking Algorithms and a Novel Binary Neural Approach*, IEEE Transactions on Knowledge and Data Engineering, volume 15, (2003), pages 1073–1081.

[Ham50] R. W. Hamming: *Error Detecting and Error Correcting Codes*, The Bell System Technical Journal, volume 26(2), (1950), pages 147–160.

[Hei96] N. Heintze: *Scalable Document Fingerprinting*, *In Proc. USENIX Workshop on Electronic Commerce*.

[Hir75] D. S. Hirschberg: *A linear space algorithm for computing maximal common subsequences*, Commun. ACM, volume 18(6), (1975), pages 341–343.

[HR01] D. Hiemstra and S. Robertson: *Relevance Feedback for Best Match Term Weighting Algorithms in Information Retrieval*, *Dublin City University*, pages 37–42.

[HYKL08] Y. Hong, T. Yang, J. Kang and D. Lee: *Record Linkage as DNA Sequence Alignment Problem*, 2008.

[HZW02] S. Heinz, J. Zobel and H. E. Williams: *Burst Tries: A Fast, Efficient Data Structure for String Keys*, ACM Transactions on Information Systems, volume 20, (2002), pages 192–223.

[J.95] P. J.: *The Generative Lexicon*.

[Jo03] T. Jo: *Neural Based Approach to Keyword Extraction from Documents*, V. Kumar, M. Gavrilova, C. Tan and P. LEcuyer (eds.), *Computational Science and Its Applications ICCSA 2003*, volume 2667 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2003, pages 963–963, URL `http://dx.doi.org/10.1007/3-540-44839-X_49`.

[KDS10] F. Kleber, M. Diem and R. Sablatnig: *Document reconstruction by layout analysis of snippets*, 2010, URL `http://link.aip.org/link/?PSI/7531/753107/1`.

[KG06] J. Korst and G. Geleijnse: *Efficient Lyrics Retrieval and Alignment*, 2006.

[Kir84] S. Kirkpatrick: *Optimization by simulated annealing: Quantitative studies*, Journal of Statistical Physics, volume 34, (1984), pages 975–986, URL `http://dx.doi.org/10.1007/BF01009452`, 10.1007/BF01009452.

[KP05] J. Koza and R. Poli: *Genetic Programming*, E. K. Burke and G. Kendall (eds.), *Search Methodologies*, Springer US, 2005, pages 127–164.

[Kuk92] K. Kukich: *Techniques for automatically correcting words in text*, ACM Comput. Surv., volume 24, (1992), pages 377–439.

[Lev66] V. Levenshtein: *Binary Codes Capable of Correcting Deletions, Insertions and Reversals*, Soviet Physics Doklady, volume 10, (1966), page 707.

[LMD01] C. Lyon, J. Malcolm and B. Dickerson: *Detecting Short Passages of Similar Text in Large Document Collections*, *PROCEEDINGS OF THE 2001 CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING*, pages 118–125.

[MA04] A. S. M. Marciniszyn and A.Weißs: *E-Jigsaw - Computergestützte Rekonstruktion zerrissener Stasi-Unterlagen*, Informatik Spekturm, (2004), pages 248–254.

[Mai78] D. Maier: *The Complexity of Some Problems on Subsequences and Supersequences*, J. ACM, volume 25(2), (1978), pages 322–336.

[Mel06] A. Mellor: *G.K. Zipf, The Psycho-biology of Language, MIT Press, Cambridge, MA. (1935).*, System, volume 34(3), (2006), pages 455–457, URL `http://dx.doi.org/10.1016/j.system.2006.07.004`.

[MI03] Y. Matsuo and M. Ishizuka: *Keyword Extraction from a Single Document using Word Co-occurrence Statistical Information*, International Journal on Artificial Intelligence Tools, volume 13, (2003), page 2004.

[MNY99] S. Mori, H. Nishida and H. Yamada: *Optical Character Recognition*, John Wiley and Sons, Inc., New York, NY, USA, 1999.

[Mor68] D. R. Morrison: *PATRICIA - Practical Algorithm To Retrieve Information Coded in Alphanumeric*, J. ACM, volume 15, (1968), pages 514–534.

[NDH08] T. R. Nielsen, P. Drewsen and K. Hansen: *Solving jigsaw puzzles using image features*, Pattern Recognition Letters, volume 29(14), (2008), pages 1924–1933.

[NS06] B. Nickolay and J. Schneider: *Automatische virtuelle Rekonstruktion vernichteter Dokumente*, (2006), pages 6–7.

[NW70] S. B. Needleman and C. D. Wunsch: *A general method applicable to the search for similarities in the amino acid sequence of two proteins.*, Journal of molecular biology, volume 48(3), (1970), pages 443–453, URL `http://view.ncbi.nlm.nih.gov/pubmed/5420325`.

[PW00] T. A. Phelps and R. Wilensky: *Robust Hyperlinks: Cheap, Everywhere, Now*, *DDEP/PODDP*, pages 28–43.

[Riv92] R. Rivest: *The MD5 Message-Digest Algorithm*, 1992.

[RSJ88] S. E. Robertson and K. Sparck Jones: *Relevance weighting of search terms*, Taylor Graham Publishing, London, UK, UK, 1988, pages 143–160, URL `http://portal.acm.org/citation.cfm?id=106765.106783`.

[RW99] V. V. Raghavan and S. K. M. Wong: *A critical analysis of vector space model for information retrieval*, Journal of the American Society for Information Science, volume 37(5), (1999), pages 279–287.

[SB88] G. Salton and C. Buckley: *Term-weighting approaches in automatic text retrieval*, *INFORMATION PROCESSING AND MANAGEMENT*, pages 513–523.

[SBP05] P. D. Smet, J. D. Bock and W. Philips: *Semiautomatic reconstruction of strip-shredded documents*, International Society for Optical Engineering; 1999.

[Sch02] R. E. Schapire: *The Boosting Approach to Machine Learning: An Overview*, 2002.

[SGm95] N. Shivakumar and H. Garcia-molina: *SCAM: A Copy Detection Mechanism for Digital Documents*, *In Proceedings of the Second Annual Conference on the Theory and Practice of Digital Libraries.*

[Sha51] C. E. Shannon: *Prediction and entropy of printed English*, Bell Systems Technical Journal, volume 30, (1951), pages 50–64.

[SM86] G. Salton and M. J. McGill: *Introduction to Modern Information Retrieval*, McGraw-Hill, Inc., New York, NY, USA, 1986.

[SNC01] I. Soboroff, C. Nicholas and P. Cahan: *Ranking retrieval systems without relevance judgments*, *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '01, ACM, New York, NY, USA, pages 66–73, URL `http://doi.acm.org/10.1145/383952.383961`.

[Spa88] K. Sparck Jones: *A statistical interpretation of term specificity and its application in retrieval*, (1988), pages 132–142.

[Spi94] A. Spink: *Term relevance feedback and query expansion: relation to design*, *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '94, Springer-Verlag New York, Inc., New York, NY, USA, pages 81–90, URL `http://portal.acm.org/citation.cfm?id=188490.188511`.

[Ste05] B. Stein: *Fuzzy-Fingerprints for Text-Based Information Retrieval*, K. Tochtermann and H. Maurer (eds.), *Proceedings of the 5th International Conference on Knowledge Management (I-KNOW 05), Graz, Austria*, Journal of Universal Computer Science, Know-Center, pages 572–579.

[SWY75] G. Salton, A. Wong and C. S. Yang: *A vector space model for automatic indexing*, Commun. ACM, volume 18, (1975), pages 613–620, URL `http://doi.acm.org/10.1145/361219.361220`.

[Tay08] M. Taylor: *CQL: Common Query Language @ONLINE*, 2008, URL `http://zing.z3950.org/cql/index.html`.

[TE96] X. Tong and D. A. Evans: *A Statistical Approach to Automatic OCR Error Correction In Context*, *Proceedings of the Fourth Workshop on Very Large Corpora (WVLC-4*, pages 88–100.

[Tse98] Y.-H. Tseng: *Multilingual keyword extraction for term suggestion*, *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '98, ACM, New York, NY, USA, pages 377–378.

[Tyb04] R. Tybon: *Generating Solutions to the Jigsaw Puzzle Problem*, Ph.D. thesis, Griffith University, Australia, 2004.

[TZMFE96] X. Tong, C. Zhai, N. Milic-Frayling and D. A. Evans: *OCR Correction and Query Expansion for Retrieval on OCR Data - CLARIT TREC-5 Confusion Track Report*, 1996.

[Vit03] A. Viterbi: *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm*, Information Theory, IEEE Transactions on, volume 13(2), (2003), pages 260–269.

[WCB06] S. Wu, F. Crestani and Y. Bi: *Evaluating Score Normalization Methods in Data Fusion*, *Information Retrieval Technology, AIRS 2006*, pages 642–648, URL `http://dx.doi.org/10.1007/11880592_57`.

[Wes00] D. B. West: *Introduction to Graph Theory (2nd Edition)*, Prentice Hall, 2000.

[WF74] R. A. Wagner and M. J. Fischer: *The String-to-String Correction Problem*, J. ACM, volume 21(1), (1974), pages 168–173.

[WHT$^+$07] C.-P. Wei, P. Hu, C.-H. Tai, C.-N. Huang and C.-S. Yang: *Managing Word Mismatch Problems in Information Retrieval: A Topic-Based Query Expansion Approach*, J. Manage. Inf. Syst., volume 24, (2007), pages 269–295, URL `http://portal.acm.org/citation.cfm?id=1481765.1481775`.

[Wis93] M. J. Wise: *String Similarity via Greedy String Tiling and Running Karp-Rabin Matching.*

[WS97] Y. Wilks and M. Stevenson: *Sense Tagging: Semantic Tagging with a Lexicon*, *In Proceedings of the SIGLEX Workshop*, pages 47–51.

[XC96] J. Xu and W. B. Croft: *Query Expansion Using Local and Global Document Analysis*, *In Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 4–11.

[Zip49] G. K. Zipf: *Human Behavior and the Principle of Least Effort*, Addison-Wesley (Reading MA), 1949.