SCALABILITY OF SEMANTIC ANALYSIS IN NATURAL LANGUAGE PROCESSING

RNDr. Radim Řehůřek

Ph.D. Thesis

Faculty of Informatics
Masaryk University

May, 2011

Thesis Supervisor
Assoc. Prof. PhDr. Karel Pala, CSc.

Thesis Advisor
Assoc. Prof. RNDr. Petr Sojka, Ph.D.

ABSTRACT

Data mining applications that work over input of very large scale (*web-scale* problems) pose challenges that are new and exciting both academically and commercially. Any web-scale algorithm must be robust (dealing gracefully with the inevitable data noise), scalable (capable of efficiently processing large input) and reasonably automated (as human intervention is very costly and often impossible on such scales).

This thesis consists of two parts. In the first part, I explore scalability of methods that derive a semantic representation of plain text documents. The focus will be entirely on *unsupervised* techniques, that is, on methods that do not make use of manually annotated resources or human input. I develop and present scalable algorithms for Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA), two general-purpose statistical methods for semantic analysis that serve as building blocks for more concrete, applied algorithms. Scalability is achieved by building the semantic models in a *constant amount of memory* and *distributing the computation* over a cluster of autonomous computers, connected by a high-latency network. In addition, the novel LSA training algorithm operates in a single pass over the training data, allowing continuous online training over infinite-sized training streams.

The second part of the thesis deals with possible applications of these general semantic algorithms. I present my research in the field of Information Retrieval (IR), including work on topic segmentation of plain-text documents, on document-document similarities ("semantic browsing") in digital libraries and on language segmentation of documents written in multiple languages.

ACKNOWLEDGMENTS

I would like to thank my supervisor Prof. Karel Pala for his support and gentle nudging when progress was slow. To Prof. Petr Sojka for helping me with typesetting this document and discussing practical aspects of my work on digital libraries. To my employer, the Seznam.cz company, for putting practical research into perspective. To my late colleague Milan Kolkus whose untimely departure put all research into perspective.

I would also like to acknowledge the help and contributions from Dr. Laura Dietz (who kindly sent me her LaTeX macros for Bayesian graphs), to Dr. Mark Tygert for his valuable remarks on my work on low-rank matrix decompositions, to Prof. Popelínský for his early review comments and to all the anonymous reviewers who helped improve my various papers.

Special thanks to users of my open-source *gensim* software package that accompanies this thesis—it's always more fun to produce for an audience.

And of course thanks to Oana for her saintly patience and providing the ultimate motivation for my work.

ACRONYMS

ANN	Artificial Neural Network
CSC	Compressed Sparse Column
IR	Information Retrieval
LDA	Latent Dirichlet Allocation
LSA	Latent Semantic Analysis
LSI	Latent Semantic Indexing
MCMC	Markov-Chain Monte Carlo
ML	Machine Learning
NLP	Natural Language Processing
NMF	Non-negative Matrix Factorization
PCA	Principal Component Analysis
SVD	Singular Value Decomposition
SVM	Support Vector Machines
VSM	Vector Space Model

NOTATION

Throughout this thesis, bold uppercase letters will denote matrices, $\mathbf{A}$, and lowercase letters vectors and scalars, x, y, v_1, vec. The letters m and n will represent the number of rows and columns of a matrix $\mathbf{A}$, respectively. In the Vector Space Model, observations will correspond to matrix columns (so that n equals the total number of observations) and features rows (with m the total number of features). In parts that discuss dimensionality reduction, I will use the shorthand $\mathbf{A}_{m \times n}$ for a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, that is, for a real-valued matrix with m rows (features) and n columns (observations, documents). The following table summarizes some of the notation used in the thesis.

$\lVert x \rVert$	Euclidean (L_2) norm of vector x, $\lVert x \rVert = \sqrt{\sum_i x_i^2}$.
$a_{i,j}$	Element of matrix $\mathbf{A}$ at i-th row and j-th column.
$\mathbf{A}^T$	Transposition of matrix $\mathbf{A}$.
$\lVert \mathbf{A} \rVert$, $\lVert \mathbf{A} \rVert_F$	Frobenius norm of a matrix, $\lVert \mathbf{A} \rVert = \sqrt{\sum_{i,j} a_{i,j}^2}$.
$\lVert \mathbf{A} \rVert_2$	Spectral norm of a matrix, $\lVert \mathbf{A} \rVert_2 = \max_{\lVert x \rVert = 1} \lvert \mathbf{A}x \rvert$.
$\left[\mathbf{A}_1, \mathbf{A}_2\right]$	Column-wise block matrix concatenation.
$\begin{bmatrix}\mathbf{A}_1 \\ \mathbf{A}_2\end{bmatrix}$	Row-wise block matrix concatenation.
$\overline{x}$	Mean of vector x.
$A_{i,*}, A_{*,j}$	The i-th row (j-th column) vector of matrix $\mathbf{A}$.

CONTENTS

1 INTRODUCTION

> Man is the best computer we can put aboard a spacecraft,
> and the only one that can be mass produced with unskilled labor.
> (Wernher von Braun)

Although the advances in the field of Artificial Intelligence (AI) over the past 60 years are considerable[1], the ultimate goal of making machines understand human language is still far off. Historically, AI research concentrated on tasks that were considered intellectually hard and therefore impressive for humans: rapid calculations, verbatim memorization, playing chess at grandmaster level, automated theorem proving... Interestingly, these are all relatively simple to master for computers. Early triumphs in these areas have inspired optimism that didn't translate to other fields, such as object recognition or understanding natural language, which in turn led to repeated periods pessimism (known as the "AI winters").

1.1 AI IN SCIENCE

The fundamental nature of the field of Artificial Intelligence is manifested in and mirrored by the multitude of approaches to understanding the role of science and engineering in producing intelligent machines.

One branch, the origins of which can be traced back to the philosophical works of Kant and Heidegger, is called the *embodied cognition hypothesis*: cognition can only come from machines equipped with sensory and

embodied cognition hypothesis

[1] The term of Artificial Intelligence in connection with modern computers is said to have originated with McCarthy, Minsky, Rochester and Shannon at the Dartmouth Conference in 1956.

motor skills (Brooks, 1990; Lakoff and Johnson, 1999). This view is particularly popular in robotics and neurobiology and is in direct contrast to the high-level "symbolic AI" approaches. Hubert Dreyfus made a strong claim as early as 1960s that human intelligence depends deeply on unconscious instincts and reflexes and that these unconscious skills cannot be captured by formal rules (Dreyfus, 1967).

symbolic AI

TIL

Another approach, popular in linguistics, psychology and philosophy, is to study AI (and communication in particular) within the context of logical propositions. This line of thinking also has a long tradition, with notable proponents of Leibniz or Russell. Propositions can be viewed as high-level mental entities that correspond to ("denote") external events and states of affairs (Cruse, 2000). In themselves propositions are meaningless, but may assume a true/false truth value with respect to some actual events or states of affairs. Reasoning (logical inference) happens over symbolic representations and a corresponding knowledge base. Extensions to this model include linguistic modality, higher-order logics, propositional attitudes (Cresswell, 1985) and so on. A particularly well-grounded theory is that of Transparent Intensional Logic (TIL), pioneered in (Tichý, 1969, 1980). TIL builds on the distinction between meaning, reference and denotation and allows principled inference over propositions generalized into logical constructs of arbitrary order. See also Materna et al. (1989); Materna (2004) for ongoing work in this fascinating theory.

connectionism

These two conflicting views were somewhat reconciled in 1990s with the emergence[2] of *sub-symbolic* computing. The difficulties were recognized to stem from uncertainty inherent in real-world tasks and consequently research focus has gradually shifted from pure logic to fuzzy reasoning and statistics. Many of the recent successes, from machine translation to speech recognition to web search (information retrieval) to personalized advertising, rely on the ability to automatically learn from massive collections of past examples. In the sub-symbolic (or connectionist) approach, emphasis is still put on solid, mathematical theories, but a system's behaviour and the "rules" producing it are no

[2] Or, rather, re-emergence: many of the sub-symbolic concepts have been acknowledged to be related to earlier work, from Wittgenstein's view on the connection between language use and meaning, to the above mentioned Dreyfus' critique of AI, Rosenblatt's early perceptron algorithm, the noisy channel model and so on.

longer necessarily open to high-level human introspection and explanation. Rule specification occurs at a lower level, and semantic properties are hoped to emerge (that is, manifest themselves in the behaviour of the machine) without having been explicitly programmed in (Chalmers, 1992). Examples include the Artificial Neural Networks (ANNs), statistical modelling and other sub-disciplines of machine learning. Thanks to the solid mathematical foundation and use of mathematics as the underlying language, this approach has allowed collaboration across many previously disconnected fields of computer science, such as image processing, speech processing or natural language processing, as well as economics, biology and mathematics itself (Russell and Norvig, 2009).

1.2 MACHINE LEARNING

The subfield of AI which focuses on constructing computer programs that learn from experience (with respect to some useful but limited class of tasks and a performance measure (Mitchell, 1997)) is called Machine Learning (ML). Its goal is to produce techniques that discover patterns and regularities in semi-structured or unstructured data. The many subbranches of ML include classification, clustering, probabilistic reasoning or decision theory.

Despite its successful applications, current state-of-the-art ML techniques contain a lot of what could be called "virtue by necessity"—the mathematical models that drive them behind the scene are chosen for their computational tractability, and typically contain simplified (sometimes blatantly wrong) assumptions; they are often hard to interpret for human experts and their output rarely directly translates to new insights or knowledge about the problem[3]. Their success is measured by their utility, not elegance in the psychological or linguistic sense. With a bit of creative license, this could also be called the "dumb but useful" paradigm: even the most advanced and state-of-the-art NLP techniques, like the variants of the Latent Dirichlet Allocation described in Chapter 4, are marked by naïve assumptions—the underlying model is plainly inadequate in modelling human language in its richness (Pinker,

[3] After all, as Richard Hamming famously quipped, "The purpose of computing is insight, not numbers".

2000). More than anything else, the word “dumb” here reflects our human intuition of elegance and our insight into how we ourselves form utterances, how we experience language internally. On the other hand, from a mathematical point of view, these methods are often beautiful and quite clever (anything but dumb!), as is to be expected after many decades of intense research.

unsupervised learning

The work described in this thesis falls fully under this “dumb but useful” paradigm. It deals with a particular subfield of Machine Learning called *unsupervised learning*, where data comes in such poor quality or in such quantities that human “supervision”, as well as any sort of human inspection during its processing, is infeasible. This initial chapter introduces the field and lays background for the subsequent part, which will describe particular methods of semantic inference in unsupervised environments. Part i discusses the scalability challenge of applying these methods to vast, modern datasets. Part ii then applies these general semantic methods to specific, real-world problems. It presents some associated semantic algorithms, like topic and language segmentation. The purpose of these is to make sure we compare “apples to apples” when dealing with semantic similarity in heterogeneous text corpora.

1.3 STATISTICAL SEMANTICS

corpus

In the following, *corpus* will denote a body of (digital) *text documents*. A document is simply a chunk of text and does not necessarily correspond to the entirety of a manuscript as it was originally published. In Information Retrieval, documents are often individual paragraphs, passages, sentences, phrases or even just sequences of characters. The ideal granularity of what constitutes a “document” is dependent on the intended application of the corpus. In these generalized settings, documents are also sometimes called *contexts* or *chunks*. It can be advantageous to view and store documents as “semantically coherent blocks of text”, where each chunk deals with a single idea or topic; will return to the particular question of semantic chunking in Chapter 5.

(Firth, 1957) postulated that "You shall know a word by the company it keeps". It has long been observed that words that occur in similar contexts are semantically related (Harris, 1954; Furnas et al., 1984); see

(Turney and Pantel, 2010) for a more complete overview. This is often phrased more generally as the *statistical semantics hypothesis*:

statistical semantics hypothesis

> *"Statistical patterns of human word usage can be used to figure out what people mean."*

Clearly, the wording here is rather broad and the hypothesis is more philosophical than practical in nature. It is most closely related to *lexical semantics* within the traditional NLP semantics hierarchy (Allen, 1995), or the *word-word* relations of (Griffiths et al., 2007). For an interesting take on word relations from the opposite direction, concentrating on statistical out-liers ("exploitations") versus salient patterns ("norms"), see (Hanks, 2011). Nevertheless, the statistical semantics hypothesis has served as an important stepping stone towards more specific, computation-oriented instantiations, such as the distance-based view of the *bag-of-words* hypothesis.

1.4 bag-of-words

In mathematics, a *bag*, also called a *multiset*, is a set with duplicates allowed. A document can be represented by the bag of its constituent tokens, so that for example the utterance *"to be or not to be"* is represented by the multiset $\{to, be, or, not, to, be\}$. Order of the elements in (multi)sets is arbitrary, so we may equivalently represent this document by the vector $\langle 2, 1, 1, 2\rangle$, given the understanding that the first element corresponds to the frequency of the word *"be"*, second to that of *"not"*, third of *"or"* and fourth of *"to"*.

In Information Retrieval, the *bag-of-words hypothesis* stipulates that such word frequency vectors can be used to assess semantic relevance of documents. In other words, it states that the frequencies of individual words are sufficiently indicative of semantic association between two documents (or a document and a query).

bag-of-words hypothesis

Needless to say, the bag-of-words hypothesis is painfully naïve from the linguistic perspective. It ignores word order, as well as any syntactic structure, which necessarily incurs a serious loss of information. Each observation is represented by a vector, i.e., by a single point in space.

Similarity then translates to standard vector similarity measures, some of which will be discussed in the following chapter.

On the positive side, the transformation from the surface text into a bag-of-words vector is computationally straightforward and efficient. Switching to the world of vectors and matrices also allows us to utilize powerful techniques and algorithms from the field of linear algebra. Perhaps the most convincing argument in this respect is the vast body of literature and successful applications based on this approach (Turney and Pantel, 2010).

1.5 THE TERM-DOCUMENT MATRIX

With a collection of many documents, their corresponding vectors can be stacked into a single matrix. By convention, document vectors form the columns, while the vector elements (called *features*) form the matrix rows. With n documents and m features, we talk of an $m \times n$ matrix $\mathbf{A}$. In more complex schemes, the integer event frequencies from bag-of-words are commonly reweighted according to their importance (reflecting the intuition that some words are more semantically important than others), and the resulting vector can be normalized to abstract from document length, so that $\mathbf{A} \in \mathbb{R}^{m \times n}$. Where clarity is not compromised, we will use the shorthand notation $\mathbf{A}^{m \times n}$ to say the same thing.

features

Under the bag-of-words hypothesis, each vector dimension corresponds to the frequency of a particular token; these dimensions form descriptive facets of the data and are therefore also called *features*. Each document is then interpreted as a measurement in a multidimensional feature space. Bag-of-word models use features with *quantitative* domain (integer or real numbers). Features used in other fields of Machine Learning, such as unordered nominals (human gender, nationality, colours) or ordered nominals (*"good-neutral-bad"*), are either avoided or must be first carefully converted. Another peculiarity of the bag-of-words approach is the very high dimensionality of the feature space: one for each token type, with the total number of dimensions often exceeding 100,000.

In many domains, only a handful of features are active in any given context. For example, in the *"to be or not to be"* example, only three words appear with non-zero frequency. All other words, such as *"shopping"* or *"hippopotamus"* have a frequency of zero. The vector that corresponds to this document would therefore have three non-zero elements, and typically tens or hundreds of thousands of zero elements. Naturally, algorithms that build on top of the bag-of-words paradigm must take sparsity into account if they are to be efficient. See for example (L'uboš Kardoš, 2010) for a discussion of the practical issues of sparse matrix storage and operations.

The vanilla bag-of-words model described above is seldom the representation of choice. More advanced techniques used in practice exploit more complex vocabulary patterns, just as additional vector transformations and similarity metrics capture more of (or a different type of) document semantics.

1.6 text preprocessing

For word-based contexts, there are a number of associated tasks that must happen before any semantic processing takes place.

- *Tokenization*, to split the text into individual words. This is already non-trivial for some Asian languages, including Chinese.

- *Token normalization*, which depending on the task can be as simple as case folding (discarding information about letter casing) or as complex as lemmatization (full morphology analysis in synthetic, and especially flective, languages like Czech).

- *Spelling correction*. How to deal with ambiguous spelling variations such as "Nokia N8" vs. "Nokia N-8" vs. "Nokia N 8", or "Windows 2000" vs. "Windows 2k", "don't" vs. "dont" and "state-of-the-art" vs. "state of the art" etc. Depending on the application, the desired course of action may be to use the exact form as it appears in the text, or normalize the tokens to a single canonical form (either during the tokenization phase or later at query resolution).

- *Multi-word expressions.* How to handle more complex lexical units? Examples include dates (*"12.4.2010"*), IP addresses

(*"147.251.48.1"*), ranges (*"10–55"*), abbreviations or compound tokens in general, including emoticons (*":-)"*, *"<3"*).

All of these choices interact and affect a system's performance. They are also crucial in the sense that errors at this fundamental level are very costly to correct later on. As an alternative, there have been experiments with bag-of-word models that are based on character n-grams (Cavnar and Trenkle, 1994). This thesis will not deal with text preprocessing issues further; a curious reader is referred to (Manning and Schütze, 1999) and (Feldman and Sanger, 2007) for more information.

1.7 SCALABILITY

Web-scale problems depart from the traditional sense of "efficiency" in that *FLOPS (Floating Point Operations per Second)* are no longer "the" metric of choice of performance. A different metric, called the *number of passes* over the input, is more important, as data is often stored in secondary memory (disk) or even off-site, and doesn't fit in main memory. Fetching the input data is the most costly operation, typically dwarfing any processing done afterward by the CPU. Moreover, practical algorithms cannot expect to use more than a constant amount of main memory with respect to the input size, if they are to be called truly "scalable".

This thesis describes several general purpose algorithms for inferring *latent* semantic structure of text documents. Robustness is achieved by using statistical (rather "brute-force", from a linguistic point of view) approach. For two of the most widely used algorithm, Latent Semantic Analysis and Latent Dirichlet Allocation, I present novel results that improve scalability by designing the algorithms to be streamed, online, working in a constant amount of memory w.r.t. the training corpus size, distributed and operating in a small number of passes over the input.

1.8 THESIS OUTLINE AND CONTRIBUTIONS

The highlight of this thesis is a novel algorithm for *Singular Value Decomposition (SVD)*, which operates in a single pass over the input collection (each document needs to be seen only once), is streamed (documents are accessed sequentially, no random access required) and can be distributed across a cluster of commodity computers connected by a high-latency network. Although the application of this SVD algorithm for the purpose of this thesis is *Latent Semantic Analysis*, SVD is a general linear algebra routine that is at the heart of many algorithms across much of the field of Computer Science, from Image Processing to Signal Processing, Collaborative Filtering, solving Differential Equation, Antenna Array Theory and others (Deprettere, 1989; Golub and Van Loan, 1996). In this way, the theoretical aspect of my work is also relevant outside of the domain of Natural Language Processing. I also present a streamed, distributed algorithm for *Latent Dirichlet Allocation* (LDA), a Bayesian modelling method which also works in a constant amount of memory. Advances in scalability of these general-purpose LSA and LDA algorithms form the first part of this thesis.

Part ii of the thesis describes some possible applications of these general-purpose semantic algorithms to the Information Retrieval (IR) domain. Both LSA and LDA are high-level, general purpose methods, and can be useful both in assessing term similarity (or term clustering) and document similarity (document clustering). In Chapters 5–7 I will discuss applications such as topical segmentation of unstructured text or semantic browsing in digital libraries.

While working on these problems, I developed a software package with efficient implementations of these algorithms. Written in Python and released as open-source under the OSI-approved LGPL license, it has already attracted a substantial user base and been used by students in a number of academic theses across the world. It has also served as a tool for further research, in areas as diverse as evaluation of word clustering (Turian et al., 2010), noun/classifier disambiguation of Mandarin Chinese (Magistry et al., 2010) or music recommendation systems (Fields et al., 2010).

2

THE VECTOR SPACE MODEL

Perhaps the first application based on the bag-of-words hypothesis was done by the research group around Gerard Salton at the Cornell University in the 1960s, later published in (Salton, 1971). Their *SMART (System for the Mechanical Analysis and Retrieval of Text)* Information **SMART** Retrieval system is widely acknowledged for coining the phrase *Vector Space Modelling (VSM)* for these type of systems (Salton, 1989). Since **VSM** then, Vector Space Modelling (and its many extensions) have become common place. Despite its unrealistic linguistic assumptions, it set the direction for one of the most successful approaches to text semantics so far, especially outside of the academic world (Turney and Pantel, 2010).

2.1 VSM PROPERTIES

The VS model requires that the system architect make several design choices: Which features to extract from the documents? What weights to assign them? How to compute the document similarity? Although all of them have a standard answer in NLP (discussed below) by now, the room for expert knowledge is still very large.

The first task, creating a suitable representation for the objects of interest (in our case, text documents), is by far the most critical. It is far easier to obtain good results with a poor algorithm on good data, than with a smart algorithm that is running over input of poor quality. Often there are additional features added to the bag-of-words vectors, capturing domain-specific knowledge. This is the place where expert knowledge has the greatest impact, and can make the greatest difference in a system's performance.

2.2 WEIGHTING SCHEMES

Intuitively, some tokens are more important than others. For example, the semantic contribution of the word *"dimensionality"* is usually much greater than that of *"the"*. Similarly, some frequency ranges are more telling than others—once a token has appeared in a document, its subsequent repetitions are less informative; this is called the "word burstiness" in NLP (Madsen et al., 2005; Doyle and Elkan, 2009). These intuitions have led to the introduction of *weighting schemes* to VSM processing. Weighting schemes are (mostly empirical) attempts to model the underlying relationships between the importance of individual dimensions, token frequency distribution and varying document lengths. Following the three-letter descriptor notation from (Salton, 1971) (one letter for each of the three types of factors), we have

1. *Local weighting*, which rescales weights within a single document. The four most popular local weighting schemes are
   - the value of the term frequency, a_{ij} (letter choice "n"),
   - zero/one depending on term presence/absence in the document ("b" for binary weight),
   - augmented weight ("a") which scales the frequency into $\langle 0.5, 1\rangle$ interval with $0.5 + 0.5\frac{a_{ij}}{\max_k a_{kj}}$,
   - logarithmic weight $\log(1 + a_{ij})$, denoted by the letter "l".
2. *Global weighting* which takes into account importance of the feature across the whole collection. It can produce a constant weight of 1.0 (letter "n") or inverse document frequency $\log\left(\frac{n}{|1+d_i|}\right)$ (letter "t").
3. *Normalization* sets the vector to unit Euclidean length ("c") or does nothing ("n").

In the above, a_{ij} stands for the frequency of term i in document j in the term-documents matrix $\mathbf{A}^{m\times n}$, and d_i is the number of documents in which the term i appears at least once.

These three factors combine together to make a single weighting strategy: for example, *"bnn"* stands for binary weighting without normalization and with all features having the global weight; *"nnn"* is the unmodified bag-of-words vector etc. The most widely used combination is *"ntc"*, which is more commonly known as the *TF-IDF* weighting[1].

TF-IDF

There are several other common preprocessing steps, such as stop-list removal, thresholding very rare terms (typically OCR errors and other noise), which I will not discuss further here. See for example our paper (Řehůřek and Pomikálek, 2007) or (Yang and Pedersen, 1997; Forman et al., 2003) for a discussion of the impact of the various preprocessing parameters on classification accuracy.

2.3 SIMILARITY MEASURES

Given two real-valued vector representations, we need a way to quantify their similarity. Assuming the representations are semantically meaningful to start with, this will help us with the goal of answering semantic similarity queries. Here I will restrict the discussion to a few well established similarity functions which take two vector representations as their input and produce the similarity (a single number) on output. (Rijsbergen, 1979) observed that many of the popular similarity measures produce similar results, provided the input representations are properly normalized (vectors of unit length; posteriors of unit probability).

Other similarity functions exist, for measuring the similarity of sets (the *Jaccard's coefficient*), of strings (the *edit distance*) or even trees (the *tree edit distance* (Cobena et al., 2002)). See (Turney and Pantel, 2010) for a discussion of possible modes of Vector Space Modelling, such as the document vs. document similarity, word vs. word similarity or the similarity of word relations.

2.3.1 *Minkowski Distance*

The *Minkowski distance* L_p is actually a class of distance functions parametrized by $p \in \langle 1, \infty)$. The distance between two m-dimensional

[1] The term *TF-IDF* is also sometimes used to describe the *end product* of applying the weighting, in which case it denotes the entire resulting matrix.

vectors x, y is given by

$$L_p = \|x - y\|_p = \left(\sum_{i=1}^{m} |x_i - y_i|^p \right)^{1/p}. \qquad (2.1)$$

Euclidean distance

The popular *Manhattan and Euclidean distances* form the special cases of L_1 and L_2, respectively. The limiting case when $p \to \infty$ results in $L_\infty = \max_{i=1}^{m} |x_i - y_i|$ and is called the *Chebyshev distance*.

In our vector space scenario, distance and similarity are inversely proportional, so we may compute the latter from the former, for example by taking the inverse, $^1/_{1+\|x-y\|_p}$.

2.3.2 *Cosine Similarity*

The most common similarity measure for the Vector Space Model is *cosine similarity*, which measures cosine of the angle between two vectors in the vector space,

$$\mathrm{cossim}(x, y) = \frac{x^T \cdot y}{\|x\| \|y\|}. \qquad (2.2)$$

The range of this function is $\langle -1, 1 \rangle$. Another observation to be made is that if the similarity reduces to inner product (e.g., when using the cosine measure with normalized document lengths), all pairwise similarity scores of documents represented by term-document matrix $\mathbf{A}^{m \times n}$ may be obtained by a single matrix product $\mathbf{A}^T \cdot \mathbf{A}$. The resulting $n \times n$ matrix (n being the number of documents) is symmetric and $a_{i,j}$ holds the similarity values between the documents i and j.

2.3.3 *Hellinger Distance*

In Bayesian probability modelling, the latent semantic representation of documents correspond to probability distributions over the latent variables. *Hellinger distance* measures divergence of two probability distributions, and in the discrete case takes the form

$$H(x, y) = \sqrt{\frac{1}{2} \left(\sum_{i=1}^{m} (\sqrt{x_i} - \sqrt{y_i})^2 \right)}. \qquad (2.3)$$

The Hellinger distance is connected to the *Kullback-Leibler divergence* D_{KL} (an asymmetric distance also used in probability theory, (Rao, 1995)) by $D_{KL}(x\|y) \geqslant \frac{2}{\ln 2}H^2(x, y)$.

2.4 QUERYING

Unlike traditional database systems designed for boolean queries (such as the by now standard keyword search of Web search engines), in data mining we often need to retrieve objects that are merely similar and not necessarily identical to the query object. This is an extension not specific to text documents; methods for measuring similarity for images, faces, fingerprints, time series etc. have been developed.

The query problem may be defined as either retrieving the k most similar documents (*exact query*), or retrieving k documents that are very similar but not necessarily the most similar (*approximative query*). This type of querying is sometimes referred to as the k nearest neighbours (k-NN) search. Another type of query commonly used is the *range query*, where all documents within a given distance from the query are retrieved. The *inverted file* (also called *inverted index*) technique and its numerous extensions often used both in boolean model (see e.g. (Moffat and Zobel, 1996; Zobel et al., 1992)) cannot be effectively utilized here, because VSM vectors are real-valued.

The first observation is that modern databases are potentially large, consisting of hundreds of thousands of documents or more. The naïve approach of linearly scanning through the entire database for each query is prohibitive due to the I/O costs. Although this problem can he alleviated by replication—splitting or distribution of the database over more machines—much research has gone into more principled techniques. In general terms, these techniques usually try to exploit structure of the data and use partitioning to prune searches over the database. These techniques fall into the categories of *data partitioning* and *space partitioning* methods. Both have enjoyed a lot of attention in the past decade, and there has been a steady flood of various **-Tree* proposals.

Examples of popular *data partitioning methods* include

R-TREES (Guttman, 1984) as a generalization of B-Trees into higher dimensions. It turns out that the partitioned regions overlap more and more with increasing dimensionality, reducing efficiency of the indexing structure.

R*-TREES (Beckmann et al., 1990) are optimized versions of R-trees which try to minimize the overlap by grouping nodes using a more complex algorithm.

X-TREES (Berchtold et al., 2002) which also try to overcome the problem with R-Tree based techniques. Their approach introduces supernodes and different splitting rules to avoid high overlaps.

M-TREES (Ciaccia et al., 1997) which abstract from the Vector Space by allowing objects to come from a general metric space (i.e., requiring only a black-box distance function).

Among the *space partitioning methods* we count *grid-files* (Nievergelt et al., 1984), *quad-trees* (Finkel and Bentley, 1974) or *KD-trees* (Bentley, 1975), which divide the data space along predetermined lines regardless of data clusters. For both types of methods, the idea is to build a multidimenional index structure that will allow only a subset of all the documents to be traversed in order to answer any query.

curse of dimensionality

It has however been noted that all of these methods suffer from the so called *curse of dimensionality* (Weber et al., 1998). The effectiveness of many of these indexing structures is highly data-dependent and, in general, difficult to predict. Often a simple linear scan of all the items in the database is cheaper than using an index-based search in high dimensions. While these methods perform generally very well in low dimensional spaces, the performance drops as the number of dimensions increases. This result has been reported for R*-Trees and X-Trees (Berchtold et al., 1997, 2002) and also for the SR-Tree (Katayama and Satoh, 1997). Unfortunately this is no coincidence, but rather a fundamental property of high dimensional spaces. According to (Weber and Blott, 1997), under assumptions of data uniformity and independence of data dimensions, and data being partitioned into blocks, all such indexing structures eventually deteriorate into linear scanning as the dimensionality increases. Building and maintaining complex indexes is computationally demanding. Moreover, from a practical point of view, the real query response time is bogged down by random disk seeks during tree updates and traversal, something that the simple linear scan trivially avoids. As shown by authors of (Weber and Blott, 1997), the dimensionality of where the performance break occurs is not even very high. On their image database, linear scan starts outperforming the tree indexing structures at around the dimensionality of 10, measured by wall clock times (that is, including I/O costs).

Two solutions to the curse of dimensionality have been proposed: a) making the search distributed (Batko et al., 2006), and b) resigning on the issue and concentrating on making the linear scan as fast as possible. In the latter, a promising line of optimization is the VA-File (Weber and Blott, 1997). VA-File offers constant memory improvement while not sacrificing any accuracy. Unlike other data partitioning methods, its relative performance even *increases* with increasing dimensionality. The authors report that for a high number of dimensions (in the order of hundreds), VA-File outperforms all partitioning based methods by an order of magnitude or more. The algorithm lends itself naturally to parallelization (Weber et al., 2000). (Tesic et al., 2004) explores adaptive VA-File for relevance feedback, based on optimizing the search through use of information from previous feedback iterations.

2.5 extensions

The Vector Space Model described in this chapter falls into the family of unsupervised learning methods. The bag-of-words model is its most trivial example, in that no hidden (latent) structure is assumed to exist in the data. "Learning" of the bag-of-words representation of a corpus is straightforward: the corpus is described directly by the observed bag-of-words vectors.

This is not ideal, as it

- assumes complete independence among the terms. The appearance of the term "Barack" in a document is supposed to tell us nothing about the word "Obama" appearing in the same document, which is clearly not true.

- more generally, it makes no assumptions about the generative process that created the text, and as such cannot deal with polysemy, synonymy and other common language phenomena
- wastes resources. Human language is a communication protocol with a high degree of redundancy built in. Describing the content in a more succinct form results in more efficient processing due to the representation's smaller footprint.
- does not offer insight into the essential corpus structure. Connected to the previous point, a more abstract representation promises to describe the content of a document in a more fundamental way, so that a similarity computed over these abstract representations better matches our human intuition of "semantic resemblance".

The field of feature subset selection (see for example Yang and Pedersen (1997); Řehůřek and Pomikálek (2007)) attempts to address the third point, by eliminating features (i.e., words) that add the least information to the full set. The methods used there differ in how they quantify the notion of "little information". This is a way of reducing the computational complexity (by reducing the number of vector dimensions), but it does not address the issue of explaining the more abstract, semantic structure of text documents.

2.5.1 *Latent Semantic Analysis*

One of the earliest attempts to solve the aforementioned issues in a principled manner is the *Latent Semantic Analysis (LSA)*, also sometimes called *Latent Semantic Indexing* in the Information Retrieval community (Deerwester et al., 1990). Its main idea is to exploit term co-occurrence to derive a set of *latent concepts*; words that frequently occur together are assumed be more semantically associated (Kontostathis and Pottenger, 2006). This is in accordance with the statistical semantic hypothesis mentioned earlier (Section 1.3), as it directly models the relationship between words based on the contexts that they share. LSA analyzes term co-occurrence of higher orders, so that it is able to incorporate the relationship of words A, B which only co-occur in documents through a difference word, C, and never appear in the same document together directly. In the same way, LSA is able to de-emphasize a connection between two co-occurring terms that frequently co-occur with many other terms at the same time.

LSA has also become popular in the field of Cognitive Science, where it is believed LSA plausibly models some aspects of human cognition (Foltz, 1996; Dumais and Landauer, 1997; Landauer, 2007). It has been used as an automated essay grader (Foltz et al., 1998), in metaphor comprehension (Kintsch, 2000) or for intelligent tutoring systems (Graesser et al., 2001). In one of the most widely cited studies, LSA equalled the human performance of non-English speaking U.S. college applicants, on a standard multiple-choice synonym test from the *Test of English as a Foreign Language (TOEFL)*. The scores achieved by LSA were adequate for admission to many U.S. universities (Dumais and Landauer, 1997).

Numerically, each of the LSA concept can be viewed as a function that accepts an m-dimensional vector on input and calculates a linear combination (weighted sum) of its coordinates, a scalar. With k LSA concepts, each input vector (i.e., document) is represented by a vector in $\mathbb{R}^k$. This transformation from the *TF-IDF* vector space into one represented by the k latent concepts is realized through multiplication by a suitable matrix, which places LSA into the category of *linear models*. Finding the suitable projection matrix is done via the *Singular Value Decomposition (SVD)* algorithm of linear algebra, described below, and is computationally the most expensive part of LSA.

Use of the SVD algorithm makes LSA really just another member of the broad family of applications that make use of SVD's robust and mathematically well-founded approximation capabilities[2]. In this way, although I will discuss the results in Chapter 3 from the perspective and terminology of LSA and Natural Language Processing, the results are in fact applicable to a wide range of problems and domains across much of the field of Computer Science. For a broader perspective on SVD, LSA and topical models, see for example (Golub and Van Loan, 1996; Dumais and Landauer, 1997; Griffiths et al., 2007).

[2] Another member of that family is the *discrete Karhunen–Loève Transform*, from Image Processing; or Signal Processing, where SVD is commonly used to separate signal from noise. SVD is also used in solving shift-invariant differential equations, in Geophysics, in Antenna Array Processing, ...

Singular Value Decomposition

The SVD algorithm finds matrices **U**, **S** and **V** such that

$$\mathbf{A}_{m\times n} = \mathbf{U}_{m\times r}\mathbf{S}_{r\times r}\mathbf{V}^{T}_{r\times n}, \tag{2.4}$$

where $r = \text{rank}(\mathbf{A}) \leqslant \min(m, n)$, **U** and **V** are column-orthonormal and **S** a diagonal matrix with the main diagonal entries sorted in non-increasing order. This decomposition always exists and the r columns of **U** and **V** can be viewed as bases of new linear subspaces, called the *left* and *right* subspace, respectively.

In LSA, reducing the representation footprint and at the same time the source of semantic abstraction is the result of clipping the **U**, **S**, **V** matrices to $\widehat{\mathbf{U}}_{m\times k}$, $\widehat{\mathbf{S}}_{k\times k}$, $\widehat{\mathbf{V}}_{k\times n}$ of a lower rank k, $k \leqslant r$, which amounts to using only the first k columns of **U**, **V** and the upper left $k \times k$ part of **S**.

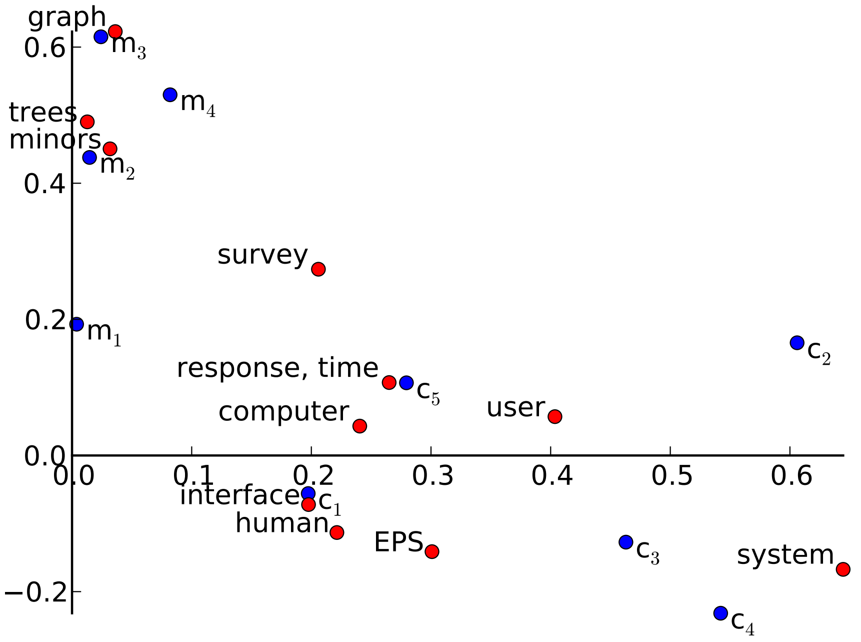
The power and popularity of SVD lies in the fact that truncating the spectrum in this way gives us a provably best rank-k approximation of the original matrix, $\widehat{\mathbf{A}}_{m\times n} = \widehat{\mathbf{U}}_{m\times k}\widehat{\mathbf{S}}_{k\times k}\widehat{\mathbf{V}}^{T}_{k\times n}$, in the sense that $\widehat{\mathbf{A}} = \underset{\text{rank}(\mathbf{X})\leqslant k}{\operatorname{argmin}} \|\mathbf{A} - \mathbf{X}\|_F$. This means that among all lossy (linear) transformations of rank at most k, the one that is defined by the truncated SVD manages to retain the greatest amount of variance of the original data (Leach, 1995).

In the context of LSA, the truncated left and right subspaces $\widehat{\mathbf{U}}$, $\widehat{\mathbf{V}}$ are called the *term* and *document* subspaces, as the m rows of $\widehat{\mathbf{U}}$ correspond to latent rank-k representations of the original m terms, and likewise the n rows of $\widehat{\mathbf{V}}$ to the latent rank-k representations of the original documents. The k columns of $\widehat{\mathbf{U}}$ are called *topics*, or *(latent) concepts*, and, as mentioned previously, are mutually orthogonal and form basis for the term subspace. An example of terms and documents of the corpus from Appendix A projected into the same 2-dimensional ($k = 2$) latent space can be seen in Figure 2.1.

LSA critique

Despite its successes, LSA also suffers from several shortcomings:

Figure 2.1.: A sample corpus from Appendix A when projected into a 2-dimensional LSA space. Both terms and documents are projected into the same space. In LSA, the document vectors are simply sums of their constituent term vectors, scaled by the singular values. Similarity can be assessed by (the cosine of) the angle between two documents (or terms).



1. Choosing the optimal value for the k parameter (the latent space dimensionality) is not obvious. In IR practice, the value is typically set to several hundred, but depends on the application as well as the structure of the input corpus (Řehůřek, 2007a; Bradford, 2008).

2. Topics are not interpretable. In other words, by looking at the m-dimensional vector of a particular topic $t_i, i \leqslant k$, it can be hard to assign a human label to the theme connecting the highest scoring terms (Griffiths and Steyvers, 2004). See Table 3.5 in Chapter 3 (page 61) for an example of some Wikipedia topics obtained by LSA.

3. Finding the **U**, **S**, **V** decomposition is costly. Although only a partial decomposition is needed (the k greatest singular triplets), as opposed to the full spectrum of rank r, for large data collections even this is challenging. Common remedies include using SVD algorithms that make use of **A**'s sparsity[3] or parallelizing the computation. Another field of research has been directed at optimizing the cost of updates to the input matrix **A**. Rather than recomputing SVD from scratch, *incremental updating* schemes have been proposed, which allow one to alter an existing decomposition $\mathbf{U}, \mathbf{S}, \mathbf{V}$ when new observations (or features) arrive (Levy and Lindenbaum, 2000; Brand, 2006; Řehůřek, 2010). This is especially relevant for LSA, because in the IR domain, collections grow constantly and some sort of incremental updating scheme is necessary.

incremental SVD updates

2.5.2 *Principal Component Analysis*

Principal Component Analysis (PCA) is another method for exploratory data analysis. It is closely related to the SVD from Latent Semantic Analysis, in that it also finds the linear projection that minimizes the average projection cost (mean squared distance between original data points and their projection). The only difference is that it works over a *centered* term-document matrix $\widehat{\mathbf{A}}_{m\times n}$, in which the feature mean has been subtracted from each dimension, $\widehat{a}_{ij} = a_{ij} - \overline{\mathbf{A}_{i*}}$.

PCA projects data into the (truncated) subspace spanned by the eigenvectors $\mathbf{Q}_{m\times k}$ of the sample covariance matrix $\mathbf{C} \equiv \frac{1}{n-1}\widehat{\mathbf{A}}\widehat{\mathbf{A}}^T$. Since **C** is a Hermitian, positive-semidefinite matrix, according to the spectral theorem a decomposition $\mathbf{C} = \mathbf{Q}\mathbf{\Sigma}\mathbf{Q}^T$ always exists with **Q** column-orthonormal and $\mathbf{\Sigma}$ real non-negative and diagonal. PCA thus uses *eigenvalue decomposition (EVD)* of the mean-centered covariance matrix, where LSA uses singular value decomposition of the original term-document matrix. When $\widehat{\mathbf{A}} = \mathbf{USV}^T$, we have $\widehat{\mathbf{A}}\widehat{\mathbf{A}}^T = \mathbf{U}(\mathbf{S}^T\mathbf{S})\mathbf{U}^T$ (**V** is column-orthonormal so $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ vanishes). We see that $\mathbf{Q} = \mathbf{U}$ and $\mathbf{S}^2 = \mathbf{\Sigma}$ and the connection between PCA and LSA (i.e., between

SVD vs. EVD

[3] Ideally by accessing **A** only through its matrix-vector products $\lambda x : \mathbf{A}x$, so that the SVD algorithm can act as a "black-box" and work over input matrices of any structure, provided an appropriate matrix-vector multiplication routine is supplied by the user.

eigen decomposition and singular decomposition) becomes clear: $\mathbf{U}$ is the EVD of $\widehat{\mathbf{A}}\widehat{\mathbf{A}}^{\mathsf{T}}$, $\mathbf{V}$ is the EVD of $\widehat{\mathbf{A}}^{\mathsf{T}}\widehat{\mathbf{A}}$ and $\mathbf{S} = \sqrt{\boldsymbol{\Sigma}}$ of either. Where PCA computes only the term space projection $\mathbf{U}$ and scaling $\mathbf{S}^2$, LSA's singular value decomposition gives both $\mathbf{U}$, $\mathbf{S}$ as well as the document space projection $\mathbf{V}$.

The $\widehat{\mathbf{A}}\widehat{\mathbf{A}}^{\mathsf{T}}_{m \times m}$ sample covariance matrix is very large, and it is therefore desirable to compute the decomposition directly over $\widehat{\mathbf{A}}$, rather than going through the intermediate step of forming $\widehat{\mathbf{A}}\widehat{\mathbf{A}}^{\mathsf{T}}$ explicitly. As this decomposition is typically accomplished by SVD for accuracy reasons anyway (Golub and Van Loan, 1996), we see that the only real difference between PCA and LSA is using $\widehat{\mathbf{A}}$ in place of $\mathbf{A}$, that is, the data centering. Subtracting the mean is a common practice in statistical analysis (Miranda et al., 2008), where it amounts to removing a single degree of freedom from the projection. With $k - 1$ additional degrees of freedom left, the practical impact of performing centering is negligible (Levy and Lindenbaum, 2000). More importantly, the centered $\widehat{\mathbf{A}}$ matrix is necessarily dense, even where the original $\mathbf{A}$ is sparse. This is extremely undesirable, as the matrix consumes much more memory and subsequent processing is slower.

As mentioned earlier, spectral decompositions like SVD and EVD have a number applications:

- solving over-specified or under-specified systems of linear equations in physics,
- text processing in IR (Latent Semantic Analysis) discussed above,
- Google's PageRank; finding the principal eigenvector of a random walk matrix (Page et al., 1998),
- collaborative filtering and recommendation systems (Koren et al., 2009),
- stylometric identification in authorship attribution (Abbasi and Chen, 2006),
- image compression and computer vision (Zhang et al., 2002),

- visualization of complex data in 2D[4],

and many others. (Roweis and Ghahramani, 1999) give an excellent overview of the connection between various linear models. They place PCA, Hidden Markov Models (HMMs), Factor Analysis, Kalman filter models (linear dynamical systems) and others in a single framework of generative modelling. Within this framework, the methods differ only in they way they are parametrized, allowing a direct comparison of their inference schemes and highlighting model similarities (such as the correspondence between the number of clusters in mixture models, number of factors in factorization models or number of states in Hidden Markov Models).

2.5.3 *Non-Negative Matrix Factorization*

Like LSA, *Non-Negative Matrix Factorization (NMF)* is a linear VSM that attempts to identify bases of a space that can best explain variation in the data (Lee and Seung, 2001). NMF decomposes the (non-negative) input matrix **A** into two non-negative factors:

$$\mathbf{A}_{m\times n} = \mathbf{W}_{m\times k}\mathbf{H}_{k\times n}. \tag{2.5}$$

This is in contrast to LSA, where the bases produced by SVD are a combination of both positive and negative entries (words "positively" and "negatively" correlated with a topic).

Iterative algorithms exist for finding **W**, **H** that minimize $\|\mathbf{A} - \mathbf{W}\mathbf{H}\|$, where the norm is either the Euclidean distance or the Kullback-Leibler divergence (Lee and Seung, 2001). These algorithms are approximative and may result in local minima (unlike SVD, where the global optimum is always guaranteed, barring numerical precision issues with ill conditioned matrices).

As with LSA, the columns of **W** are interpreted as "latent (hidden) concepts", but generally offer better interpretability thanks to the absence of negative entries and complex term cancellations. NMF was originally applied to the domain of image segmentation (Lee and Seung, 2001), but

[4] See also *Self-Organizing Maps (SOM)* for an alternative technique in the data visualization domain (Kohonen, 1983).

like other VSM techniques, its range of applicability also includes text processing (Berry and Browne, 2005), recommendation systems[5] (Koren et al., 2009), speech and music analysis (Wilson et al., 2008; Févotte et al., 2009), to cluster Scotch whiskies (Young et al., 2006) and so on.

2.5.4 *Probabilistic Latent Semantic Analysis*

Non-negative Matrix Factorization is closely related to another algorithm for unsupervised text analysis: the *Probabilistic Latent Semantic Analysis (PLSA)*. PLSA was introduced in (Hofmann, 1999) and uses a mixture model of k components to explain the data,

$$P(w_i, d_j) = \sum_{c=1}^{k} P(w_i, d_j | z_c) = \sum_{c=1}^{k} P(z_c) P(d_j | z_c) P(w_i | z_c), \tag{2.6}$$

where $z \in \mathbf{Z} = \{z_1, \dots, z_k\}$ is the latent (unobserved) variable corresponding to topics, d are the documents and w features.

Parameters of this joint distribution are estimated by maximizing the likelihood of the observed bag-of-words corpus $\mathbf{A}$, through an EM algorithm (Hofmann, 1999). As with NMF, using an iterative algorithm means that extra attention must be paid to the (speed of) convergence issues, as well as the risk of finding only a local maximum.

The relationship between PLSA and NMF stems from interpreting the $P(w_i, d_j)$ joint probability as an (obviously non-negative) $m \times n$ matrix, to be factorized into two matrices $\mathbf{W}'_{m \times k} \mathbf{H}'_{k \times n}$, $\mathbf{W}' = P(w_i | z_c) P(z_c)$ and $\mathbf{H}' = P(d_j | z_c)$. PLSA therefore corresponds to factorizing $\left[\mathbf{P}(w_i, d_j)\right]$ into two non-negative factors. It has been shown that under the Kullback-Leibler loss function and after suitable rescaling of $\mathbf{A}$, any (local) maximum likelihood solution of PLSA is equivalent to an NMF solution and vice versa (Gaussier and Goutte, 2005; Ding et al., 2008).

[5] A powerful impetus for NMF research and matrix decomposition algorithms in general was the Netflix prize competition, 2006–2009.

2.5.5 *Latent Dirichlet Allocation*

Latent Dirichlet Allocation (LDA) is another instance from a family of *mixed membership* latent models, that is, models that model each observation as a manifestation of multiple latent (hidden) semantic topics. It was first proposed in (Blei et al., 2003) and has been enjoying a lot of research focus since, mainly thanks to its solid theoretical background. LDA is a fully generative Bayesian model[6], allowing principled extensions and inference analyses.

LDA has been used in evolution of topics over time (Griffiths and Steyvers, 2004), author-document models (Rosen-Zvi et al., 2004), image processing (Fei-Fei and Perona, 2005), syntax-semantic models (Griffiths et al., 2005), image-caption models (Blei and Jordan, 2003), anomaly detection in video streams (Varadarajan et al., 2010) or network flow analysis (Airoldi et al., 2007). In Chapter 4, I will be dealing with LDA scalability, so I will describe its background in more detail here.

Bayesian inference

Consider the bag-of-words corpus of n (interchangeable) documents $\mathbf{A} = \{d_i\}_{i=1}^{n}$ as a sequence of i.i.d. realizations of some multinomial random variable, parametrized by a set of parameters $\lambda \in \Lambda$. In this formulation, to train the model means to infer λ from the observed data in $\mathbf{A}$. Bayesian modelling extends the point estimates of *Maximum Likelihood (MLE)* and *Maximum Aposteriori Estimate (MAP)* and explicitly models the whole posterior distribution. The benefits of Bayesian statistics are well-known and include better confidence evaluation (by examining the variance of the posterior) or using the posterior's expectation (as opposed to likelihood maximum in MLE or mode of the posterior in MAP) for data statistics (Gelman, 2004). The general formula follows from Bayes' rule,

$$P(\lambda|\mathbf{A}) = \frac{P(\mathbf{A}|\lambda)P(\lambda)}{\int_{\Lambda} P(\mathbf{A}|\lambda)P(\lambda)d\lambda} \triangleq \frac{\textit{likelihood} \cdot \textit{prior}}{\textit{normalization}}. \tag{2.7}$$

[6] In contrast to PLSA, where the documents are directly coupled with the model and the model cannot be used for topic inference on documents outside of the training corpus.

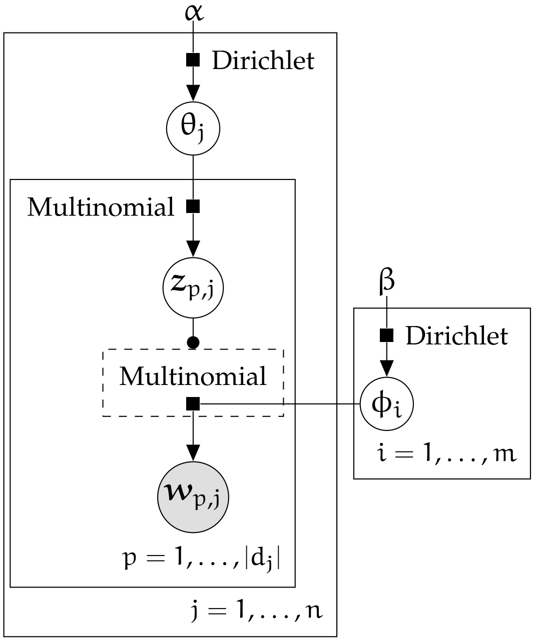


Figure 2.2.: Generative process of (smoothed) LDA as directed factor graph. In this notation, circled nodes correspond to random variables (observed variables are shaded), directed edges to dependency between variables. Plates depict repetition of the whole subgraph. See e.g. Dietz (2010) for a description of graphical notations in generative probabilistic models.

In MLE (MAP) we are only interested in finding a λ_{max} that maximizes the likelihood (posterior) and can therefore omit the denominator. In contrast, in full Bayesian analysis the normalization integral stays and is often the most difficult part to evaluate.

To avoid computational difficulties, we may choose to restrict the form of the nominator. One strategy is to choose the prior and likelihood distributions so that the posterior belongs to the same distribution as the prior, only with different parameters. Combinations of prior and likelihood distributions of this type are called *conjugate prior-likelihood pairs*[7], **conjugate prior** and play a major role in practical Bayesian inference algorithms (Robert and Casella, 2004). Examples of such conjugate prior-likelihood pairs include Normal-Normal (i.e., with a normally distributed prior and likelihood, the posterior is again normally distributed), Beta-Binomial, Gamma-Poisson or (notably) Dirichlet-Multinomial.

LDA Inference

Latent Dirichlet Allocation uses a convex combination of k topic distributions to model observations (documents). The topic probabilities themselves are conditioned on the input document, so that the full generative model can be described with Algorithm 2.1.

The latent parameter space therefore consists of $\boldsymbol{\beta} = \{\beta_i\}$ (the word-by-topic distribution, an $m \times k$ left stochastic matrix), $\boldsymbol{\theta} = \{\theta_j\}$ (the topic-by-documents distribution, a $k \times n$ left stochastic matrix) and $\boldsymbol{z} = \{z_{p,j}\}$ (the topic indicators of each corpus word, with $\sum_{j=1}^{n} |d_j|$ free parameters). See Figure 2.2 for the same process depicted in graphical form. The full joint probability of this model is

$$P(\boldsymbol{w}, \boldsymbol{z}, \boldsymbol{\theta}, \boldsymbol{\beta} | \alpha, \eta) = \prod_{i=1}^{k} P(\beta_i | \eta) \prod_{j=1}^{n} P(\theta_j | \alpha) \prod_{p=1}^{|d_j|} P(z_{p,j} | \theta_j) P(w_{p,j} | \beta_{z_{p,j}}), \tag{2.8}$$

where $|d_j|$ denotes the length of document j in words as before, and α (resp. η) are priors on the document topic mixtures (resp. word-topic distributions).

Inference of these parameters reverses the generative process from Algorithm 2.1—instead of generating observed words from randomly drawn $\boldsymbol{\beta}$, $\boldsymbol{\theta}$ and $\boldsymbol{z}$, we are given the observed words and must estimate these distributions, subject to maximizing data log-likelihood. Table 4.1 on page 76 gives an example of a document with highlighted topic indicators $\boldsymbol{z}$.

Although LDA represents a relatively straightforward model, its exact inference is still intractable. Solutions to this problem include:

[7] The data likelihood function $P(\mathbf{A}|\lambda)$ is typically fixed (determined by the problem), so we only have freedom w.r.t. the choice of the prior distribution $P(\lambda)$. For this reason, the conjugate prior-likelihood pair is sometimes also called only *conjugate prior*, with the likelihood left implicit, though still technically a pair.

Algorithm 2.1: Generative process of words under Latent Dirichlet Allocation

```
Input:
α: hyper-parameters on the document-topic mixing proportion (k-dimensional
vector or a scalar if symmetric).
η: hyper-parameters on the word-topic mixture components (m-dimensional
vector or a single scalar if symmetric).
Output:
$\mathbf{A}_{m \times n}$ corpus of m word types and n documents, with $a_{i,j}$ the frequency of term
i in document j.

// For each topic, draw a word-by-topic distribution $\phi_i$.
foreach $i \in \{1, \dots, k\}$ do
    $\beta_i \sim$ Dirichlet($\eta$)
end
foreach document $j \in \{1, \dots, n\}$ do
    // Draw topic proportions $\theta_j$.
    $\theta_j \sim$ Dirichlet($\alpha$)
    foreach word position p in document j, $p \in \{1, \dots, |d_j|\}$ do
        // Draw an indicator of which topic the word comes from.
        $z_{p,j} \sim$ Multinomial($\theta_j$)
        // Draw a word from that topic's word distribution.
        $w_{p,j} \sim$ Multinomial($\beta_{z_{p,j}}$)
    end
end
// Sum up individual occurrences of each word type in a document, to
   get a bag-of-words frequency vector.
$a_{i,j} \leftarrow \sum_p (w_{p,j} = i)$
```

mean-field variational em proposed in the original (Blei et al., 2003). The core idea is that because we cannot optimize the true posterior directly, we choose another family of distributions (the *variational distribution*) that is tractable, whereby optimizing variational parameters moves the variational distribution closer to the true posterior (minimizes the Kullback-Leibler divergence between them).

expectation propagation from (Minka and Lafferty, 2002).

gibbs sampling as a special instantiation of the *Markov-Chain Monte Carlo (MCMC)* methods. Gibbs sampling avoids the difficulties of **MCMC** dealing with the full joint probability samples by sampling each variable in turn, conditionally independent of all other variables, to produce a chain of samples. In its limit, this chain converges to the true joint probability (Robert and Casella, 2004).

COLLAPSED GIBBS SAMPLING analytically integrates out the latent parameters $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ (which can be done thanks to the Dirichlet-Multinomial conjugacy) and models the topic indicators $P(\boldsymbol{z}|\boldsymbol{w})$ directly (Griffiths and Steyvers, 2004).

COLLAPSED VARIATIONAL INFERENCE combines the idea of collapsed Gibbs sampling (marginalizing out $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$) followed by the mean field variation algorithm that approximates the posterior over topic assignments $\boldsymbol{z}$ (Teh et al., 2007).

Methods based on Gibbs sampling are more Bayesian in nature and track the true posterior; their problem lies in not knowing when we have performed "enough iterations", that is, when the Markov chain has sufficiently converged to our solution. In practice, we let Gibbs sampling run for a certain fixed amount of corpus iterations (the *burn-in*, typically in the hundreds or thousands) and only then start collecting samples. It is also not clear how to relate the samples (the ordering of columns in $\boldsymbol{\beta}$ is arbitrary) to get a coherent set of topics (Teh et al., 2007). Mean-field methods, on the other hand, are guaranteed to converge to a (local) optimum, and typically do so in a smaller number of iterations. Unlike MCMC, they are not dependent on prior conjugacy and can therefore be applied to inference on a broader class of models. Their disadvantage lies in the fact that there is an error introduced by choosing, and optimizing parameters on, a different family of distributions (of which the true posterior is certainly not a member). This bias may result in wildly incorrect parameter estimation, depending on how well we pick the variational distribution.

The α (resp. η) hyper-parameters affect sparsity of the document-topic (resp. word-topic) mixtures. Intuitively, higher concentration values lead to sparser mixtures: fewer relevant words per topic in $\boldsymbol{\phi}$ for β, or fewer relevant topics per document in $\boldsymbol{\theta}$ for α. See (Wallach et al., 2009) and (Asuncion et al., 2009) for a detailed discussion of these hyper-parameter priors, including introduction of asymmetric priors on α and η (also cf. Section 27 in (Knight, 2009)).

2.6 BLAS AND LAPACK

The previous sections briefly summarized several semantic VSM models that have been found successful in a variety of applications. For each method, I tried to list a broader range of (published) applications, without restricting them to the NLP or text processing domains. All the models capture semantic content through explicit use of statistics, so that their performance degrades gradually in the face of input noise, without a sharp discrete drop to uselessness. Another connecting feature is their focus on tractability and reasonable performance, even at the cost of making unrealistic assumptions during model training or even during modelling itself.

The wide applicability to many domains stems from their well understood theoretical foundation, where the inference problems have been delegated to the fields of linear algebra and/or probabilistic reasoning. In this way, advances in these fields contingently improve performance of the semantic methods. A building block at the heart of many algorithms are matrix manipulation routines. Matrix manipulations have been used in Computer Science since its inception (or, put differently, played an important role in its inception). Consequently, there exists a well established research field dealing with improving the efficiency and numerical stability of algorithms for common linear algebra operations.

One of the most famous examples is the *Basic Linear Algebra Subroutines (BLAS)* specification, which is conceptually divided into three levels: **BLAS**

- Level 1 BLAS: Basic vector operations, vector norms, dot products etc.
- Level 2 BLAS: Matrix-vector operations.
- Level 3 BLAS: Matrix-matrix operations.

Another example is *LAPACK*, a library for solving systems of linear **LAPACK** equations and matrix decompositions built on top of BLAS, which superceded the older EISPACK and LINPACK libraries. Practically every major hardware vendor offers its own, heavily optimized implementation for the LAPACK or at least BLAS library: examples include Intel with its *Math Kernel Library (MKL)*, AMD's *Core Math Library (AMCL)*, Apple's *vecLib*, Sun's *Sunperf*, as well as open-sourced implementations like *Automatically Tuned Linear Algebra Software (ATLAS)*, netlib's LAPACK, Kazushige Goto's *GotoBLAS*, the *eigen3* template library, . . . As these routines are used in high performance computing, a lot of effort has been put into their efficient implementations (Anderson et al., 1990). The work on scalability presented in the following chapters naturally devolves on these basic building blocks.

Part I

THE SCALABILITY CHALLENGE

Size matters.
(*anonymous*)

The gap between the amount of digitally available data and our ability to analyze and understand it is widening. Many methods that offer otherwise excellent results require simply too many resources to be applied to modern datasets. Problems that were considered "solved" decades ago re-appear, as raising the amount of input data by orders of magnitude invariably opens up new challenges, requiring new solutions.

In a way, our seemingly never-ending need to process vaster and vaster datasets comes as no surprise. As the AI techniques get more and more sophisticated, so do human expectations of them. There is always a more challenging problem around the corner, a trickier dataset, another group of dissatisfied users. The most exciting Machine Learning problems always seem to be one step ahead of our available, proven "solutions".

The scalability challenge discussed in this chapter will come in two forms: better use of available processing power (scaling in time) and making processing of large input feasible on computers with limited memory (scaling in space). The first problem is typically solved by devising a scheme to split the problem into smaller subproblems to distribute the computation (and then merge the partial solutions), the latter by focusing on models and techniques that summarize all necessary information in a constant amount of space, independent of the size of the training set.

3

LATENT SEMANTIC ANALYSIS

This chapter presents original advances in the theory of scalability of the Latent Semantic Analysis (LSA). The goal is to compute Singular Value Decomposition (the matrix decomposition algorithm at the core LSA) efficiently, in a distributed manner in a small number of passes over the input document stream.

Preliminary results described here were first presented at the *3rd International Conference on Agents and Artificial Intelligence* (Řehůřek, 2011a), followed by a full account at the *33rd European Conference on Information Retrieval (ECIR)* (Řehůřek, 2011b). Analysis of the large scale experiments described in Section 3.5 appeared at the *Low-rank Methods for Large-scale Machine Learning* workshop at the *Neural Information Processing Systems (NIPS) 2010* conference (Řehůřek, 2010).

3.1 MOTIVATION

Matrix decomposition algorithms are commonly used in a variety of domains across much of the field of Computer Science. One of the most widely used matrix decomposition algorithms is the Singular Value Decomposition (SVD), or its closely related eigen decomposition (EVD). These produce a provably optimal (in the least-squares sense) rank-k factorizations when truncated, see Section 2.5.1. In the following, n will denote the number of observations (matrix columns), m the number of features (matrix rows) and k the truncated target rank, $k \ll m \ll n$. In practice, the optimal decompositions are notoriously expensive to compute and truly large-scale applications are rare (Zha et al., 1998; Vigna, 2008). The most common remedy is a) approximation (subsampling the input), b) some sort of incremental updating scheme which avoids recomputing the truncated models from scratch every time an observation/feature is updated, or c) giving up on a globally optimal solution and using another, heuristic algorithm. One way or another, the algorithm must avoid asking for $O(n)$ memory, as the number of observations is assumed to be too large in modern problems.

In this chapter I will introduce two modern approaches to large-scale eigen decomposition that satisfy these criteria: a novel one-pass streamed distributed algorithm and a modified stochastic streamed two-pass algorithm based on (Halko et al., 2009). Both are streamed, meaning no random access to observations is required and their memory requirements are constant in the number of observations. I will also present a hybrid of the two methods here, an algorithm which takes advantage of the speed of two-pass stochastic approach while retaining the one-pass quality of the other algorithm.

As a short recapitulation of Section 2.5.1, Latent Semantic Analysis assumes that each document (or, more generally, observation) can be described using a fixed set of real-valued features (*variables*). These features capture the usage frequency of distinct words in the document, and are typically re-scaled by some *TF-IDF* scheme (Salton, 1971). However, no assumption is made as to what the particular features are or how to extract them from raw data—LSA simply represents the input data collection of n documents, each described by m features, as a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, with documents as columns and features as rows.

In the context of LSA, we can represent documents in a space of much lower dimensionality, $k \ll m$, both saving resources and getting rid of data noise at the same time. For this reason, SVD can be viewed both as a dimensionality reduction and a noise reduction process, improving efficiency and efficacy at the same time.

3.1.1 *SVD Characteristics*

In terms of practical ways of computing SVD, there is an enormous volume of literature (William et al., 1988; Comon and Golub, 1990; Golub and Van Loan, 1996; Zha and Simon, 1999). The algorithms are well-studied and enjoy favourable numerical properties and stability, even in

Table 3.1.: Selected algorithms for truncated, partial eigen-decomposition and their characteristics. "—" stands for *no/not found*.

Algorithm	**Distrib.**	**Incremental in**		**# passes**	**Subspace tracking**	**Implementations**
		observ.	features			
Krylov subspace methods (Lanczos, Arnoldi)	yes	—	—	$O(k)$	—	PROPACK, ARPACK, SVDPACK, MAHOUT, ...
(Halko et al., 2009)	yes	—	—	$O(1)$	—	redsvd, pca.m, our own
(Gorrell and Webb, 2005)	—	—	—	$O(k)$	—	LingPipe, our own
(Zha and Simon, 1999)	—	yes	yes	1	yes	—, our own
(Levy and Lindenbaum, 2000)	—	yes	—	1	yes	—, our own
(Brand, 2006)	—	yes	yes	1	—	—, our own
(Řehůřek, 2011b)	yes	yes	—	1	yes	our own, open-sourced

the face of badly conditioned input. They differ in their focus on what role SVD performs—batch algorithms vs. online updates, optimizing FLOPS vs. number of passes, prioritizing accuracy vs. speed etc.

Table 3.1 enumerates several such interesting characteristics, and evaluates them for a selected set of known algorithms. Its columns have the following interpretation:

DISTRIBUTABLE Can the algorithm run in a distributed manner? Here, I only consider distribution of a very coarse type, where each computing node works autonomously. This type of parallelization is suitable for clusters of commodity computers connected via standard, high-latency networks, as opposed to specialized hardware, supercomputers or fine threading.

INCREMENTAL UPDATES Is the algorithm capable of updating its decomposition as new data arrives, without recomputing everything from scratch? The new data may take form of new observations (documents), or new features (variables). Note that this changes the shape of the **A** matrix. With LSA, we are more interested in whether we can efficiently add new documents, rather than new features. The reason is that vocabulary drift (adding new words; old words acquiring new meanings) is a relatively slow phenomena in natural languages, while new documents appear all the time.

MATRIX STRUCTURE Does the algorithm make use of the structure of the input matrix? In particular, does the algorithm benefit from sparse input? Algorithms that can be expressed in terms of *Basic Linear Algebra Subprograms (BLAS)* routines over the input matrix are relatively easily adapted to any type of input structure.

Another possible distinction here is whether data can be incrementally *subtracted* from an existing decomposition, resulting in a "downdate".

SUBSPACE TRACKING In online streaming environments, new observations come in asynchronously and the algorithm cannot in general store all the input documents in memory (not even out-of-core memory). The incoming observations must be immediate processed and then discarded[1].

Being online has implication on the decomposition itself, because we cannot even afford to keep the truncated right singular vectors $\mathbf{V}$ in memory. The size of $\mathbf{V}_{n \times m}$ is $O(n)$, linear in the number of input documents, which is prohibitive. Therefore, only the U, S matrices are retained and the decomposition is used as a *predictive* (rather than descriptive) model. We call the $\mathbf{P}_{m \times m} \equiv \mathbf{S}^{-1}\mathbf{U}^T$ matrix the *projection matrix*, and the projection process $V_x = \mathbf{P} \cdot x$ is called *folding-in* in the context of LSA[2].

In such subspace tracking scenario, the input data stream can be assumed to be non-stationary. This allows us to introduce an explicit factor for "forgetting" old observations and adjusting the decomposition in favour of new data. This is realized by introducing a parameter $\gamma \in \langle 0.0, 1.0 \rangle$, called the *decay factor*, which dictates the rate of discounting the relevancy of old observations.

[1] This is in contrast to offline, *batch* algorithms, where the whole dataset is presented at once and the algorithm is allowed to go back and forth over the dataset many times.

[2] Note that folding-in is different to updating the decomposition: during folding-in, the $\mathbf{U}$, $\mathbf{S}$ matrices stay intact and an existing model is only used to predict positions of documents in the latent space. In particular, $\mathbf{V}^T_{m \times n} = \mathbf{S}^{-1}\mathbf{U}^T\mathbf{A} = \mathbf{P}_{m \times m}\mathbf{A}_{m \times n}$, so that even though we cannot store the right singular vectors $\mathbf{V}^T$ during computations, they can still be recovered in a streaming fashion if needed, provided one has access to the projection matrix $\mathbf{P}$ and the original collection $\mathbf{A}$.

AVAILABLE IMPLEMENTATIONS While secondary from a theoretical point of view, I consider the availability of a real, executable reference implementation critical for a method's adoption. The application of LSA is relevant to a wider audience who simply do not possess the time or motivation to disentangle terse mathematical equations into functional programs. I also observe that most of the existing SVD implementations (with the notable exceptions of the Apache MAHOUT project and LingPipe) are written in somewhat opaque, FORTRANish style of coding, even when implemented in other languages.

3.1.2 *Related Work*

Historically, most research on SVD optimization has gone into Krylov subspace methods, such as Lanczos-based iterative solvers (see e.g. (Vigna, 2008) for a recent large scale Lanczos-based SVD effort). Our problem is, however, different in that we can only afford a small, *constant number of passes* over the input corpus (Krylov subspace solvers require $O(k)$ passes in general). The special case where the decomposition must be updated on-the-fly, in a *single* pass and in constant memory (for example, whenever the stream of observations cannot be repeated or even stored in off-core memory), can be viewed as an instance of *subspace tracking*. See (Comon and Golub, 1990) for an excellent overview on the complexity of various forms of matrix decomposition algorithms in the context of subspace tracking.

An explicitly formulated $O(m(k+c)^2)$ method (where c is the number of newly added documents, the increment) for incremental LSA updates is presented in (Zha and Simon, 1999). They also give formulas for updating rows of $\mathbf{A}$ as well as rescaling row weights. Their algorithm is completely streamed and runs in constant memory. It can therefore also be used for online subspace tracking, by simply ignoring all updates to the right singular vectors $\mathbf{V}$. The complexity of updates was further reduced in (Brand, 2006) who proposed a linear $O(mkc)$ update algorithm by a series of c fast rank-1 updates. However, in the process, the ability to track subspaces is lost: the improvement is realized by appending new columns to the orthonormal basis $\mathbf{U}$ each time this basis has insufficient rank, avoiding the costly $O(mk^2)$ rotations otherwise necessary to keep $\mathbf{U}$ in truncated form. As there are in general $O(n)$ rank increasing updates, this means $\mathbf{U}$ grows to consume $O(n)$ memory, which is prohibitive in subspace tracking. Their approach is akin to the k-*Nearest Neighbours (*k*-NN)* method of Machine Learning: the lightning speed during training is offset by memory requirements of storing the model.

These methods concern themselves with adding new documents to an existing decomposition. What is needed for a distributed version of LSA is a slightly different task: given two existing decompositions, merge them together into one. I did not find any explicit, efficient algorithm for merging decompositions in the literature. I will therefore seek to close this gap, provide such algorithm and use it for computing distributed LSA. The following section describes the algorithm and states conditions under which the merging makes sense when dealing with only truncated rank-k approximation of the decomposition.

3.2 DISTRIBUTED SINGLE PASS ALGORITHM

In this section, I derive an algorithm for distributed online computing of LSA over a cluster of computers, in a single pass over the input matrix. The algorithm is flexible enough to also realize subspace tracking (only tracking the term space $\mathbf{U}_{m \times k}$, while ignoring updates to the (prohibitively large) document space $\mathbf{V}_{n \times k}$), and includes an optional model decay factor to account for subspace drift through its γ parameter.

3.2.1 *Overview*

Distribution will be achieved by column-partitioning the input matrix $\mathbf{A}$ into several smaller submatrices, called *jobs*,

$$\mathbf{A}^{m \times n} = \left[\mathbf{A}_1^{m \times c_1}, \mathbf{A}_2^{m \times c_2}, \cdots, \mathbf{A}_j^{m \times c_j}\right], \sum_{i=1}^{j} c_i = n.$$

Since columns of $\mathbf{A}$ correspond to documents, each job $\mathbf{A}_i$ amounts to processing a chunk of c_i input documents. The sizes of these chunks are chosen to fit available resources of the processing nodes: bigger chunks mean faster overall processing but on the other hand consume more memory.

Jobs are then distributed among the available cluster nodes, in no particular order, so that each node will be processing a different set of column-blocks from $\mathbf{A}$. The nodes need not process the same number of jobs, nor process jobs at the same speed; the computations are completely asynchronous and independent. Once all jobs have been processed, the decompositions accumulated in each node will be merged into a single, final decomposition $P = (\mathbf{U}, \mathbf{S})$ (see Section 3.1.1 on subspace tracking for where $\mathbf{V}^T$ disappeared). As a reminder, $\mathbf{U}$ and $\mathbf{S}$ are respectively an orthonormal and a diagonal matrix such that $\mathbf{A} = \mathbf{USV}^T$, or equivalently and perhaps more naturally for avoiding mentioning the unused $\mathbf{V}^T$, such that $\mathbf{AA}^T = \mathbf{US}^2\mathbf{U}^T$. The former factorization is called the Singular Value Decomposition, the latter is its related eigen decomposition.

What is needed are thus two algorithms:

1. BASE DECOMPOSITION: In main memory, find $P_i = (\mathbf{U}_i^{m \times c_i}, \mathbf{S}_i^{c_i \times c_i})$ eigen decomposition of a single job $\mathbf{A}_i^{m \times c_i}$ such that $\mathbf{A}_i\mathbf{A}_i^T = \mathbf{U}_i\mathbf{S}_i^2\mathbf{U}_i^T$.

2. MERGE DECOMPOSITIONS: Merge $P_i = (\mathbf{U}_i, \mathbf{S}_i)$, $P_j = (\mathbf{U}_j, \mathbf{S}_j)$ of two jobs $\mathbf{A}_i$, $\mathbf{A}_j$ into a single decomposition $P = (\mathbf{U}, \mathbf{S})$ such that $\left[\mathbf{A}_i, \mathbf{A}_j\right]\left[\mathbf{A}_i, \mathbf{A}_j\right]^T = \mathbf{US}^2\mathbf{U}^T$. Merging will allow us to construct the final decomposition of $\mathbf{A}$ in a hierarchical manner, starting with merging the base decompositions, then merging the merged results and so on, up to the decomposition of the whole matrix $\mathbf{A}$.

I would like to highlight the fact that the first algorithm will perform decomposition of a *sparse* input matrix, while the second algorithm will merge two *dense* decompositions into another dense decomposition. This is in contrast to incremental updates discussed in the literature (Brand, 2006; Levy and Lindenbaum, 2000; Zha and Simon, 1999), where the existing decomposition and the new documents are mashed together into a single matrix, losing any potential benefits of sparsity as well as severely limiting the possible size of a job due to increased memory requirements. The explicit merge procedure also makes the distributed version of the algorithm straightforward, so that the computation can be split across a cluster of computers.

3.2.2 *Solving the Base Case*

In Latent Semantic Analysis, density of the job matrices is well below 1%, so a sparse solver is called for. Also, a direct sparse SVD solver of A_i is preferable to the roundabout eigen decomposition of $\mathbf{A}_i\mathbf{A}_i^T$, both to save memory and to improve numerical accuracy (see e.g. (Golub and Van Loan, 1996)). Finally, because $k \ll m$, a partial decomposition is required which only returns the k greatest factors—computing the full spectrum would be a terrible overkill. There exist a multitude of partial sparse SVD solvers that work in-core. We view the particular implementation as "black-box" and note that the Lanczos-based implementations mentioned in Table 3.1 are particularly suitable for this in-memory task.

3.2.3 *Merging Decompositions*

No efficient algorithm (as far as I know) exists for merging two truncated eigen decompositions (or SVD decompositions) into one. I therefore propose my own, novel algorithm here, starting with its derivation and summing up the final version in the end.

The problem can be stated as follows. Given two truncated eigen decompositions $P_1 = (\mathbf{U}_1^{m \times k_1}, \mathbf{S}_1^{k_1 \times k_1})$, $P_2 = (\mathbf{U}_2^{m \times k_2}, \mathbf{S}_2^{k_2 \times k_2})$, which come from the (by now lost and unavailable) input matrices $\mathbf{A}_1^{m \times c_1}$, $\mathbf{A}_2^{m \times c_2}$, $k_1 \leqslant c_1$ and $k_2 \leqslant c_2$, find $P = (\mathbf{U}, \mathbf{S})$ that is the eigen decomposition of $[\mathbf{A}_1, \mathbf{A}_2]$.

Our first approximation will be the direct naïve

$$\mathbf{U}, \mathbf{S}^2 \xleftarrow{\text{eigen}} [\mathbf{U}_1\mathbf{S}_1, \mathbf{U}_2\mathbf{S}_2]\,[\mathbf{U}_1\mathbf{S}_1, \mathbf{U}_2\mathbf{S}_2]^T. \tag{3.1}$$

This is terribly inefficient, and forming the matrix product of size $m \times m$ on the right hand side is prohibitively expensive. Writing SVD_k for truncated SVD that returns only the k greatest singular numbers and their associated singular vectors, we can equivalently write

$$\mathbf{U},\mathbf{S},\mathbf{V}^{\mathsf{T}} \xleftarrow{\mathrm{SVD}_k} \left[\gamma\mathbf{U}_1\mathbf{S}_1,\mathbf{U}_2\mathbf{S}_2\right]. \tag{3.2}$$

This is more reasonable, with the added bonus of increased numerical accuracy over the related eigen decomposition. Note, however, that the computed right singular vectors $\mathbf{V}^{\mathsf{T}}$ are needed at all, which is a sign of further inefficiency. Also, the fact that $\mathbf{U}_1, \mathbf{U}_2$ are orthonormal is completely ignored. This leads us to break the algorithm into several steps:

Algorithm 3.1: Baseline merge

```
Input: Truncation factor $k$, decay factor $\gamma$, $P_1 = (\mathbf{U}_1^{m\times k_1}, \mathbf{S}_1^{k_1\times k_1})$,
       $P_2 = (\mathbf{U}_2^{m\times k_2}, \mathbf{S}_2^{k_2\times k_2})$
Output: $P = (\mathbf{U}^{m\times k}, \mathbf{S}^{k\times k})$
1 $\mathbf{Q},\mathbf{R} \xleftarrow{\mathrm{QR}} [\gamma\mathbf{U}_1\mathbf{S}_1, \mathbf{U}_2\mathbf{S}_2]$
2 $\mathbf{U}_R, \mathbf{S}, \mathbf{V}_R^{\mathsf{T}} \xleftarrow{\mathrm{SVD}_k} \mathbf{R}$
3 $\mathbf{U}^{m\times k} \leftarrow \mathbf{Q}^{m\times(k_1+k_2)}\mathbf{U}_R^{(k_1+k_2)\times k}$
```

On line 1, an orthonormal subspace basis $\mathbf{Q}$ is found which spans both of the subspaces defined by columns of $\mathbf{U}_1$ and $\mathbf{U}_2$, $\mathrm{span}(\mathbf{Q}) = \mathrm{span}([\mathbf{U}_1, \mathbf{U}_2])$. Multiplications by $\mathbf{S}_1$, $\mathbf{S}_2$ and γ provide scaling for $\mathbf{R}$ only and do not affect $\mathbf{Q}$ in any way, as $\mathbf{Q}$ will always be column-orthonormal. Our algorithm of choice for constructing the new basis is QR factorization, because we can use its other product, the upper trapezoidal matrix $\mathbf{R}$, to our advantage. [3] Now we're almost ready to declare $(\mathbf{Q},\mathbf{R})$ our target decomposition $(\mathbf{U},\mathbf{S})$, except $\mathbf{R}$ is not diagonal. To diagonalize the small matrix $\mathbf{R}$, we perform an SVD on it, on line 2. This gives us the singular values $\mathbf{S}$ we need as well as the rotation of $\mathbf{Q}$ necessary to represent the basis in this new subspace. The rotation

[3] The subspaces spanned by $\mathbf{U}_1$ and $\mathbf{U}_2$ will often intersect, which causes $\dim(\mathbf{Q}) = s < k_1 + k_2 = \dim(\mathbf{U}_1) + \dim(\mathbf{U}_2)$ and $\mathbf{R}$ will have zeros on the diagonal. Implementations of the QR algorithm will typically permute $\mathbf{R}$, pushing the $s - (k_1 + k_2)$ diagonal zeros only after all non-zeros, restoring diagonal shape in the upper left part of $\mathbf{R}$. This is of no consequence to our discussion, because $\mathbf{Q}$ is always permuted accordingly.

is applied on line 3. Finally, both output matrices are truncated to the requested rank k. The costs are $O(m(k_1+k_2)^2)$, $O((k_1+k_2)^3)$ and $O(m(k_1+k_2)^2)$ for line 1, 2 and 3 respectively, for a combined total of $O(m(k_1+k_2)^2)$.

Although more elegant than the direct decomposition given by Equation 3.2, the baseline algorithm is only marginally more efficient than the direct SVD. This comes as no surprise, as the two algorithms are quite similar and SVD of rectangular matrices is often internally implemented by means of QR in exactly this way. Luckily, we can do better.

First, we observe that the QR decomposition makes no use of the fact that $\mathbf{U}_1$ and $\mathbf{U}_2$ are already orthogonal. Capitalizing on this will allow us to represent $\mathbf{U}$ as an update to the existing basis $\mathbf{U}_1$, $\mathbf{U} = \left[\mathbf{U}_1, \mathbf{U}'\right]$, dropping the complexity of the first step to $O(mk_2^2)$. Secondly, the application of rotation $\mathbf{U}_R$ to $\mathbf{U}$ can be rewritten as $\mathbf{U}\mathbf{U}_R = \left[\mathbf{U}_1, \mathbf{U}'\right]\mathbf{U}_R = \mathbf{U}_1\mathbf{R}_1 + \mathbf{U}'\mathbf{R}_2$, dropping the complexity of the last step to $O(mkk_1 + mkk_2)$. Plus, the algorithm can be made to work by modifying the existing matrices $\mathbf{U}_1, \mathbf{U}_2$ in place inside BLAS routines, which is a considerable practical improvement over Algorithm 3.1, which requires allocating additional $m(k_1+k_2)$ floats.

Algorithm 3.2: Optimized merge

```
Input: Truncation factor k, decay factor $\gamma$, $P_1 = (\mathbf{U}_1^{m\times k_1}, \mathbf{S}_1^{k_1\times k_1})$,
       $P_2 = (\mathbf{U}_2^{m\times k_2}, \mathbf{S}_2^{k_2\times k_2})$
Output: $P = (\mathbf{U}^{m\times k}, \mathbf{S}^{k\times k})$
$\mathbf{Z}^{k_1\times k_2} \leftarrow \mathbf{U}_1^T\mathbf{U}_2$
$\mathbf{U}', \mathbf{R} \xleftarrow{QR} \mathbf{U}_2 - \mathbf{U}_1\mathbf{Z}$
$\mathbf{U}_R, \mathbf{S}, \mathbf{V}_R^T \xleftarrow{SVD_k} \begin{bmatrix} \gamma\mathbf{S}_1 & \mathbf{Z}\mathbf{S}_2 \\ 0 & \mathbf{R}\mathbf{S}_2 \end{bmatrix}^{(k_1+k_2)\times(k_1+k_2)}$
$\begin{bmatrix} \mathbf{R}_1^{k_1\times k} \\ \mathbf{R}_2^{k_2\times k} \end{bmatrix} = \mathbf{U}_R$
$\mathbf{U} \leftarrow \mathbf{U}_1\mathbf{R}_1 + \mathbf{U}'\mathbf{R}_2$
```

The first two lines construct the orthonormal basis $\mathbf{U}'$ for the component of $\mathbf{U}_2$ that is orthogonal to $\mathbf{U}_1$; $\text{span}(\mathbf{U}') = \text{span}((\mathbf{I} - \mathbf{U}_1\mathbf{U}_1^T)\mathbf{U}_2) = \text{span}(\mathbf{U}_2 - \mathbf{U}_1(\mathbf{U}_1^T\mathbf{U}_2))$.

As before, we use QR factorization because the upper trapezoidal matrix $\mathbf{R}$ will come in handy when determining the singular vectors $\mathbf{S}$.

Line 3 is perhaps the least obvious, but follows from the requirement that the updated basis $[\mathbf{U}, \mathbf{U}']$ must satisfy

$$[\mathbf{U}_1\mathbf{S}_1, \mathbf{U}_2\mathbf{S}_2] = [\mathbf{U}_1, \mathbf{U}']\,\mathbf{X}, \tag{3.3}$$

so that

$$\mathbf{X} = [\mathbf{U}_1, \mathbf{U}']^T [\mathbf{U}_1\mathbf{S}_1, \mathbf{U}_2\mathbf{S}_2] = \begin{bmatrix} \mathbf{U}_1^T\mathbf{U}_1\mathbf{S}_1 & \mathbf{U}_1^T\mathbf{U}_2\mathbf{S}_2 \\ \mathbf{U}'^T\mathbf{U}_1 & \mathbf{U}'^T\mathbf{U}_2\mathbf{S}_2 \end{bmatrix}. \tag{3.4}$$

Using the equalities $\mathbf{R} = \mathbf{U}'^T\mathbf{U}_2$, $\mathbf{U}'^T\mathbf{U}_1 = 0$ and $\mathbf{U}_1^T\mathbf{U}_1 = \mathbf{I}$ (all by construction) we obtain

$$\mathbf{X} = \begin{bmatrix} \mathbf{S}_1 & \mathbf{U}_1^T\mathbf{U}_2\mathbf{S}_2 \\ 0 & \mathbf{U}'^T\mathbf{U}_2\mathbf{S}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{S}_1 & \mathbf{Z}\mathbf{S}_2 \\ 0 & \mathbf{R}\mathbf{S}_2 \end{bmatrix}. \tag{3.5}$$

Line 4 is just a way of saying that on line 5, $\mathbf{U}_1$ will be multiplied by the first k_1 rows of $\mathbf{U}_R$, while $\mathbf{U}'$ will be multiplied by the remaining k_2 rows. Finally, line 5 seeks to avoid realizing the full $[\mathbf{U}_1, \mathbf{U}']$ matrix in memory and is a direct application of the equality

$$[\mathbf{U}_1, \mathbf{U}']^{m\times(k_1+k_2)}\,\mathbf{U}_R^{(k_1+k_2)\times k} = \mathbf{U}_1\mathbf{R}_1 + \mathbf{U}'\mathbf{R}_2. \tag{3.6}$$

As for complexity of this algorithm, it is again dominated by the matrix products and the dense QR factorization, but this time only of a matrix of size $m \times k_2$. The SVD of line 3 is a negligible $O(k_1 + k_2)^3$, and the final basis rotation comes up to $O(mk \max(k_1, k_2))$. Overall, with $k_1 \approx k_2 \approx k$, this is an $O(mk^2)$ algorithm.

In Section 3.4, I will compare the runtime speed of both these proposed merge algorithms on real corpora.

3.2.4 *Effects of Truncation*

While the equations above are exact when using matrices of full rank, it is not at all clear how to justify truncating all intermediate matrices to rank k in each update. What effect does this have on the merged decomposition? How do these effects stack up as we perform several updates in succession?

In (Zha and Zhang, 2000), the authors identified general conditions under which operating with truncated matrices produces exact results. Their results are applicable to our algorithm as well. Moreover, they show by way of perturbation analysis that the results are stable (though no longer exact) even if the input matrix only approximately satisfies this condition. They investigate matrices of the so-called *low-rank-plus-shift* structure, which (approximately) satisfy

$$\mathbf{A}^T\mathbf{A}/m \approx \mathbf{C}\mathbf{W}\mathbf{C}^T + \sigma^2\mathbf{I_n}, \tag{3.7}$$

where $\mathbf{C}$, $\mathbf{W}$ are of rank k, $\mathbf{W}$ positive semi-definite, σ^2 the variance of noise. That is, $\mathbf{A}^T\mathbf{A}$ can be expressed as a sum of a low-rank matrix and a multiple of the identity matrix. They show that matrices coming from natural language corpora under the bag-of-words paradigm do indeed possess the necessary structure and that in this case, a rank-k approximation of $\mathbf{A}$ can be expressed as a combination of rank-k approximations of its submatrices without a serious loss of precision,

$$\text{best}_k(\left[\mathbf{A}_1, \mathbf{A}_2\right]) = \text{best}_k(\left[\text{best}_k(\mathbf{A}_1), \text{best}_k(\mathbf{A}_2)\right]). \tag{3.8}$$

Interestingly, the authors realize the potential for a divide-and-conquer style distributed algorithm stemming from this formula. However, they only present a serial version for incrementally adding new documents. In a rather Fermat-like fashion, they gloss over the distributed version with "Due to space limitation, the high performance computing issues related to the divide-and-conquer approach will not be further discussed here." (Zha et al., 1998) and proceed to describe a Lanczos-based (non-incremental) MPI implementation of distributed SVD, which is a batch algorithm unsuitable for streaming environments.

3.2.5 *Putting It Together*

Let $N = \{N_1, \ldots, N_p\}$ be the p available cluster nodes. Each node will be processing incoming jobs sequentially, running the base case decomposition on each job followed by merging the result with the current model by means of Merge Algorithm 3.1 or 3.2.

Algorithm 3.3: LSA_Node_N_i

```
Input: Truncation factor $k$, Queue of jobs $\mathbf{A}_1, \mathbf{A}_2, \ldots$
Output: $P = (\mathbf{U}^{m \times k}, \mathbf{S}^{k \times k})$ decomposition of $[\mathbf{A}_1, \mathbf{A}_2, \ldots]$

$P = (\mathbf{U}, \mathbf{S}) \leftarrow 0^{m \times k}, 0^{k \times k}$
foreach job $\mathbf{A}_i$ do
    $P' = (\mathbf{U}', \mathbf{S}') \leftarrow$ Basecase_Algo$(k, \mathbf{A}_i)$
    $P \leftarrow$ Merge_Algo$(k, P, P')$
end
```

To construct decomposition of the full matrix $\mathbf{A}$, we let the p nodes work in parallel, distributing the jobs as soon as they arrive, to whichever node seems idle. I do not describe the technical issues of load balancing and recovery from node failure here, but standard practices apply[4]. Once we have processed all the jobs (or temporarily exhausted the input job queue, in the infinite streaming scenario), we merge the p individual decompositions into one, by means of Merge Algorithm 3.1 or 3.2.

Algorithm 3.4: Distributed LSA

```
Input: Truncation factor $k$, Queue of jobs $\mathbf{A} = [\mathbf{A}_1, \mathbf{A}_2, \ldots]$
Output: $P = (\mathbf{U}^{m \times k}, \mathbf{S}^{k \times k})$ decomposition of $\mathbf{A}$

1 $P_i = (\mathbf{U}_i, \mathbf{S}_i) \leftarrow$ LSA_Node_$N_i$(k, subset of jobs from $\mathbf{A}$), for
  $i = 1, \ldots, p$
2 $P \leftarrow$ Reduce(Merge_Algo, $[P_1, \ldots, P_p]$)
```

Here, line 1 is executed in parallel, making use of all p processing nodes at once, and is the source of parallelism of the algorithm. On line two, *Reduce* applies the function that is its first argument cummulatively to the sequence that is its second argument, so that it effectively

[4] A note on implementation: there are a number of available choices for a parallel programming frameworks: the MPI standard (with its Open MPI implementation), the Parallel Virtual Machine (PVM) or Remote Procedure Calls (RPC).

merges P_1 with P_2, followed by merging that result with P_3, etc. Several divide-and-conquer schemes that realize this final merging are possible. A particularly appealing one is where the p decompositions are merged in pairs coming from approximately the same number of input documents, so that the two sets of merged singular values are of comparable magnitude. Doing so promises to lead to improved numerical properties, but I have not investigated this effect in detail.

The algorithm is formulated in terms of a (potentially infinite) sequence of jobs, so that when more jobs arrive, we can continue updating the decomposition in a natural way. The whole algorithm can act as a continuous daemon service, providing LSA decomposition of all the jobs processed so far on demand.

3.3 STREAMED TWO PASS ALGORITHM

As far as I know, the algorithm derived in the previous section is the first published distributed single pass decomposition algorithm for streaming data collections. If we relax the constraint on the number of passes to a small, *constant* number (though no longer necessarily one), we can also use the algorithm from (Halko et al., 2009) for in-core decompositions. They describe several variants of this efficient stochastic algorithm and analyze their properties. However, the one-pass stochastic algorithm as described there is unsuitable for large-scale decompositions, because the computation requires $O(nk + mk)$ memory.

I propose we can reduce this to a manageable $O(mk)$, i.e. independent of the input stream size n, at the cost of running *two passes* over the input matrix instead of one[5]. This is achieved by two optimizations:

1. the sample matrix is constructed piece-by-piece from the stream, instead of a direct matrix multiplication, and
2. the final dense decomposition is performed on a smaller $k \times k$ eigenproblem $\mathbf{BB}^T$ instead of the full $k \times n$ matrix $\mathbf{B}$.

These two optimizations allow us to compute the decomposition in constant memory, by processing the observations one after another, or,

[5] Actually, $2 + q$ passes are needed when using q power iterations.

preferably, in as large chunks as fit into core memory. The result is summarized in Algorithm 3.5, an $O(1)$ pass algorithm published in (Řehůřek, 2010). Although this algorithm is no longer single-pass, it only requires a small, constant number of passes over the input collection, and may therefore still be a good choice in some circumstances. It is worth pointing out at this place that streamed one-pass algorithms (like the one described earlier) are fundamentally different from the 2-pass algorithm described here (or any other q-pass algorithm, albeit with a small $q > 1$), in that they allow us to process infinite input streams. In environments where the input cannot be persistently recorded and stored, a single pass algorithm is the only option.

infinite input streams

3.3.1 *Hybrid Single Pass Algorithm*

Although the two-pass Algorithm 3.5 cannot be directly used in single-pass environments, we can use the flexibility of the distributed single pass algorithm and plug the stochastic decomposition as the "black-box" Basecase decomposition into Algorithm 3.3.

Algorithm 3.5: Two-pass Stochastic Decomposition in Constant Memory with Streamed Input

```
Input: $m \times n$ input matrix $\mathbf{A}$, presented as a stream of observation chunks
       $\mathbf{A} = [\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_C]$. Truncation factor $k$. Oversampling factor $l$.
       Number of power iterations $q$.
Output: $\mathbf{U}$, $\mathbf{S}^2$ spectral decomposition of $\mathbf{A}$ (i.e., $\mathbf{U}\mathbf{S}^2\mathbf{U}^T = \mathbf{A}\mathbf{A}^T$) truncated
        to the $k$ greatest factors.
Data: Intermediate matrices require $O(m(k+l))$ memory; in particular,
      the algorithm avoids materializing any $O(n)$ or $O(m^2)$ size matrices.

// Construct the $m \times (k+l)$ sample matrix $\mathbf{Y} = \mathbf{AO}$, in one pass
   over the input stream.
$\mathbf{Y} \leftarrow$ sum($\mathbf{A}_i\mathbf{O}_i$ for $\mathbf{A}_i$ in $\mathbf{A}$) ; // each $\mathbf{O}_i$ is a random $|\mathbf{A}_i| \times (k+l)$
gaussian matrix
// Run q power iterations to improve accuracy (optional),
   $\mathbf{Y} = (\mathbf{A}\mathbf{A}^T)^q\mathbf{AO}$.  Needs q extra passes.
for $iteration \leftarrow 1$ to $q$ do
    $\mathbf{Y} \leftarrow$ sum($\mathbf{A}_i(\mathbf{A}_i^T\mathbf{Y})$ for $\mathbf{A}_i$ in $\mathbf{A}$);
end
// Construct the $m \times (k+l)$ orthonormal action matrix $\mathbf{Q}$, in-core.
$\mathbf{Q} \leftarrow orth(\mathbf{Y})$;
// Construct $(k+l) \times (k+l)$ covariance matrix $X = \mathbf{B}\mathbf{B}^T$ in one
   pass, where $\mathbf{B} = \mathbf{Q}^T\mathbf{A}$.
$\mathbf{X} \leftarrow$ sum($(\mathbf{Q}^T\mathbf{A}_i)(\mathbf{Q}^T\mathbf{A}_i)^T$ for $\mathbf{A}_i$ in $\mathbf{A}$) ; // BLAS rank-$k$ update routine
SYRK
// Compute $\mathbf{U}$, $\mathbf{S}$ by means of the small $(k+l) \times (k+l)$ matrix $\mathbf{X}$.
$\mathbf{U}_X, \mathbf{S}_X \leftarrow eigh(\mathbf{X})$;
// Go back from the eigen values of $\mathbf{X}$ to the eigen values of $\mathbf{B}$
   (= eigen values of $\mathbf{A}$).
$\mathbf{S}^2 \leftarrow$ first $k$ values of $\sqrt{\mathbf{S}_X}$;
$\mathbf{U} \leftarrow$ first $k$ columns of $\mathbf{Q}\mathbf{U}_X$;
```

3.4 EXPERIMENTS

In this section, I will describe two sets of experiments. The first set concerns itself with numerical accuracy of the proposed single-pass algorithm, the second with its performance.

In all experiments, the decay factor γ is set to 1.0, that is, there is no discounting in favour of new observations. The number of requested factors is $k = 200$ for the small and medium corpus and $k = 400$ for the large Wikipedia corpus.

3.4.1 *Algorithms*

I will be comparing four implementations for partial Singular Value Decomposition:

SVDLIBC A direct sparse SVD implementation due to Douglas Rohde[6]. SVDLIBC is based on the SVDPACK package by Michael Berry (Berry, 1992). I used its LAS2 routine to retrieve only the k dominant singular triplets.

ZMS implementation of the incremental one-pass algorithm from (Zha et al., 1998). All the operations involved can be expressed in terms of Basic Linear Algebra Subroutines (BLAS). For this reason I use the NumPy library, which makes use of whatever LAPACK library is installed in the system, to take advantage of fast blocked routines. The right singular vectors and their updates are completely ignored so that my implementation of their algorithm also realizes subspace tracking.

DLSA My proposed method. I will be evaluating two different versions of the merging routine, Algorithms 3.1 and 3.2, calling them $DLSA_1$ and $DLSA_2$ in the tables. I will also observe effects of varying the job sizes c and the number of cluster nodes p. Again, NumPy is used for dense matrix operations. The base case decomposition is realized by an adapted LAS2 routine from SVDLIBC.

[6] `http://tedlab.mit.edu/~dr/SVDLIBC/`, accessed in March 2010.

HEBB Streamed stochastic Hebbian algorithm from (Gorrell and Webb, 2005) which loops over the input dataset, in $k * epochs$ passes, to converge at the singular triplets. The straightforward implementation suffered from serious convergency issues that I couldn't easily fix, so I only include it in the comparisons for the smallest dataset. This algorithm internally updates the singular triplets with explicit array loops (no BLAS).

With the exception of SVDLIBC, all the other algorithms operate in a streaming fashion, so that the corpus need not reside in core memory all at once. Also note that although memory footprint of the subspace tracking algorithms ZMS and DLSA is independent of the size of the corpus, it is still linear in the number of features, $O(m)$. It is assumed that the decomposition $(U^{m \times k}, S^{k \times k})$ fits entirely into core memory.

3.4.2 *Datasets*

For the experiments, I will be using three datasets.

MEDIUM SIZE CORPUS A corpus of 61,293 mathematical articles collected from the digital libraries of NUMDAM, arXiv and DML-CZ. Together these comprise about 270 million corpus positions, with over 6 million unique word types (I parse out mathematical equations and use them as separate word types). After the standard procedure of pruning out word types that are too infrequent (hapax legomena, typos, OCR errors, etc.) or too frequent (stop words), we are left with 315,002 distinct features. The final matrix $A^{315,002 \times 61,293}$ has 33.8 million non-zero entries, with density less than 0.18%. This corpus was chosen so that it fits into core memory of a single computer and its decomposition can therefore be computed directly. This will allow us to establish the "ground-truth" decomposition and set an upper bound on achievable accuracy and speed.

SMALL CORPUS A subset of 3,494 documents from the medium size corpus. It contains 39,022 features and the sparse $A^{39,022 \times 3,494}$ matrix has 1,446,235 non-zero entries, so that it is about 23 times smaller than the medium size corpus.

LARGE CORPUS The last corpus is the English Wikipedia[7]. This corpus contains 3.2 million documents covering almost 8 million distinct word types in 2 billion tokens. I clip the vocabulary size to the 100,000 most frequent word types, after discarding all words that appear in more than 10% of the documents ("stop-list"). This leaves us with a sparse term-documents matrix with 0.5G non-zero entries, or 14 times the size of the medium corpus.

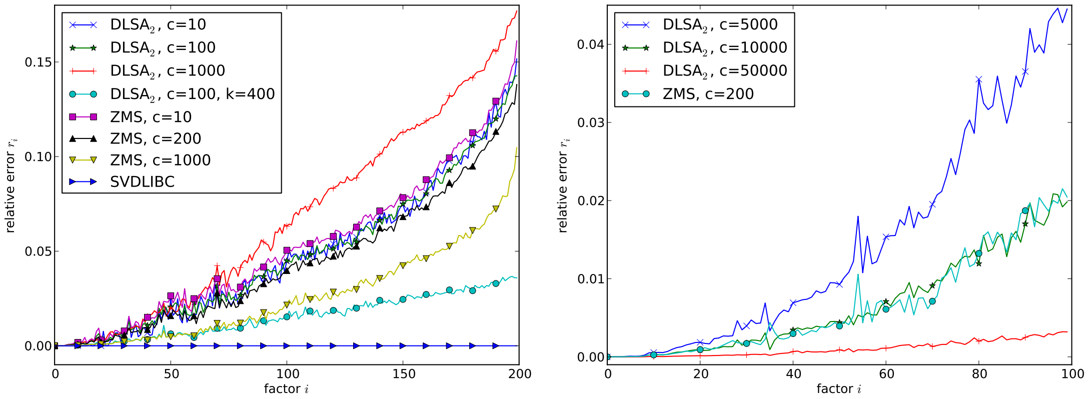


Figure 3.1.: Accuracy of singular values for various decomposition algorithms on the small corpus (left) and medium corpus (right).

3.4.3 *Accuracy*

Figure 3.1 plots the relative accuracy of singular values found by DLSA, ZMS, SVDLIBC and HEBB algorithms compared to known, "ground-truth" values S_G. I measure accuracy of the computed singular values $\overline{S}$ as $r_i = |\overline{s_i} - s_{Gi}|/s_{Gi}$, for $i = 1, \ldots, k$. The ground-truth singular values S_G are computed directly with LAPACK's DGESVD routine for the small corpus and with SVDLIBC's LAS2 routine for the medium corpus.

We observe that the largest singular values are practically always exact, and accuracy quickly degrades towards the end of the returned

[7] The latest static dump as downloaded from `http://download.wikimedia.org/enwiki/latest`, June 2010.

spectrum. This leads us to the following refinement: When requesting x factors, compute the truncated updates for $k > x$, such as $k = 2x$, and discard the extra $x - k$ factors only when the final projection is actually needed. This approach is marked as "DLSA2, c=100, k=400" in Figure 3.1 (left) and then used as default in the larger experiments on the medium corpus. The error is then below 5%, which is comparable to the ZMS algorithm (while DLSA is at least an order of magnitude faster, even without any parallelization).

Even with this refinement, the error is not negligible, so we are naturally interested in how it translates into error of the whole LSA application. This way of testing has the desirable effect that errors in decomposition which do not manifest themselves in the subsequent application do not affect the evaluation, while decomposition errors that carry over to the application are still correctly detected.

To this end, I conducted another set of accuracy experiments. In Latent Semantic Analysis, the most common application is measuring cosine similarity between documents represented in the new, "latent semantic" space. We will compute inter-document similarity of the entire input corpus, forming an $n \times n$ matrix $\mathbf{C}$, where each entry $c_{i,j} = \text{cossim}(\overline{\text{doc}_i, \text{doc}_j})$ (see Section 2.3.2). We do the same thing for the corpus represented by the ground truth decomposition, obtaining another $n \times n$ matrix. Difference between these two $n \times n$ matrices (measured by Root Mean Square Error, or RMSE) then gives us a practical estimate of the error introduced by the given decomposition[8]. Note that in addition to testing the magnitude of the singular values, this also tests accuracy of the singular vectors at the same time. In Table 3.2, we can observe that the error of DLSA2 is around 2%, which is usually acceptable for assessment of document similarity and for document ranking.

[8] This is a round-about sort of test—to see how accurate a decomposition is, we use it to solve a superordinate task (similarity of documents), then compare results of this superordinate task against a known ground truth.

Table 3.2.: Decomposition accuracy on the small corpus, measured by RMSE of document similarities based on ground truth vs. given algorithm.

Algorithm	Job size	RMSE	Algorithm	Job size	RMSE
SVDLIBC	3,494	0.0	DLSA2	10	0.0204
ZMS	10	0.0204	DLSA2	100	0.0199
ZMS	200	0.0193	DLSA2	1,000	0.0163
ZMS	1,000	0.0162	DLSA2	100, $k = 400$	0.0094

3.4.4 *Performance*

Performance was measured as wall-clock time on a cluster of four dual-core 2GHz Intel Xeons, each with 4GB of RAM, which share the same Ethernet segment and communicate via TCP/IP. The machines were not dedicated but their load was reasonably low during the course of the experiments. To make sure, I ran each experiment three times and report the best achieved time. These machines did not have any optimized BLAS library installed, so I also ran the same experiments on a "cluster" of one node, a dual-core 2.53GHz MacBook Pro with 4GB RAM and *vecLib*, a fast BLAS/LAPACK library provided by the vendor. This HW setup is marked as "serial" in the result tables, to differentiate it from the otherwise equivalent 1-node setup coming from the BLAS-less four-node cluster.

Table 3.3 summarizes performance results for the small corpus. For ZMS, the update time is proportional to *number of updates* · *cost of update* $\approx \lceil \frac{n}{c} \rceil \cdot m(k+c)^2$, so that the minimum (fastest execution) is necessarily attained by setting job size $c = k$. Overall, we can see that the direct in-core SVDLIBC decomposition is the fastest. The HEBB implementation was forcefully terminated after one hour, with some estimated eight hours left to complete. The speed of DLSA2 approaches the speed of SVDLIBC as the job size increases; in fact, once the job size reaches the size of the whole corpus, it becomes equivalent to SVDLIBC. However, unlike SVDLIBC, DLSA can proceed in document chunks smaller than the whole corpus, so that corpora that do not fit in

Table 3.3.: Performance of selected partial decomposition algorithms on the small corpus, $A^{39,022\times 3,494}$. Times are in seconds, on the serial setup.

Algorithm	**Job size** c	**Time taken [s]**
HEBB	N/A	> 1h
SVDLIBC	3,494	16
ZMS	10	346
ZMS	100	165
ZMS	200	150
ZMS	500	166
ZMS	1,000	194
ZMS	2,000	out of memory

(a) Serial algorithms

Job size c	**DLSA$_2$**	**DLSA$_1$**
10	190	2,406
100	122	350
1,000	38	66
3,494	21	21

(b) DLSA variants

RAM can be processed, and so that different document chunks can be processed on different nodes in parallel.

Table 3.4.: Performance of selected partial decomposition algorithms on the medium corpus, $A^{315,002\times 61,293}$. Times are in minutes.

Algorithm	**Job size** c	**Time taken**
SVDLIBC	61,293	9.2
ZMS	200	360.1

(a) Serial algorithms, serial setup.

		No. of nodes p		
Job size c	serial	1	2	4
1,000	55.5	283.9	176.2	114.4
4,000	21.8	94.5	49.6	38.2
16,000	15.5	29.5	32.0	23.0

(b) distributed DLSA$_2$

For experiments on the medium corpus, I only included algorithms that ran reasonably fast during the accuracy assessment on the small corpus, that is, only ZMS, DLSA in its fastest variant DLSA$_2$ and SVDLIBC.

Performance of running the computation in distributed mode is summarized in Table 3.4. As expected, performance scales nearly linearly, the only overhead being sending and receiving jobs over the network and the final merges at the very end of the distributed algorithm. This overhead is only significant when dealing with a slow network and/or with extremely few updates. The faster the connecting network and the greater the number of updates, $\lceil \frac{n}{c} \rceil \gg p$, the more linear this algorithm becomes.

Another interesting observation comes from comparing DLSA2 speed in the "serial setup" (fast BLAS library) vs. "cluster setup" (no fast BLAS) in Table 3.4 (right). The serial setup is about five times faster than the corresponding cluster version with $p = 1$, so that even spreading the computation over four BLAS-less nodes results in *slower* execution than on the single "serial" node. As installing a fast, threaded BLAS library[9] is certainly cheaper than buying five times as many computers to get comparable performance, I strongly recommend doing the former (or both).

English Wikipedia results

Since the Wikipedia corpus is too large to fit in RAM, I only ran the streamed ZMS and $DLSA_2$ algorithms, asking for 400 factors in each case. On the "serial setup" described above, ZMS took 109 hours, $DLSA_2$ 8.5 hours. I would like to stress that these figures are achieved using a single commodity laptop, with a one-pass online algorithm on a corpus of 3.2 million documents, without any subsampling. In distributed mode with six nodes, the time of $DLSA_2$ drops to 2 hours 23 minutes[10]—by far the fastest result reported in literature.

3.5 LSA OVER THE ENGLISH WIKIPEDIA

The ability to process arbitrarily large corpora with the streamed, distributed, single-pass LSA algorithm is exciting and its performance on the English Wikipedia deserves a more thorough evaluation. The following experiments were reported in (Řehůřek, 2010) and extend the work of (Řehůřek, 2011b) presented above. Their goal is to track the effect of distributed computing, oversampling and memory trade-offs on the accuracy and performance of the large-scale eigen decomposition algorithms described in Sections 3.2, 3.3 and 3.3.1. They are the one-pass streamed distributed algorithm, a modified stochastic streamed

[9] Options include vendor specific libraries (e.g. Intel's MKL, Apple's vecLib, Sun's Sunperf), ATLAS, GotoBLAS, ...

[10] For details and reproducibility instructions, see the *gensim* package documentation.

two-pass algorithm and a hybrid of the two; the experiments refer to them as P_1, P_2 and P_{12}, respectively.

All three algorithms are streamed, meaning no random access to observations is required and their memory requirements are constant in the number of observations. The comparison will be done over a 100,000 × 3,199,665 sparse matrix with 0.5 billion non-zero entries (0.15% density). This matrix represents the entire English Wikipedia[11], with the vocabulary (number of features) clipped to the 100,000 most frequent word types[12]. In all experiments, the number of requested factors is set to $k = 400$. The experiments used three 2.0GHz Intel Xeon workstations with 4GB of RAM, connected by Ethernet on a single network segment. Compared to the experiments described above, all machines had an optimized BLAS library, ATLAS, installed[13].

3.5.1 *Oversampling*

This set of experiments examines the relative accuracy of the three algorithms. P2 has two parameters which affect accuracy: the oversampling factor l and the number of power iterations q. In the one-pass algorithms P1 and P12, accuracy is improved by asking for extra factors l during intermediate computations, to be truncated at the very end of the decomposition.

Figure 3.2 summarizes both the relative accuracy and runtime performance of the algorithms, for multiple choices of l and q. We see that although all methods are very accurate for the greatest factors, without oversampling the accuracy quickly degrades. This is especially true of the P2 algorithm, where no amount of oversampling helps and power iterations are definitely required.

Because the "ground-truth" decomposition is unknown, absolute errors are not available. However, according to preliminary experiments on a smaller corpus (not reported here), the stochastic algorithm with

[11] Static dump as downloaded from `http://download.wikimedia.org/enwiki/latest`, June 2010.

[12] The corpus preprocessing setup is described in more detail online.

[13] In fact, they were the same machines as in the previous distributed experiments, but I followed my own recommendation and carefully upgraded and tuned their BLAS stock.

Table 3.5.: The first ten topics coming from the P2 decomposition with three power iterations ($q = 3$) and 400 extra samples ($l = 400$). The top ten topics are apparently dominated by meta-topics of Wikipedia administration and by robots importing large databases of countries, films, sports, music etc.

Topic i	**Singular value s_i**	**The ten most salient words (both positively and negatively) for topic i, with their weights**
1.	201.118	-0.474*“delete" + -0.383*“deletion" + -0.275*“debate" + -0.223*“comments" + -0.220*“edits" + -0.213*“modify" + -0.208*“appropriate" + -0.194*“subsequent" + -0.155*“wp" + -0.117*“notability"
2.	143.479	0.340*“diff" + 0.325*“link" + 0.190*“image" + 0.179*“www" + 0.169*“user" + 0.157*“undo" + 0.154*“contribs" + -0.145*“delete" + 0.116*“album" + -0.111*“deletion"
3.	136.235	0.421*“diff" + 0.386*“link" + 0.195*“undo" + 0.182*“user" + -0.176*“image" + 0.174*“www" + 0.170*“contribs" + -0.111*“album" + 0.105*“added" + -0.101*“copyright"
4.	125.436	0.346*“image" + -0.246*“age" + -0.223*“median" + -0.208*“population" + 0.208*“copyright" + -0.200*“income" + 0.190*“fair" + -0.171*“census" + -0.168*“km" + -0.165*“households"
5.	117.243	0.317*“image" + -0.196*“players" + 0.190*“copyright" + 0.176*“median" + 0.174*“age" + 0.173*“fair" + 0.155*“income" + 0.144*“population" + -0.134*“football" + 0.129*“households"
6.	100.451	-0.504*“players" + -0.319*“football" + -0.284*“league" + -0.194*“footballers" + -0.141*“image" + -0.132*“season" + -0.117*“cup" + -0.113*“club" + -0.110*“baseball" + -0.103*“f"
7.	92.376	0.411*“album" + 0.275*“albums" + 0.217*“band" + 0.215*“song" + 0.184*“chart" + 0.164*“songs" + 0.160*“singles" + 0.149*“vocals" + 0.139*“guitar" + 0.129*“track"
8.	84.024	0.246*“wikipedia" + 0.183*“keep" + -0.179*“delete" + 0.167*“articles" + 0.153*“your" + 0.150*“my" + -0.141*“film" + 0.129*“we" + 0.123*“think" + 0.121*“user"
9.	79.548	*the word “category" in ten different languages (and their exotic un-TeX-able scripts)*
10.	79.074	-0.587*“film" + -0.459*“films" + 0.129*“album" + 0.127*“station" + -0.121*“television" + -0.119*“poster" + -0.112*“directed" + -0.109*“actors" + 0.095*“railway" + -0.085*“movie"

extra power iterations and oversampling gives the most accurate results; I will therefore plot it in all subsequent figures as a frame of reference, under "P_2, $l = 400$, $q = 3$". Note that all algorithms consistently err on the side of *underestimating* the magnitude of the singular values—as a rule of thumb, the greater the singular values in each plot, the more accurate the result.

3.5.2 *Chunk Size*

The one-pass algorithms P1 and P12 proceed in document chunks that fit into core memory. A natural question is, what effect does the size of these chunks have on performance and accuracy? With smaller chunks, the algorithm requires less memory; with larger chunks, it performs fewer merges, so we might expect better performance. This intuition is quantified in Figure 3.3 (left), which lists accuracy and performance results for chunk sizes of 10,000, 20,000 and 40,000 documents.

We see that chunk sizes in this range have little impact on accuracy, and that performance gradually improves with increasing chunk size. This speed-up is inversely proportional to the efficiency of the decomposition merge algorithm: with a hypothetical zero-cost merge algorithm, there would be no improvement at all, and runtime would be strictly dominated by costs of the in-core decompositions. On the other hand, a very costly merge routine would imply a linear relationship, doubling the runtime every time the number of input chunks doubles.

3.5.3 *Input Stream Order*

In the Wikipedia input stream, observations are presented in lexicographic order—an observation vector corresponding to the Wikipedia entry on *anarchy* arrives before the entry on *bible*, which comes before *censorship* etc. This order is of course far from random, so we are naturally interested in how it affects the resulting decomposition of the single-pass algorithms (the two-pass algorithm is order-agnostic by construction).

To test this, I randomly shuffled the input stream and re-ran the experiments on P1. Ideally, the results should be identical, no matter how we permute the input stream. Results in Figure 3.3 (right) reveal that this is not the case: singular values coming from the shuffled runs are distinct from the ones coming from the original, alphabetically ordered sequence. This shows that the one-pass truncated scheme has some difficulties adjusting to gradual subspace drift. With the shuffled input, no significant drift can occur thanks to the completely random observation order, and a much higher accuracy is retained even without oversampling.

3.5.4 *Distributed Computing*

The two single pass algorithms, P1 and P12, lend themselves to easy parallelization. In Figure 3.4, I evaluate them on a cluster of 1, 2 and 4 computing nodes. The scaling behaviour is linear in the number of machines, as there is virtually no communication going on except for dispatching the input data and collecting the results. As with chunk size, the choice of cluster size does not significantly affect accuracy.

The P2 algorithm can be distributed too, but is already dominated by the cost of accessing data in its $q + 2$ passes. Routing the data around the network gives no performance boost, so I omit the results from the figure. Nevertheless, distributing P2 would still make sense under the condition that the data be already predistributed to the computing nodes, perhaps by means of a distributed filesystem.

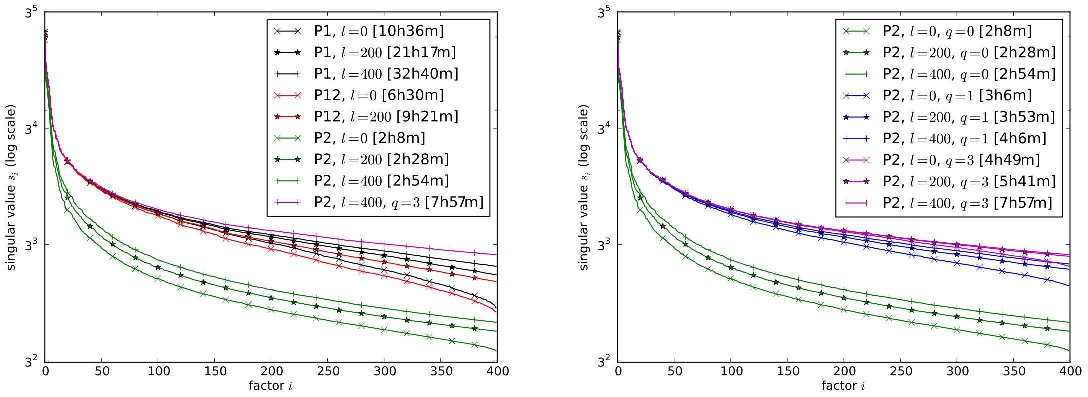


Figure 3.2.: Experiment 3.5.1: effects of the oversampling parameter l on accuracy of the P1, P2 and P12 algorithms (left) and oversampling and power iterations on the P2 algorithm (right). Wall-clock times are in brackets. Experiments were run on a single machine, with chunks of 20,000 documents.

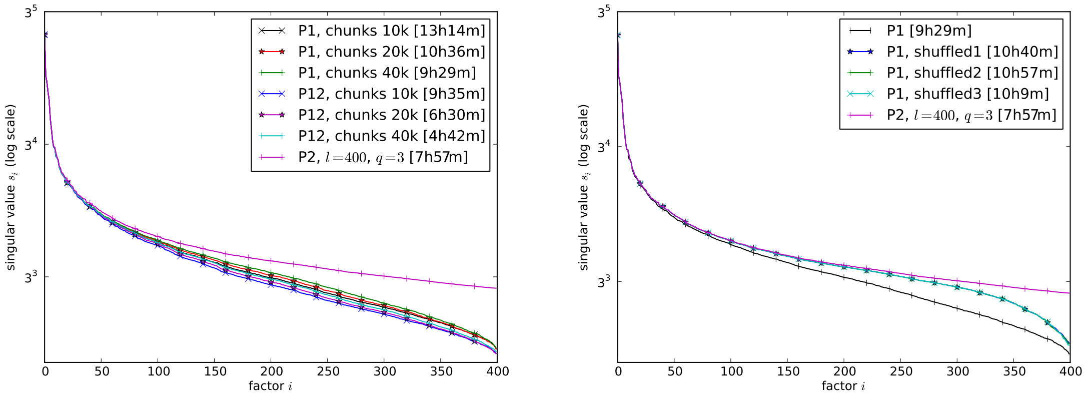


Figure 3.3.: **Left:** Accuracy and wall-clock times for different chunk sizes in P1 and P12 (Experiment 3.5.2), no oversampling.
Right: Effects of input order on the P1 algorithm (Experiment 3.5.3). Chunk size is set to 40,000 documents, no oversampling.

3.6 conclusions

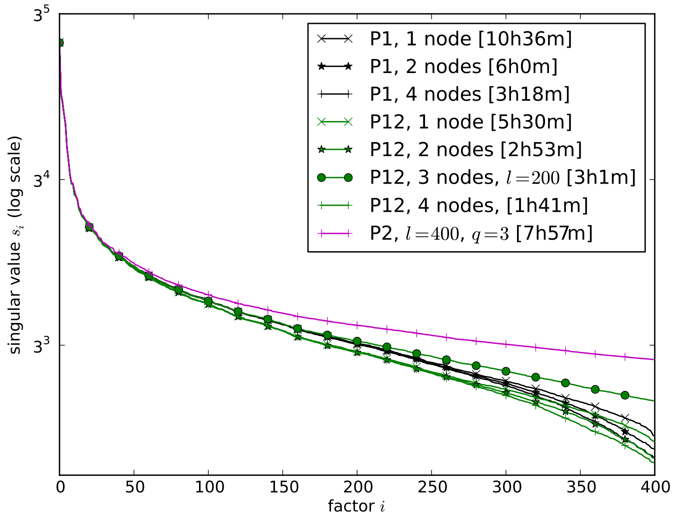


Figure 3.4.: Distributed computing for algorithms P1, P12 (Experiment 3.5.4). The chunk size is set to 20,000 documents, no oversampling.

In this chapter I compared several LSA inference methods that scale gracefully either due to their a) incremental nature (SVD can be updated with new observation without the need to recompute everything from scratch) or b) distributed nature, which allows them to utilize a cluster of computers to achieve fast decomposition in distributed memory.

I developed and presented a novel distributed single-pass eigen decomposition method, which runs in constant memory w.r.t. the number of observations. This method is suited for processing extremely large (possibly infinite) sparse matrices that arrive as a stream of observations, where each observation must be immediately processed and then discarded. The method is embarrassingly parallel, so I also implemented and evaluated its distributed version.

I also presented a novel streamed version of a two-pass stochastic eigen decomposition algorithm, and compared it to the single-pass algorithm. The comparison was done in the context of Latent Semantic Analysis, on a corpus of 3.2 million documents comprising the English Wikipedia.

On a single 2GHz machine, the top achieved decomposition times were 4 hours and 42 minutes for the one-pass algorithm and 3 hours 6 minutes for the stochastic multi-pass algorithm. Without power iterations and with reduced amount of oversampling, I recorded even lower times, but at the cost of a serious loss of accuracy. On a cluster of four computing nodes on three physical machines, the distributed single pass decomposition was completed in 1 hour and 41 minutes.

We observed that the lightning-fast stochastic algorithm suffers from serious accuracy issues, which can be remedied by increasing the number of passes over the input (power iterations), as suggested in (Halko et al., 2009). But, as the number of passes is the most precious resource in streaming environments, the otherwise slower one-pass algorithms become quickly competitive. The one-pass algorithms, one the other hand, suffer from dependency on the order of observations in the input stream; I will return to this behaviour in future work.

A practical and equally exciting contribution is a modern implementation of these algorithms, that I release into open-source as *gensim*. Written in Python, it still manages to get top performance thanks to the use of Python's NumPy library with fast BLAS calls under the hood.

4

LATENT DIRICHLET ALLOCATION

In the previous chapter, I presented research into scalability of Latent Semantic Analysis, a robust method of unsupervised document analysis based on linear algebra. This chapter presents a scalable version of *Latent Dirichlet Allocation (LDA)*, a method based on Bayesian inference. As with LSA, performance of the resulting distributed LDA algorithm will be measured against the English Wikipedia, a large, topically heterogeneous corpus.

4.1 MOTIVATION

In the introduction Chapter 2.5.5, I described the history and theoretical background of Latent Dirichlet Allocation. As a reminder, LDA is a hierarchical Bayesian model that represents each document as mixture of several topics, where the topics are in turn distributions over words. Each topic assigns high probability to a different set of semantically coherent words. As is the case with all models based on the statistical semantics hypothesis, what is "semantically coherent" depends on word co-occurence patterns within a given corpus of documents. This view of semantic relatedness is useful in word clustering as well as document similarity assessment (we expect documents with similar topic distributions to be semantically related, too), but has little connection to the traditional notion of semantics in linguistics or psychology (see the discussion in Chapter 1).

The associated inference algorithms of estimating an LDA model from a given training corpus are costly, and much research has been directed at improving their efficiency. There are three major branches of optimization:

- Approximative algorithms, which make the classic trade-off between model accuracy and performance.
- Distributed algorithms, which attack the problem by using more hardware (computer clusters).
- Online algorithms, which reduce complexity by processing the input incrementally, in smaller batches of documents.

The novel Latent Semantic Analysis algorithm of the previous chapter combined results in all three of these areas. The LDA algorithm described here uses an existing approximative algorithm (the mean-field variational algorithm of Blei et al. (2003)). I extend it to work over a cluster of autonomous computers, and over infinite input streams (updating an existing model with new documents). As for the third branch, online processing, my own work on an incremental updating LDA scheme has been obsoleted by Hoffman et al. (2010), who published a principled approach to online LDA training based on gradient ascent. The final algorithm described below will incorporate all three optimization areas.

4.2 PRIOR ART

While Latent Semantic Analysis has been successfully applied to collections of millions of documents (see Vigna (2008) and my own work in the previous chapter), experiments on LDA are usually done over corpora two orders of magnitude smaller. Among the exceptions are (Yao et al., 2009), who, by clever use of sparsity of the topic-word distributions, achieve a constant-factor speed up in their Gibbs sampler[1]. Another approach presented in Song et al. (2005) realizes online training through incremental collapsed Gibbs sampling, but in the process loses the guarantee of convergence.

Use of distributed algorithms was explored in Newman et al. (2007), where the authors devised two parallelized approximative algorithms.

[1] The authors also released their code through the software tool Mallet.

However, their algorithm is based on Gibbs sampling and its memory requirements scale linearly with the number of input documents (in fact, with the number of input words), which is infeasible for web-scale corpora. The cluster distribution there serves merely to speed up the computation, not overcome problems with memory scalability. Wang et al. (2009) implemented their algorithms to work over MPI and Google's MapReduce framework.

4.3 VARIATIONAL EM

The core inference algorithm is based on Variational Expectation-Maximization (sometimes also called Variational Bayes, VB) algorithm (Attias, 2000; Blei et al., 2003). This algorithm replaces the true posterior $P(\boldsymbol{\theta}, \boldsymbol{z}, \boldsymbol{\beta} | \boldsymbol{w}, \alpha, \eta)$ with a more simple, tractable distribution $Q(\boldsymbol{\theta}, \boldsymbol{z}, \boldsymbol{\beta} | \boldsymbol{\gamma}, \boldsymbol{\phi}, \boldsymbol{\lambda})$. $\boldsymbol{\gamma}, \boldsymbol{\phi}, \boldsymbol{\lambda}$ are free parameters, so that Q is in fact a family of distributions. We choose the form of Q so that dependencies between $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$ disappear, with Q factorizing to

$$\begin{aligned} Q(z_{i,j} = k) &= \phi_{j, w_{i,j}, k} \\ Q(\theta_d) &= \text{Dirichlet}(\gamma_d) \\ Q(\beta_k) &= \text{Dirichlet}(\lambda_k), \end{aligned}$$

that is, $\boldsymbol{\phi}$, $\boldsymbol{\gamma}$ and $\boldsymbol{\lambda}$ are parameters for the posterior distributions over $\boldsymbol{z}$, $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$, respectively.

It can be shown through use of Jensen's inequality[2] that minimizing the lower bound

$$\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\gamma}, \boldsymbol{\lambda}, \boldsymbol{w}) \triangleq \mathbb{E}_q[\log P(\boldsymbol{w}, \boldsymbol{z}, \boldsymbol{\theta}, \boldsymbol{\beta} | \alpha, \eta)] - \mathbb{E}_q[\log Q(\boldsymbol{z}, \boldsymbol{\theta}, \boldsymbol{\beta})] \tag{4.1}$$

with respect to the free parameters $\boldsymbol{\phi}, \boldsymbol{\gamma}, \boldsymbol{\lambda}$ results in minimizing the Kullback-Leibler divergence between P and Q, and, therefore, in Q becoming as close approximation of P as our choice of the family of Q permits.

This basic intuition leads to an *Expectation-Maximization (EM)* algorithm for training an LDA model:

[2] The derivation is rather involved; see the original article of Blei et al. (2003) for details.

E-STEP: Optimize $\boldsymbol{\phi}, \boldsymbol{\gamma}$ while holding $\boldsymbol{\lambda}$ (the parameters for topics) fixed.

M-STEP: Optimize $\boldsymbol{\lambda}$ using the newly computed $\boldsymbol{\phi}, \boldsymbol{\gamma}$.

The optimization is done by coordinate ascent over the parameter space of $\boldsymbol{\phi}, \boldsymbol{\gamma}, \boldsymbol{\lambda}$:

$$\phi_{d,w,k} \propto exp(\mathbb{E}_q[log\theta_{d,k}] + \mathbb{E}_q[log\beta_{k,w}])$$

$$\gamma_{d,k} = \alpha + \sum_{i=1}^{m} A_{i,d}\phi_{d,i,k}$$

$$\lambda_{k,i} = \eta + \sum_{d=1}^{n} A_{i,d}\phi_{d,i,k},$$

where $\mathbb{E}_q[log\theta_{d,k}] = \psi(\gamma_{d,k}) - \psi(\sum_{t=1}^{k} \gamma_{d,t})$, $\mathbb{E}_q[log\beta_{k,w}] = \psi(\lambda_{k,w}) - \psi(\sum_{i=1}^{m} \lambda_{k,i})$ and ψ is the digamma function.

4.3.1 *Streamed VEM*

The algorithm as formulated above is not suitable for large training corpora. The reason is that storage for the parameters $\boldsymbol{\gamma}^{n \times k}$ and $\boldsymbol{\phi}^{n \times m \times k}$ scales linearly with the input size n, which is not acceptable. Luckily, unlike Gibbs sampling, we can avoid storing these parameters explicitly in memory. The key observation here is that there is no coupling between parameters corresponding to different documents. Each update in the E step can be performed independently of the others; the documents are independent conditional on topics.

Algorithm 4.1 realizes this document-by-document training in constant memory (independent of n). In anticipation of distributing the E-step over several cluster nodes, we compute the E-step over a chunk of documents $\mathbf{C}$ (instead of the full matrix $\mathbf{A}$), and only collect sufficient statistics $\boldsymbol{\chi}$ instead of updating $\boldsymbol{\lambda}$ at the end of the algorithm. The reason is, $\boldsymbol{\lambda}$ will be updated based on $\boldsymbol{\chi}$ aggregated from all chunks, in a higher level routine.

Algorithm 4.1: LDA Inference (E step on a single document chunk)

Input: A chunk $\mathbf{C}_{m\times c} = \{d_1, d_2, \dots, d_c\}$ of c documents; c is chosen such that the whole chunk fits in main memory. $\alpha_{k\times 1}$ are the topic priors. $\boldsymbol{\beta}_{k\times m}$ are the current word-topic probabilities.

Output: Document-topic proportions $\theta_{k\times c}$. Sufficient statistics $\chi_{k\times m}$ for the M step.

```
$\chi_{k\times m} \leftarrow 0$
foreach document $d_i^{m\times 1}$ in C do
    $\theta_i^{k\times 1} \sim$ Dirichlet(Gamma(1,0.1)) ; // initialize θ randomly
    repeat
        $\boldsymbol{\phi}_{m\times k} \leftarrow \exp(\mathbb{E}_q[\log\theta_i] + \mathbb{E}_q[\log\boldsymbol{\beta}])$
        $\gamma_i \leftarrow \alpha + \boldsymbol{\phi}^T d_i$
        $\theta_i \leftarrow$ Dirichlet($\gamma_i$)
    until $\theta_i$ has converged
    $\chi \leftarrow \chi + \theta_i d_i^T$
end
```

4.3.2 *Distributed VEM*

The EM algorithm is straightforward to distribute; after aggregating the sufficient statistics χ from all document chunks in the E step, we set $\lambda = \eta + \sum \chi$ in the M step. In vanilla batch VB, this EM coordinate ascent is repeated until convergence (for example, until log-likelihood on a held-out corpus stops improving). However, this batch algorithm is not satisfactory for large corpora for multiple reasons:

1. The M step only happens once per full corpus iteration, after having processed all chunks. While this ensures convergence, we could potentially converge much faster with finer updates.

2. There is no way to update the model with new documents if we wish to run LDA as a daemon service. We can re-train the model on the whole dataset from scratch, but that is extremely inefficient (cf. the "batch" Lanczos methods for LSA from Chapter 3).

3. This approach breaks down completely for infinite training streams. Here we cannot afford multiple iterations over the training corpus even in principle.

4. Additionally, combining the previous two points, it is desirable to have a way to discount old observations in favour of new ones, so that the model can adjust to new data trends in a non-stationary training input stream[3].

Algorithm 4.2: LDA model blending

```
Input: Sufficient statistics matrices $\chi_1$ and $\chi_2$, created from chunks
       of $n_1$ and $n_2$ documents respectively. Blending factor $\rho$.
Output: $\chi$ which blends $\chi_1$ and $\chi_2$ proportionally to $\rho$.
$\overline{\chi}_1 \leftarrow \frac{n_1+n_2}{n_1}\chi_1$
$\overline{\chi}_2 \leftarrow \frac{n_1+n_2}{n_2}\chi_2$
$\chi \leftarrow (1-\rho)\overline{\chi}_1 + \rho\overline{\chi}_2$
```

Recently, Hoffman et al. (2010) addressed the first two points by introducing an *online* LDA inference algorithm, based on subsampling. While still limited to training over corpora of finite, predefined size (the final corpus size is an explicit parameter), their algorithm is elegant, is easy to extend and comes with convergence guarantees. The core idea is introducing a function that takes two existing models (for example, the current model and a model coming from a new chunk of documents) and *blends* them into a single resulting model. In this way, each online update can be seen as being *blended into* the existing model.

The novel algorithm presented below builds on this work, and extends it to cover the last two points as well: new observations may be added indefinitely (infinite training stream) and new observations may be given more (or less) weight during the training, to enable dynamic modelling of non-stationary input. Algorithm 4.2 describes one possibility of such modified model blending. The final model is a linear combination of expected counts χ of the two input models stretched to equal size.

[3] In the same fashion we introduced a decay factor γ for LSA subspace tracking in Chapter 3.

The final distributed algorithm is summed up in Algorithm 4.3. The outer-most loop acts as a daemon: it accepts new training corpus of documents and updates the underlying model. The update is done in the inner loop, by the online algorithm discussed above: new documents are split into chunks of manageable size that fit in main memory and distributed among the available processors.

Following the analysis from Hoffman et al. (2010), convergence is ensured by letting the impact of later updates gradually diminish by setting the blending factor $\rho_t = (1+t)^{-0.5}$. Where convergence is not desired (such as for non-stationary streams), a fixed number of updates in the inner loop can be run instead, blending the models in proportion to the number of documents they were created from.

This algorithm introduces another parameter κ, the purpose of which is to explicitly control the frequency of model updates. The model is updated (a complete EM step if performed) once per $\kappa \cdot P$ chunks, where P is the number of computing nodes in the cluster. With $\kappa = 1, P = 1$, the algorithm behaves exactly like the online stochastic algorithm of Hoffman et al. (2010). Setting $\kappa = 1, P > 1$ corresponds to a "distributed" version of their algorithm: the update happens after every processor has processed exactly one input chunk of documents. This is a very natural extension: in the same amount of time, we process P chunks in parallel on P processors. Increasing $\kappa > 1$ results in fewer updates; more chunks in the E step before performing an M step. On the other extreme, with $\kappa \geqslant \lceil \frac{n}{P} \rceil$, the algorithm falls back to batch mode, performing a complete corpus pass for a single EM update. Therefore, through the κ parameter, the granularity of updates can be naturally adjusted on a scale that includes one update per chunk on one end, and one update per entire corpus on the other. The M step is a natural synchronization point, where each processor sends out its accumulated state $\boldsymbol{\chi}$ and receives a new matrix of topics $\boldsymbol{\beta}$. Setting κ therefore amounts to balancing between communication costs (higher κ means fewer synchronizations) and faster convergence (lower κ).

Algorithm 4.3 requires storing km floats on lines 2 and 3. On line 8, we observe that the aggregation of $\boldsymbol{\chi}$ can be done cummulatively, as each partial $\boldsymbol{\chi}_i$ arrives, without storing all individual statistics in memory, so this line requires only additional km floats. On line 7, each chunk $\mathbf{C}$ is be realized in memory before being sent out to individual

Algorithm 4.3: Distributed Online LDA

Input: Stream (possibly infinite) of update matrices $\mathbf{A_i}$. Each update is represented as a stream of c_i document chunks $\mathbf{A} = \left[\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_{c_i}\right]$ (the number c_i is chosen so that each chunk fits into main memory). Number of requested topics k. Asymmetric document-topic priors $\alpha_{t\times 1}$, symmetric word-topic priors η. Cluster of P computing nodes. Online parameter κ.

Output: Word-topic probabilities $\boldsymbol{\beta}_{t\times m}$, which constitute a variational LDA model.

This algorithm is streamed and avoids materializing any $O(n)$ or $O(m^2)$ matrices in memory, so that it can process arbitrarily large input.

```
$\chi_{k\times m} \sim$ Gamma(1, 0.1) ; // initialize the model randomly
$\boldsymbol{\beta}_{k\times m} \sim$ Dirichlet($\chi$);
foreach update A do
    $t \leftarrow 0$;
    repeat
        Distribute current topics $\boldsymbol{\beta}$ to all $P$ processors;
        // E step: process $\kappa \cdot P$ document chunks of A in
           parallel on P cluster nodes, Algorithm 4.1
        Compute $\chi_i$ of each chunk $\mathbf{C}_{t\kappa P \bmod c}, \ldots, \mathbf{C}_{(t\kappa P+\kappa P-1) \bmod c}$;
        $\overline{\chi} \leftarrow \sum_{i=1}^{P\cdot\kappa} \chi_i$ ; // aggregate sufficient statistics
        // M step: update model using the collected
           sufficient statistics
        Blend $\chi$ with $\overline{\chi}$ using $\rho = (1+t)^{-0.5}$ ; // Algorithm 4.2
        $\boldsymbol{\beta} \sim$ Dirichlet($\eta + \chi$);
        $t \leftarrow t + 1$;
    until $\boldsymbol{\beta}$ stable (model has converged) ;
end
```

processors. The size of **C** is completely configurable; in memory-constrained environments, a “chunk” may even be a single document. The overall space complexity is therefore $O(km)$ floats, which is optimal, as that is the size of the model $\beta^{k \times m}$ itself. We do not store $\theta^{k \times n}$ (document-topic mixtures) during training, as the size of θ is $O(n)$. If required, θ of the training corpus can be reconstructed through the inference procedure in Alg. 4.1 with an extra pass over the corpus[4].

Time complexity is dominated by the E step, Algorithm 4.1. For each document n_j, the inner loop performs $O((|n_j| + 1)k)$ operations until convergence. (Blei et al., 2003) report that it took them about $|n_j|$ iterations to converge, so that the whole procedure requires $O(|n_j|^2 k)$ operations per document. In the online updating scheme ($\kappa \ll \frac{n}{p}$), we can afford to set a hard upper limit on the number of variational iterations per document, because the impact of each document on the model is much more immediate[5]. Algorithm 4.1 then therefore runs in $O(|n_j|k)$ per training document, or $O(\text{nnz} \cdot k)$ for the entire training corpus, nnz being the number of non-zeroes in the sparse input bag-of-words matrix $\mathbf{A}^{m \times n}$.

4.4 WIKIPEDIA EXPERIMENTS

Equipped with an online, streamed, distributed algorithm for LDA model estimation, we can proceed to build models from large, web-scale corpora. As with LSA, our corpus of choice will be the entire English Wikipedia[6], a collection of 3.2 million documents. Its bow-of-words represenation is a 100,000 × 3,199,665 sparse matrix with 0.5 billion non-zero entries (0.15% density). The matrix is implicit; its columns (the documents) are accessed sequentially one after another, and therefore the matrix never needs to be fully realized in memory. The vocabulary was clipped to the 100,000 most frequent word types, after discarding word types that appear in more than 10% documents (stop-list). In all experiments, the number of requested topics is set to $k = 100$. The hyperparameters α and η were both set to $\frac{1}{k} = 0.01$.

[4] This is a complete analogy to the way we only store the left singular vectors **U** in LSA, ignoring the $O(n)$ matrix **V** of right singular values; see remark 2 on page 40.

[5] In practise and in the Wikipedia experiments below, I limited the number of per-document iterations to fifty.

[6] Static dump as downloaded from `http://download.wikimedia.org/enwiki/latest`, June 2010.

Table 4.1.: LDA topic assignments in a sample document. Word colours correspond to values of the hidden topic-indicator variable z—five different topics appear in the text. No colour means the word was removed during pre-processing and therefore did not affect the computation. This excerpt comes from the Wikipedia article on "Military Medical Business" (retrieved February 2011), topics are automatically inferred using the 100-topic model of the whole English Wikipedia described in the text.

Military Medical Business was one of the professional medical journals of the Worker's and Peasant's Red Army (RKKA). The journal was published monthly in Moscow. Between 1938–1940, each issue cost 1 ruble and 50 kopecks. A 12 month subscription to the journal for the year 1939 cost 18 rubles. The Moscow publishing house published total of 33 different professional medical journals for the RKKA in the year 1939. The Leningrad publishing house published 5 different journals that same year. Four different serials were also published by the Moscow publishing house under the title Central Medical Abstract Journal.[1] By the end of 1940, the price of single issue of Military Medical Business had increased to 2 rubles. The Moscow publishing house advertised 33 different journals, the Leningrad publishing house advertised 5 journals, and the Kazan publishing house advertised 1 journal for the year 1941. A 12 month subscription to Military Medical Business cost 30 rubles in 1941. The Moscow publishing house also advertised four serials under the title Central Medical Abstract journal for the year 1941.[2]

The first experiment was run with $\kappa = 1$, i.e., updating the model once every P document chunks. When run with $P = 1$ (a single Core2Duo laptop, 2.53GHz, 4GB RAM), Algorithm 4.3 took 11 hours to estimate this Wikipedia model. With $P = 4$ (four 2.0GHz Intel Xeon workstations with 4GB of RAM, connected by Ethernet on a single network segment), time taken dropped to 3h20m. A few of the resulting topics are listed in Table 4.2. Unlike Latent Semantic Analysis, these topics are not cumulative and therefore easier to interpret (cf. Table 3.5 on page 61). As was the case with LSA, some of these LDA topics correspond to Wikipedia management concepts and templates.

Table 4.2.: Ten most probable words for sixteen randomly selected topics coming from a 100-topic LDA model of the English Wikipedia.

disease	king	film	user	air	light	radio	image
medical	ii	films	edit	aircraft	star	station	jpg
treatment	prince	directed	block	airport	sun	fm	file
cells	emperor	movie	wikipedia	force	moon	news	uploaded
cell	son	cast	here	squadron	earth	channel	png
health	duke	director	blocked	flight	stars	am	media
cancer	iii	man	edits	wing	planet	stations	contribs
blood	empire	story	my	aviation	camera	tv	notify
patients	kingdom	plot	revert	flying	sky	broadcasting	logs
acid	royal	love	me	airlines	solar	pm	picture
bar	olympic	president	business	h	class	canada	building
text	olympics	washington	companies	g	assessed	canadian	built
till	women	served	million	o	rev	ontario	historic
shift	men	virginia	bank	j	quality	toronto	buildings
color	summer	pennsylvania	market	r	importance	quebec	tower
fontsize	championships	massachusetts	management	f	rating	columbia	hall
xs	gold	ohio	financial	l	unknown	british	architecture
id	medal	governor	services	k	reassessed	alberta	construction
y	games	republican	industry	p	stub	montreal	street
value	event	law	development	w	start	vancouver	park

4.5 DISCUSSION

A scalable, distributed algorithm for Latent Dirichlet Allocation inference has been lacking. Large-scale LDA efforts, when done at all, are hand-tuned, monumental, and cite-worthy on that merit alone[7]. At the same time, the general NLP public is becoming increasingly interested in modern trends in topic modelling, and unsupervised text exploration and analysis have a potential on impact communities well outside the academic core. One example are digital libraries, where organizing and cataloguing the (often retro-digitzed and metadata-free) texts by hand is tedious and costly. In Chapter 6, I will describe one such application of LDA on an existing digital library.

The algorithm presented in this chapter combines several desirable properties:

1. it is distributed (scales linearly with more machines),

[7] `http://christo.cs.umass.edu/wiki40/`

2. it is streamed (runs in contant memory and does not require random access to the training dataset) and
3. is incremental (can update an existing model with new training data, so that it can process infinite input streams).

To the best of my knowledge, mine is the first algorithm with these properties combined. The algorithm does not introduce any further approximations or heuristics, beyond the variational mean-field approximation in document inference.

As with LSA, I release the distributed implementation into open-source as part of *gensim*, a Python framework for vector space modelling. As a side effect of this release, the results presented in the experiment section are easily reproduced[8], as they only rely on a publicly available dataset (the English Wikipedia) and the gensim code.

[8] `http://nlp.fi.muni.cz/projekty/gensim/wiki.html`

Part II

APPLICATIONS

Ever tried. Ever failed.
No matter. Try again.
Fail again. Fail better.
(Samuel Beckett, *Worstward Ho*)

Even though this thesis concerns itself with "text semantics" of a limited type (see the introduction in Chapter 1), there is a wealth of applications that call for exactly this sort of simplistic, utility-driven statistical approach.

As an example, Chapter 7 will describe an algorithm for language identification. The language identification task has been traditionally formulated as "Given a chunk of text, which language was it written in?". The field was considered "solved" by the mid-90s, and only with the rise of the Internet did new challenges appear. What about documents written in multiple languages? Can the methods tell the input document was "written" in no natural language at all (the messiness of the Web data can only be appreciated by those who have had to deal with it)? How robust are the methods on very short text segments? Can they tell the boundary points where the text changes from one language to another in a document? Are they fast enough?

This "problem redefinition" is typical of modern applications—rather than invalidating old solutions, the problem formulation shifts in new, equally exciting directions. Following the silver thread running throughout this thesis, particular emphasis will be placed on efficiency and scalability of the proposed methods.

I place these applied algorithms here, after the part on general-purpose semantic analysis techniques of LSA and LDA, even though these applications at the same time *motivate* the general algorithms and *build on top of them*. The work on LSA and LDA scalability was fuelled by the practical need of topical browsing in an existing digital library (Chapter 6); the language segmentation algorithm in Chapter 7 was developed for Seznam.cz, a search engine company; the topical segmentation of Chapter 5 is a result of having to deal with heterogeneous text collections in a prototype of a plagiarism detection system (Řehůřek, 2008b).

Of course, the list of applications considered in the subsequent chapters is by no means exhaustive. Other uses of the unsupervised Vector

Space Model include, and are not limited to, essay grading (Landauer et al., 1998), authorship attribution (Juola, 2006; Abbasi and Chen, 2006), thesaurus generation (Grefenstette, 1994; Curran and Moens, 2002) or word sense disambiguation (Leacock and Chodorow, 1998; Pantel and Lin, 2002; Yuret and Yatbaz, 2010).

5

TOPIC SEGMENTATION

Topic segmentation algorithms aim to split an unstructured text document into contiguous blocks, called *segments*, each of which covers a single compact topic, while consecutive blocks cover different topics. Applications include finding topic boundaries in text transcriptions of audio news (Reynar, 1998), Information Retrieval (Hearst and Plaunt, 1993), anaphora resolution (Kozima, 1993), improving text navigation for visually impaired (Choi, 2002) or intrinsic plagiarism (*anomaly*) detection (Řehůřek, 2008a). It can also be used to improve Information Retrieval performance, which is main target application for the method described in this Chapter.

The content of this chapter is based on the work presented at the *13th Portuguese Conference on Artificial Intelligence (EPIA)*, (Řehůřek, 2007b). I start by observing which features are desirable in text segmentation algorithms for the Information Retrieval domain, with emphasis on improving high similarity search of heterogeneous texts. I proceed to describe a robust, purely statistical method based on context overlap that aims to fulfil these desired features. Experimental results are presented, along with a comparison to other existing algorithms.

5.1 MOTIVATION

To see how topic segmentation might improve IR performance, consider a standard IR scenario. Here documents are transformed via the Vector Space Model and indexing techniques are employed to allow efficient exact and proximity queries. Given the widely heterogeneous documents that a general IR system may expect, some of these documents may be monothematic and compact, dealing with a single topic. Others can be a mixture of various topics, connected not thematically but rather incidentally (for example, documents containing news agglomerated by date, not topic). Some may cover multiple topics intentionally, such as complex documents involving passages in different languages. The problem here is that once the documents are converted into the Vector Space Model, all structural information is lost. The resulting document vector shifts away from any one topic included in the original document. Still, user queries are typically monothematic and in this way the chance of high similarity match between user query and document vector decreases. This can result in missed hit. Thus having basic retrieval blocks correspond to single topics rather than whole documents seems like a methodologically sound step. It is up to final application to merge and present topical, sub-document hits to the user. It also depends on application to set granularity of topics that we wish to tell apart. Identifying compact document chunks also has applications in intrinsic plagiarism detection, where it helps to reduce the number of suspicious passages and subsequent expensive, more detailed queries.

There are practical considerations that are important in real-world IR systems. Driven by the need to understand a system's behaviour (especially unexpected behaviour) and the ability to make extensions to the system during the development cycle, it is advantageous to keep the system architecture as simple, clear and robust as possible. Based on these concerns, three important properties of text segmentation for IR systems may be identified:

1. Domain independence. As little external knowledge as possible is required for segmentation of a document. For large collections, even semi-automatic techniques (i.e., techniques that require some kind of human intervention during the segmentation process) are problematic.

2. Language independence. Although techniques for automatic language detection exist (see Chapter 7), using separate topic segmentation algorithms for different types of input data is cumbersome and difficult to maintain. An algorithm should ideally work based solely on document content, without any extra meta-data. Additionally, it should be able to deal with the case where the input

text is not a strictly well-behaved part of a language (which is rare in real-world documents). An example is robustness towards commonly appearing inserted content such as small ASCII drawings, inserted passages in different languages, short tables, poorly OCR-ed documents and so on.

3. Granularity. Desirable is an option that allows one to set a customizable level of granularity as to what constitutes a "sufficient" topic shift. This allows the system maintainer to set the segmentation policy, based on expected query granularity. There is no point clogging the system with many small segments that are conceptually identical with respect to user interests, just as it's not desirable to keep widely varying topics in one single document.

In addition to this list, technical aspects such as effectiveness are also important. Segmentation must perform well enough to allow text segmentation of large documents in reasonable time. This limits the choices to algorithms that utilize readily identifiable surface cues, such as those based on text cohesion and lexical repetition.

5.2 RELATED WORK

Different text segments should more or less correspond to different topics. Paragraph breaks could be suggestive of a topic shift. However, simply splitting at new line breaks is problematic. In many documents the new line breaks are not indicative of paragraphs, but rather of the place where the text meets the right margin of the page. This may be caused by OCR, certain text editors or retransmission through e-mail clients. More sophisticated segmentation algorithms which take into account the document content are required.

An early linear segmentation algorithm called TextTiling is due to Hearst (Hearst, 1994). The basic idea is to take note of lexical repetition. A window of fixed length is being gradually slid through the text, and information about word overlap between the left and right part of the window is converted into a digital signal. Shape of the post-processed signal is used to determine segment breaks. Complexity of this method is linear in the length of the document and requires virtually no linguistic knowledge or tuning beyond the choice of parameters for window size, step for the window slide and a segmentation threshold[1]. These features make TextTiling a good candidate for IR purposes.

A more recent approach is that of Choi (Choi, 2000). His *C99* algorithm computes a sentence-to-sentence similarity matrix (again using lexical cohesion), and applies a relative ranking to get rid of unreliable absolute values. Divisive clustering on the rank matrix is applied to obtain the final segmentation. Choi's results show that C99 is vastly superior to the previous methods of Hearst, various modifications of Reynar's maximization algorithm (Reynar, 1998) and the Segmenter algorithm (Kan et al., 1998).

5.3 LSITILING

The algorithm proposed here, called *LSITiling*, is based on and extends Hearst's TextTiling algorithm. It uses the same sliding window to obtain a context similarity signal, then analyses this signal to arrive at a final segmentation. The difference lies in what constitutes a context. Where TextTiling relies on text surface features of character token units to determine context overlap, LSITiling uses context overlap in a conceptual space. The reasoning behind this enhancement is following: segmentation aims at discerning individual topics. These may be vaguely defined as chunks of text pertaining to a single idea. People, however, commonly use different words to express the same idea. Indeed, it is a recommended stylistic guideline to vary the vocabulary and avoid lexical repetition. TextTiling (as well as other algorithms) go a step in the direction of lexical independence by stemming the text—that is, unifying tokens that share a common word stem. Apart from failing point two of our three point wish-list (language independence), this does not address basic common language phenomena like synonymy.

LSITiling computes the context similarities in *topic space*. Topic space is a space where each dimension represents a concept present in original data. From a broader perspective, LSITiling belongs to the class of Vector Space Model methods. Construction of the concept space (also called *embedded* or *latent* space) is done by applying the IR technique of *Latent*

[1] See (Hearst, 1994) for exhaustive algorithm details.

Semantic Analysis (or sometimes *Indexing*) *(LSA, LSI)*, which was conceptually introduced in Section 2.5.1 and extended in Chapter refchap:lsa. LSI was chosen for its established role in IR, but other decomposition schemes discussed in Chapter 2 are also applicable. As a reminder, LSA is a straightforward statistical method which projects data onto a lower dimensional space, with the projection optimal in the least squares sense. Translated into the Natural Language Processing domain, it uses word co-occurrence of both first and higher orders to derive co-occurrence patterns, called *concepts*, which define the dimensions in the new concept space.

In the proposed algorithm, LSI is applied to the domain of text segmentation as follows: let the input be a single text document we wish to segment. Next we construct a corpus of *pseudo-documents*. This is a critical step—although previous work (Choi et al., 2001; Foltz, 1996; Bestgen, 2006) also sought to apply LSI to text segmentation, they split the input document into individual sentences and/or paragraphs and used those as a collection of pseudo-documents. Additionally, Foltz added other domain-specific documents, such as encyclopaedia entries and book excerpts to this collection, to increase quality of the latent model. Here the approach differs in that I consider *overlapping* pieces (or chunks) of the input document. In practice, I choose as pseudo-documents exactly those chunks considered by the TextTiling algorithm as contexts. All terms are identified (no stemming is performed) and *Term Frequency * Inverse Document Frequency*, or *TF-IDF*, matrix is constructed (see Section 2.2). The result is a large, sparse matrix—a perfect candidate for the efficient SVD algorithms discussed in Chapter 3. A number of latent dimensions is chosen and each pseudo-document is projected into the new, reduced latent space.

Dimensionality of the latent space determines how much data variance is retained. Using only the single, most dominant concept would result in each pseudo-document being represented by a single number. Keeping more dimensions keeps more information about the pseudo-document's content, allowing more refined representation in terms of adherence to multiple concepts. As the other extreme, keeping all dimensions means all information present in the data is kept (the TF-IDF matrix can be reconstructed without any loss). However, it is desirable to keep the number of latent dimensions low. This is to reduce data noise—dimensions with low significance correspond to noise and thus affect performance negatively. Also, for the sake of topic segmentation, we are only interested in the salient document concepts. Here the proposed approach also differs from the previous art, in that I explore effects of very low dimensionality, as opposed to Choi (Choi et al., 2001), where the golden IR standard of hundreds of dimensions was used. An interesting algorithmic feature of LSI is that concept spaces are nested: once we have obtained a document representation in the 100 most significant dimensions, we may work with the document in a ten dimensional space by simply omitting all but its first ten dimensions. This allows us to produce and store a single latent model, and tune concept granularity by only adjusting the latent space dimensionality.

Once the new context representations are obtained, context similarity is computed by the standard cosine similarity measure (see also Section 2.3):

$$\text{cossim}(v_i, v_{i+1}) = \frac{v_i^T \cdot v_{i+1}}{|v_i||v_{i+1}|}.$$

By computing cosine similarity between the consecutive pseudo-document vectors we obtain a one-dimensional digital time series, which is then smoothed and analyzed for local minima. Figure 5.1 shows a side by side comparison of LSITiling and TextTiling on that same input document; it is apparent the two signals have very different signal variance. With LSITiling, most of the signal for overlapping pseudo-documents runs at a near exact-match (cosine similarity score of 1.0), with occasional drops near the suspected segment boundaries. This allows us to reliably set the segmentation threshold at a high number (such as 1.0), as opposed to TextTiling, where the threshold is set according to a particular combination of signal mean and variance.

To test applicability of this new algorithm on more general data, and to establish a way to automatically determine its internal parameters, a series of experiments were conducted.

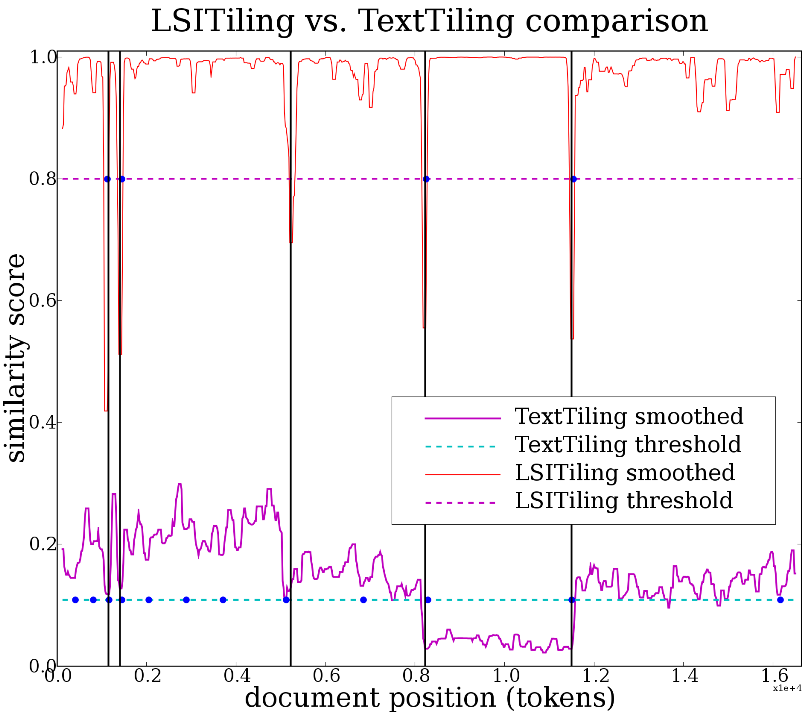


Figure 5.1.: A comparison of smoothed LSITiling and TextTiling signals on a sample input text of 16,634 tokens. The TextTiling threshold is set to signal mean minus half its standard deviation (as proposed by Hearst). The black vertical lines denote the *true* segment boundaries. Compare these to the blue dots along threshold lines, which denote segment boundaries *proposed* by the respective algorithms. In this example, LSITiling missed one segment boundary. TextTiling missed no boundary but produced seven additional, erroneous boundaries.

5.4 EXPERIMENTS

5.4.1 *Evaluation Methodology*

Given an input document and a segmentation proposed by a segmentation algorithm, we seek to numerically capture the quality (or "closeness") of this segmentation, with respect to a true segmentation (the "ground-truth"). This is a common concept in IR evaluation. However, the traditional IR evaluation measures of precision and recall are not suitable, because they penalize near-misses too much. We would like to penalize errors more "softly", or gradually, depending on the distance between the true and the proposed segment boundary. For this reason, I chose the *WindowDiff metric* proposed in (Pevzner and Hearst, 2002). The idea is to run a window of fixed length through the input text, taking note of how many times the number of proposed and real segment boundaries within the window disagree. The length of this window is set to one half of the average true segment length. Resulting error value is scaled to lie between 0.0 and 1.0 (0.0 being the best and 1.0 the worst score). For a discussion of WindowDiff's advantages over the previously used P_k probability metric, as well as its theoretical justification, see (Pevzner and Hearst, 2002).

WindowDiff

5.4.2 *Datasets*

Evaluating segmentation performance on real documents is problematic. This stems from the difficulty of obtaining a referential ("true") segmentation. Indeed, given two human judges, three proposed "true" segmentations are likely to appear. One way to overcome this difficulty is to combine results of multiple human judges. Another way is to create an artificial document by merging several smaller texts that deal with different topics. This way, although the resulting document is artificial, we can evaluate segmentation quality more precisely, because the segment boundaries are known exactly. Following other work in the field of topic segmentation, I adopted the latter approach for the following two sets of experiments.

In the first set, a collection of 31 text passages was assembled. These were picked to represent various topics and styles. 24 of these passages were written in English (both by native and non-native speakers), two in German, one in Czech and one in Dutch. Also present were text passages of unidentifiable language, such as an ASCII art picture or text table drawings. These 31 passages thematically covered law, biology, computer science, art, religion and politics. The texts took many different forms, including book excerpts, scientific papers, paper abstracts, text questionnaire, executive summary, part of a stage play, an interview,

Table 5.1.: Corpus statistics for the first corpus (segments coming from a pool of 31 documents) and the second corpus (segments from the BNC). Lengths are given in tokens, with the standard deviation in brackets.

corpus	#docs	segments per document	avg. document length	avg. segment length
corpus #1	525	1–14	51,549 (32,883)	7,003 (5,711)
corpus #2	700	10	1,766 (474)	177 (79)

product overview or an end user license agreement (EULA). Two of the passages come from poor book OCR, resulting in much garbled content. All passages were chosen with requirements that

- they deal with mutually exclusive topics, so that no matter how we concatenate them, the true segment boundaries are exactly at the points of concatenation,
- and that they are real-world documents, the concatenation of which may conceivably appear in a digital collection.

Apart from the lack of pair-wise semantic overlap between passages, no other constraints were placed and no text postprocessing was done. Passages had widely varying lengths and formatting, mimicking the requirements placed on segmentation algorithms mentioned in Section 5.1. From these passages, a corpus of 525 documents was assembled. Each document was created by merging one to fourteen passages, which were picked randomly without repetition. Each test document is thus made up from one to fourteen true segments, and the segmentation algorithm then tries to retrieve these segments given their concatenation as its only input. Average length is 7,003 tokens per segment and 51,549 tokens per document. These corpus statistics are also summarized in Table 5.1 under "corpus #1". Notable is the very high passage length deviation, implying text segments of very different lengths.

5.4.3 *Algorithm Comparison*

I will consider six algorithms in the comparison:

HEARST C implementation of the TextTiling algorithm from (Hearst, 1994), with default parameters[2].

TEXTTILING My own, slightly modified Python implementation of the same algorithm. The modification is in the way final boundaries are determined and was motivated by the abysmal performance of the original implementation. Rather than selecting boundaries based on signal mean and variance, a fixed threshold of 0.2 is set. The value of 0.2 was selected for its good performance in preliminary experiments. It was not tuned for this dataset, and thus may not be optimal.

C99 Choi's Java implementation of his C99 algorithm, with default parameters[3].

LSITILING Algorithm described in this chapter, with default parameters. These exactly equal the TextTiling parameters, with the dimensionality of the latent space set to 5 and the gap threshold equal to 1.0. Again, these parameters were predetermined without looking at the corpus.

FIXED An algorithm which inserts a segment boundary after each block of tokens, irrespective of document content. The block size equals 120 tokens by default.

RANDOM A referential algorithm which splits document randomly. The number of segments is selected randomly to lie between one and one plus a thousandth of number of document tokens.

Overall accuracy results on the first corpus are summarized in Table 5.2 and Figure 5.2 (left). Figure 5.2 (right) lists average performance over documents which share the same number of true segments. LSITiling consistently outperforms all other algorithms, with the exception of TextTiling for less than 3 true segments. The poor performance of the original TextTiling algorithm (called Hearst here) is due to the way it

[2] As obtained from `http://elib.cs.berkeley.edu/src/texttiles`, March 2007.

[3] Downloaded from `http://www.lingware.co.uk/homepage/freddy.choi/software/software.htm`, March 2007.

Table 5.2.: Experiment results for the first corpus. Numbers show statistical *WindowDiff* characteristics aggregated across all documents.

document scores	LSITiling	TextTiling	C99	Random	Hearst	Fixed
WindowDiff mean	0.345	0.374	0.566	0.567	0.950	0.995
WindowDiff stddev	0.186	0.164	0.217	0.275	0.177	0.023

sets its threshold. Upon inspection, it turns out that the segmentation boundaries are being inserted after every couple of sentences, and the *WindowDiff* metric marks this nearly as bad as the Fixed algorithm. The modified TextTiling version with fixed threshold performs much better and, in fact, outperforms C99. Poor performance of C99 (which is on par with the Random algorithm here) is surprising. I suspected that a possible reason for such discrepancy between results reported in (Choi, 2000) and my results may lie either in the different metric used (*WindowDiff* vs. *Error Rate*) or in the fundamentally different datasets used. To clarify this issue, I ran another set of experiments, on a different corpus more closely mimicking the setup of (Choi, 2000).

Table 5.3.: Experiment results for the second corpus

document scores	C99	LSITiling	TextTiling	Random	Fixed	Hearst
WindowDiff mean	0.175	0.345	0.372	0.378	0.388	0.442
WindowDiff stddev	0.083	0.062	0.038	0.035	0.098	0.092

The second corpus was created according to Choi's description, with the exception that the British National Corpus (BNC) was used instead of the Brown corpus. 700 documents were constructed by merging prefixes of ten random BNC documents. The number of true segments was thus fixed to 10, as opposed to 1–14 range from the first experiment. A "document prefix" here simply denotes the first few sentences extracted from a particular document. The length of the prefix was selected randomly between 3 and 11 sentences for 400 documents, between 3–5 sentences for 100 documents, 6–8 sentences for another 100 documents and finally 9–11 sentences for the remaining 100 documents. This is the exact

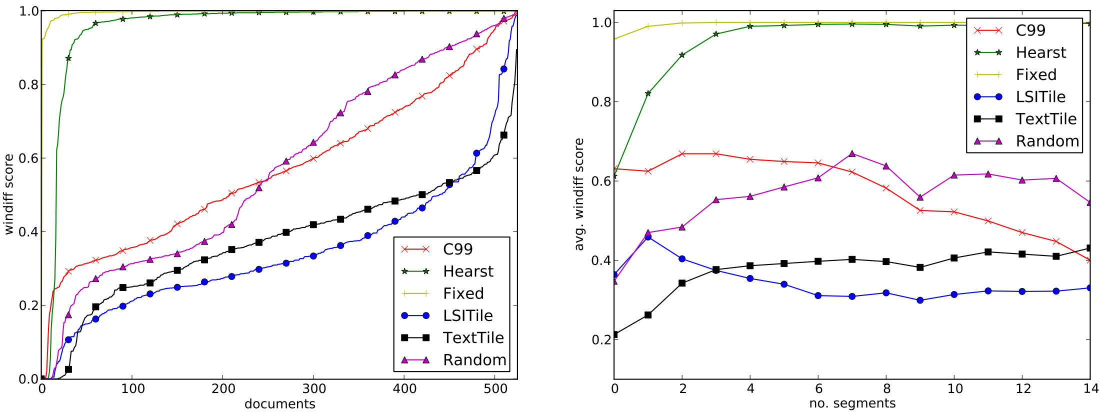


Figure 5.2.: **Left:** *WindowDiff* results for the first corpus. **Right:** For each number of segments 1–14, scores for all documents in the first corpus with exactly that number of true segments were averaged. This figure compares how different algorithms cope with increasing topical fragmentation of input texts.

same configuration as used in (Choi, 2000). As a general remark, these settings more closely follow the news feed segmentation scenario, with rather short true segments and smaller variation in true segment lengths. Data statistics are summarized in Table 5.1 under "corpus #2". For this dataset, C99 clearly outperforms all other algorithms by a large margin (Table 5.3 and Figure 5.3). It would appear that C99 is well suited for this particular problem, and not so well suited for the IR scenario with widely varying segments of considerable size. On this second corpus, although it lagged far behind C99, LSITiling (still with the default parameters) outperformed all other algorithms.

5.4.4 *Tuning the Parameters*

The choice of value for the latent space dimensionality parameter is not straightforward. The next series of tests therefore sought to establish a relationship between the data and this parameter. Ideally, we would like

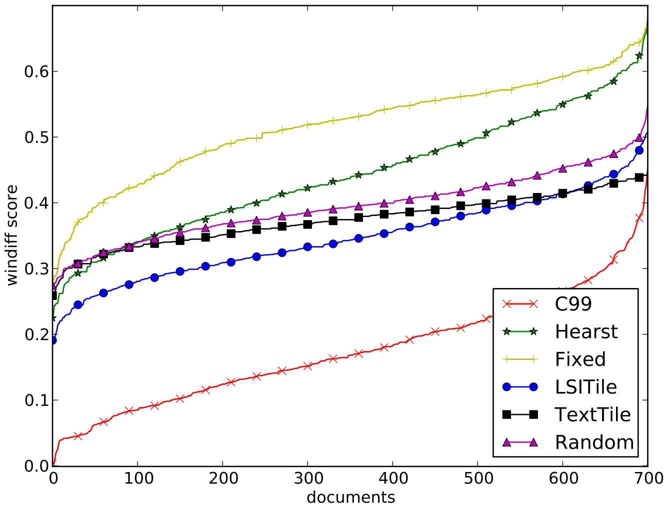


Figure 5.3.: Experiment results for the second corpus

to make setting of this parameter transparent to the user, so that LSITiling requires no additional parameters to these of TextTiling. Remember that in the previous experiments, the semantic space dimensionality value was set to be 5 for all documents. To establish relationship between the number of segments and the dimensionality (also referred to as kDim), I evaluated LSITiling again for each document of the first corpus, with kDim values between 1 and 100. As an example, Fig. 5.4 (left) shows average scores for all documents that have 3, 5 or 10 true segments. For these documents I force LSITiling to return respectively 3, 5 or 10 proposed segments as well, by splitting the input document at the 3, 5 or 10 gaps with most dissimilar left and right contexts. From this graph we may observe that the optimal kDim value appears to equal the true number of segments. Going through the same procedure for all other segments number (recall that the first corpus has between one and fourteen true segments per document, unlike the second where the number of referential segments is always ten) and plotting the results we obtain Fig. 5.5. The plots denote all kDim values for which the score was within one standard deviation from the optimal value, with plot

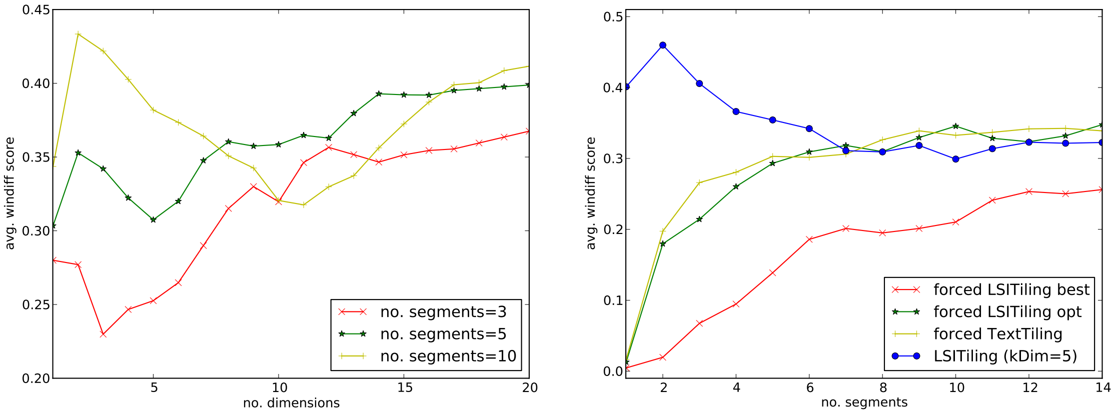


Figure 5.4.: **Left:** Average kDim scores for documents with selected number of true segments. Scores for kDim values above 20 did not improve and were clipped from the figure to improve readability. **Right:** Comparison of kDim selection strategies.

intensity decreasing as distance to optimal increases (solid black dots denote the best scores). From this graph we may observe three prevailing trends:

1. The horizontal line at true segments equal to 1 is trivial—it simply shows that for one true segment, with LSITiling forced to return one true segment, the segmentation is (trivially) perfect.

2. The second trend is diagonal—it shows that the correlation between number of segments and kDim may in fact be as simple as linear.

3. The third trend is the vertical line at small kDim. This is not trivial and shows that basing segmentation on a very small, fixed number of latent dimensions may be justified. This is in contrast to (Choi et al., 2001), where the authors use hundreds of latent dimensions in their proposed LSI algorithm.

Figure 5.4 (right) summarizes these results with respect to the number of document segments. It plots the original LSITiling result (kDim equal

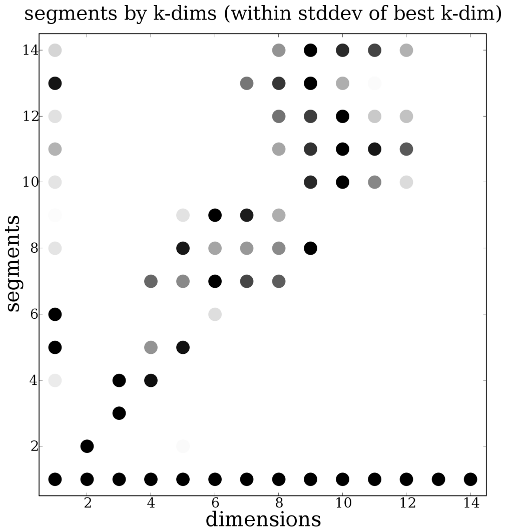


Figure 5.5.: Best kDim scores per number of segments. See text for explanation.

to 5) against results obtained by fixing the number of hypothesized segments to the true number of segments. *LSITiling best* denotes the results obtained by "magically" picking the best kDim for each number of segments. It is included for illustrative purpose only, because in practice we obviously do not have a referential segmentation to determine what the best kDim is. *LSITiling opt* plots the results obtained by setting kDim to the number of true segments (the diagonal trend). Although not straightforward, it is feasible to determine the number of segments for new, unseen texts[4]. Also included in the graph for reference are the TextTiling scores, where the algorithm is forced to return segmentation with a fixed number of segments in the same way LSITiling was. We can see that for low number of true segments, all forced algorithms perform better than the default LSITiling. This means that the default LSITiling overestimates the number of true segments here. However, for higher number of segments, the default LSITiling performs comparably to *LSITiling opt* and forcing it to return a fixed number of segments offers on average no improvement.

[4] Clusterization techniques (such as the one used in C99) can be employed here, and in particular Spectral Graph Analysis, close in spirit to SVD, seems promising.

5.5 DISCUSSION

This chapter presented a novel algorithm for semantic text segmentation. The core idea of this algorithm is using Latent Semantic Analysis on overlapping document chunks, and comparing topic deviations by mean of a window sliding through the text. Although the algorithm satisfies the Information Retrieval needs for robustness, independence of language, independence of domain and of adjustable segmentation granularity outlined in the introduction, it is qualitatively is still far off human performance. Its target application is one where heterogeneous texts of widely varying lengths can be expected. Here it exhibits encouraging performance and outperforms other commonly used (unsupervised) algorithms. Its robustness draws from design which does not rely on linguistic tools such as sentence or paragraph boundary disambiguation, language dependent stemmers, stop-lists and thesauri. For the same reason, its crude and statistic-based behaviour is mainly suited for refinement of Information Retrieval systems, where a missed or superfluous boundary is not critical for overall system usability.

Perhaps the most surprising discovery comes from analysis of LSITiling's internal parameter of latent space dimensionality. This parameter can be tuned automatically or, as I showed above, simply left constant at a very small number. This is in contrast to the body of previously published literature, which generally relies on setting the dimensionality to magical multiples of 100, based on results from LSA research in other IR areas.

There are several ways in which the approach of LSITiling could be extended. Firstly, the notion of context overlap could apply equally well to other semantic models, such as Latent Dirichlet Allocation, Non-negative Matrix Factorization and other Vector Space Models from Chapter 2.5. Secondly, the relationship between the amount of overlap within the sliding window and LSITiling performance could deserve an additional study.

6

CZECH DIGITAL MATHEMATICS LIBRARY

In the previous chapter, I described an application of unsupervised semantic analysis to the problem of topic segmentation. This is an internal system task, not directly observable by the end user. In contrast, in this chapter I will presents work done on semantic analysis in the context of digital libraries. The goal is to directly improve user experience, through enabling similarity browsing in the *Czech Digital Mathematics Library, DML-CZ*[1].

This work was published at the *The Seventh International Conference on Mathematical Knowledge Management (MKM)* (Řehůřek and Sojka, 2008) and was also a part of my project within the *Rector's Programme to Support MU Students' Creative Work* in 2010.

6.1 MOTIVATION

Mathematicians from all over the world dream of a World Digital Mathematics Library (Jackson, 2003), where (almost) all of reviewed mathematical papers in all languages will be stored, indexed and searchable with the today's leading edge information retrieval machinery. Good resources towards this goal—in addition to the publisher's digital libraries—are twofold:

1. "Local" repositories of digitized papers such as NUMDAM (Bouche, 2008), DML-CZ (Sojka, 2005), or born-digital archives CEDRAM (Bouche, 2006)), arXiv.org[2].

[1] http://dml.cz

[2] http://arxiv.org/archive/math

2. Two review services for the mathematical community: both ZentralBlatt Math[3] and Mathematical Reviews[4] have more than 2,000,000 entries (paper metadata and reviews) from more than 2,300 mathematical serials and journals.

Google Scholar[5] is becoming increasingly useful in the meantime, but lacks specialized math search, plus metadata guessed from parsing crawled papers are of low quality (compared to the controlled repositories).

Table 6.1.: Example of Math Subject Classification (MSC) categories for the top-level category *20: Group theory and generalizations*.

MSC	**Category description**	**Remark**
20	Group theory and generalizations	primary MSC
20.30	Abelian groups	historical (1959-1972)
20.92	Semigroups, general theory	historical (1959-1972)
20.93	Semigroups, structure and classification	historical (1959-1972)
20Dxx	Abstract finite groups	
20Exx	Structure and classification of infinite or finite groups	
20Fxx	Special aspects of infinite or finite groups	
20Kxx	Abelian	
20L05	Groupoids (i.e. small categories in which all morphisms are isomorphisms).	For sets with a single binary operation, see *20N02*; for topological groupoids, see *22A22*, *58H05*.
20Mxx	Semigroups	
20Nxx	Other generalizations of groups	
20P05	Probabilistic methods in group theory	See also *60Bxx*.

Both review services agreed on the supported *Math Subject Classification (MSC)*, a two-level hierarchical classification scheme[6]. Most journals request authors to include MSC classification tags when submitting articles for publication; however, most retrodigitized papers published before 1990 have not been classified by MSC yet.

[3] http://www.zblmath.fiz-karlsruhe.de/MATH/

[4] http://www.ams.org/mr-database

[5] http://scholar.google.com

[6] http://www.ams.org/msc/

6.2 MSC CLASSIFICATION

Within the DML-CZ project we have investigated possibilities to classify (retrodigitized) mathematical papers by machine learning techniques, to enrich math searching capabilities and to allow semantically related search. Text of scanned pages is usually optically recognized, so machine learning algorithms may use full text in addition to metadata (and reviews, if any) to aid training.

Classification is an example of *supervised* text analysis: the training algorithm requires *labelled* examples with MSC codes assigned, in order to learn a decision function (a "classifier") that will thereafter automatically assign MSC codes to new, *unlabelled* articles. In (Řehůřek and Sojka, 2008), we have investigated several machine learning algorithms with respect to the quality of the resulting classifier. Detailed evaluation shows that with almost all methods we can easily exceed 90% classification accuracy to classify the first two letters of MSC (the *primary MSC*). With fine-tuning, the best method (Support Vector Machine with Mutual Information feature selection, *atc* term weighting and 500–2000 features) we can increase the accuracy to 95% or more.

Since the focus of this thesis is unsupervised learning, I will not go into detail of our MSC classification experiments. Instead, I will turn to *exploratory data analysis* of the DML-CZ library, where the goal is to offer similar articles to the interested user, without making use of any metadata (MSC codes or otherwise).

6.3 DATASETS

We used three digital libraries as data source for our experiments:

1. Metadata and full texts of mathematical journals covered by the DML-CZ project During the first three years of the project, we have digitized and collected data in a digital library, accessible via web tool called Metadata editor[7]. To date there are 351 volumes in the digitized part of 12 journals: 2,443 issues, 26,987 articles on 276,026 pages.

[7] editor.dml.cz

2. arXMLiv, a project for translating the arXiv library into XML with semantic MathML mark-up. We used part of arXMLiv that deals with mathematics, for a total of 20,666 articles.

3. 17,636 fulltext articles from the NUMDAM repository.

All 65,289 articles were tokenized and converted into bag-of-words vectors. We also extracted mathematical formulae where possible (arXMLiv and digital-born parts of DML-CZ), and treated them as additional vector space dimension. Here the hope was that mathematical mark-up, when available, may aid similarity assessment, in a similar way to words. The final dimensionality (size of vocabulary plus mathematical formulae) was 355,025 features. The implicit 65,289 × 355,025 matrix was very sparse (39 million non-zero entries, or density of 0.167%), which is expected and common in NLP applications.

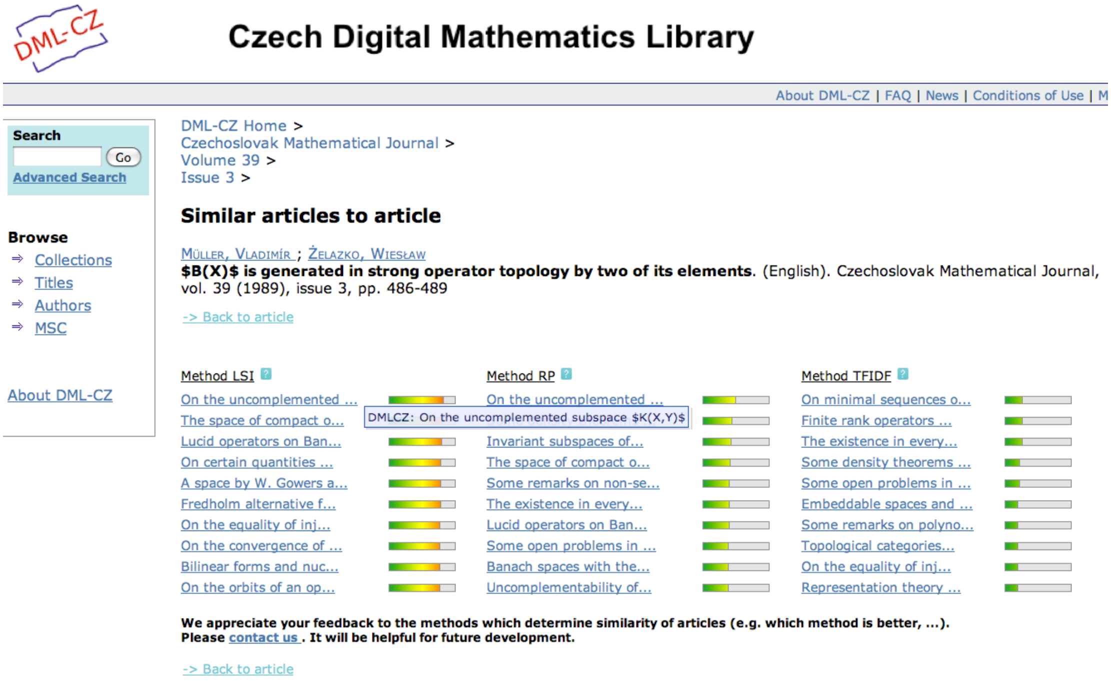
We did not perform any additional preprocessing steps beyond removing stop words (word types that appear in more than 30% of documents) and word types that appear in less than 5 documents (mostly OCR and spelling errors).

6.4 TOPIC MODELLING

For each document in our database, we periodically pre-compute the ten most semantically related articles and present them to the user through our web interface. Figure 6.1 illustrates these similarities for one randomly selected article.

A natural question is, which method of semantic analysis performs the best? For example, statistics show that the mean difference between normalized cosine similarity scores produced by TF-IDF vs. LSA is 0.0299, with a standard deviation of 0.037. Inspection reveals that in most cases the scores are indeed very similar, but there are also many pairs of documents for which the two methods vary widely, as the relatively large standard deviation would suggest. In Appendix B, I give an example of a pair of documents where TF*IDF suggested almost no similarity (score of 0.08) while LSA scored 0.98, i.e., near identity.

Examples like these are indicative of a failure of plain word overlap detection methods (TF-IDF is basically a weighted word unigram model),



Czech Digital Mathematics Library

About DML-CZ | FAQ | News | Conditions of Use | M

Search

Go

Advanced Search

Browse

- Collections
- Titles
- Authors
- MSC

About DML-CZ

DML-CZ Home >
Czechoslovak Mathematical Journal >
Volume 39 >
Issue 3 >

Similar articles to article

Müller, Vladimír ; Żelazko, Wiesław

$B(X)$ is generated in strong operator topology by two of its elements. (English). Czechoslovak Mathematical Journal, vol. 39 (1989), issue 3, pp. 486-489

-> Back to article

Method LSI	Method RP	Method TFIDF
On the uncomplemented ...	On the uncomplemented ...	On minimal sequences o...
The space of compact o...	DMLCZ: On the uncomplemented subspace $K(X,Y)$	Finite rank operators ...
Lucid operators on Ban...	Invariant subspaces of...	The existence in every...
On certain quantities ...	The space of compact o...	Some density theorems ...
A space by W. Gowers a...	Some remarks on non-se...	Some open problems in ...
Fredholm alternative f...	The existence in every...	Embeddable spaces and ...
On the equality of inj...	Lucid operators on Ban...	Some remarks on polyno...
On the convergence of ...	Some open problems in ...	Topological categories...
Bilinear forms and nuc...	Banach spaces with the...	On the equality of inj...
On the orbits of an op...	Uncomplementability of...	Representation theory ...

We appreciate your feedback to the methods which determine similarity of articles (e.g. which method is better, ...). Please contact us . It will be helpful for future development.

-> Back to article

Figure 6.1.: Web interface for similarity browsing within the DML-CZ library. Screenshot taken from `http://dml.cz//handle/10338.dmlcz/102320/SimilarArticles`.

and are therefore also interesting from a plagiarism detection perspective. Finding actual plagiates in our dataset is unlikely, due to the highly specialized nature of the domain (mathematics) and its structured and dedicated community, where each article is officially reviewed and classified by experts before publishing (with the review becoming permanently a part of that article, with the reviewer's name attached). Still, out of curiosity, we examined all pairs of highly similar articles, to see if we could find any cases of plagiarism within our union of articles from three different repositories.

The results were published in (Řehůřek, 2008a). Analysis shows that all suspicious document pairs are in fact innocuous conference announcements (differently worded though clearly templated), in-memoriams and the like—the documents in Appendix B are a good example. If there was plagiarism present in the dataset, its complexity was beyond the methods' ability to detect it.

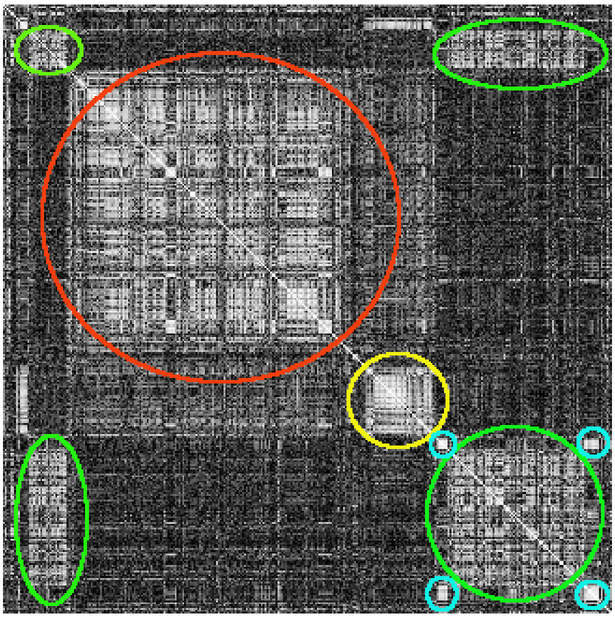


Figure 6.2.: An example of pairwise document similarities on documents about MSC 20 (*Group theory and generalizations*, see Table 6.1). Each pixel represents similarity of one pair of documents, the lighter the colour, the more similar. Note that the diagonal is necessarily white, because a document is always maximally similar to itself. The method used is Latent Semantic Indexing. See text for information on the highlighted regions.

6.5 EVALUATION

Perhaps more interesting than comparing one semantic analysis method to another is to consider how well do these similarities translate to semantic relatedness as perceived by us, human users. Unfortunately we do not have a referential tagged corpus of pair-wise document similarities to compare our results against. However, thanks to the nature of our dataset, we have access to article metadata. One piece of metadata present for each of our articles is its position within the MSC classification hierarchy. Although this metadata was not used during similarity computations (all used methods are unsupervised), we can use it for evaluation.

In Figure 6.2, we selected one node in the MSC hierarchy and considered only those documents in our collection that fall into this category. The category is named *20: Group theory and generalizations* and is further subdivided into smaller categories (*20Dxx Abstract finite groups* etc.), see Table 6.1. We group documents along the axes according to these subcategories and observe how well does the suggested similarity—represented by shades of gray—correspond to subcategory clusters suggested by MSC. We may observe there are four main "light" clusters, that is, clusters of documents that have high in-cluster similarity and low similarity to articles outside of the cluster. These are highlighted in red, yellow, green, blue and correspond to articles from categories *20Dxx+20Exx+20Fxx*, *20.30+20Kxx*, *20.92+20.93+20Mxx* and *20Lxx+20Nxx*, respectively. Although these visual results cannot substitute full analytical evaluation, this is an encouraging result: the LSA method correctly identified MSC boundaries assigned by hand by human reviewers, despite knowing nothing about MSC and not being allowed to use MSC during training.

Note that all of the suggested clusters are meaningful and also that the algorithm correctly linked obsolete categories *20.92* and *20.93* (used between the years 1959 and 1972) with their new version of *20Mxx*.

6.6 DO FORMULAE MATTER?

During article parsing, we extracted mathematical formulae from digital born documents. Are these additional features helpful for topic modelling? We used our algorithm from Chapter 4 to compute LDA topics over the 20,666 arXMLiv articles, which contain formulas in MathML notation. The result is shown in Table 6.2, where we can see that some short general formulas (delimited by "$" characters in that table) are indeed highly indicative of topic. Some of these features are even in the

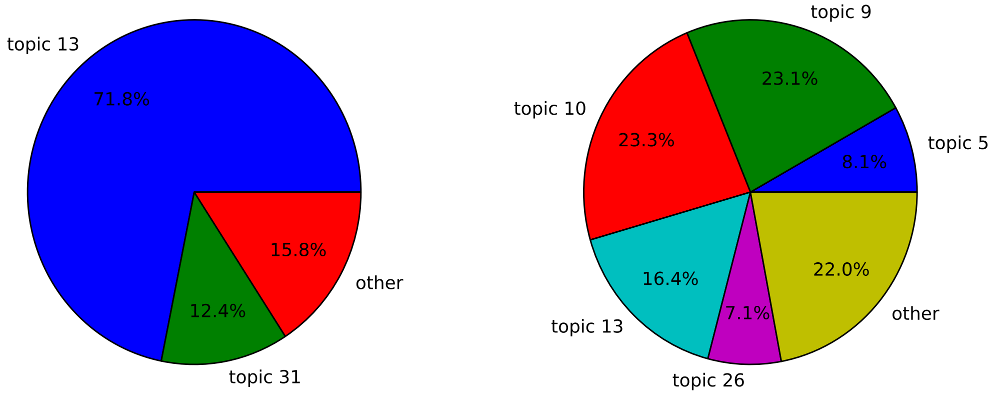
Table 6.2.: Ten most probable features for fourteen topics coming from a 50-topic LDA model. The model was trained on the arXMLiv database of 20,666 math articles. See Figure 6.3 for how these topics can combine to describe documents.

topic 1	topic 5	topic 9	topic 10	topic 11	topic 13	topic 14
$C*$	module	lattice	subgroup	$ϕ$	prime	operators
algebras	ring	lattices	orbit	triple	rational	measure
neumann	modules	vectors	subgroups	jordan	primes	norm
von	ideal	code	irreducible	triples	galois	spectral
unital	finitely	automorphism	$Γ$	homomorphism	adic	transform
unitary	rings	norm	abelian	E	characteristic	hilbert
projections	prime	primitive	character	$ψ$	fields	valued
$ ∗ $	exact	codes	representations	ternary	conjecture	spectrum
subalgebra	projective	roots	orbits	Z	power	fourier
trace	injective	unimodular	parabolic	projections	E	converges

topic 16	topic 24	topic 29	topic 30	topic 31	topic 32	topic 39
graph	process	$Ω$	curve	matrices	algebras	random
vertices	stochastic	(i)	curves	partition	module	probability
vertex	measure	(ii)	surface	row	graded	distribution
edge	initial	loop	genus	column	modules	process
edges	processes	kernel	plane	partitions	subalgebra	measure
path	motion	bergman	lines	entries	associative	bound
graphs	solutions	consequently	rational	odd	homomorphism	variables
tree	brownian	domain	intersection	$λ$	ideal	markov
$ν$	weak	$**K**$	singular	block	D	model
paths	diffusion	(iii)	divisor	rows	central	walk

first position, meaning they are the single most salient feature for the given topic.

As a reminder of Chapters 3 and 4, semantic models like LSA or LDA represent each input document as a soft mixture of topics. Figure 6.3 showcases two examples of such mixtures. It is produced by using the LDA model from arXMLiv described above to infer topic mixtures of documents of four DML-CZ articles. Note that the two articles depicted in that figure were not used at all during LDA training; the topic inference happens over new, unseen documents using a pre-trained model. Semantic similarity browsing in the digital library is realized through comparing such topic distributions, as opposed to (or rather, in addition to) the more common keyword search.



(a) *"On superpseudoprimes"*

(b) *"Recognizability in the lattice of convex ℓ—subgroups of a lattice-ordered group"*

Figure 6.3.: LDA topic mixtures for two DML-CZ documents. Only topics represented by at least 5% are explicitly plotted; contribution of all smaller topics is aggregated under "other". See Table 6.2 for a quick view of some of the topics. The two articles are *On superpseudoprimes* (http://dml.cz/handle/10338.dmlcz/130094) and *Recognizability in the lattice of convex ℓ—subgroups of a lattice-ordered group* (http://dml.cz/handle/10338.dmlcz/101966) and were not part of the training corpus.

6.7 conclusion

Digital libraries are an ideal test-bed for methods of unsupervised text analysis. The vast amount of (often unstructured and very noisy) data coupled with a need to intelligently guide user in his quest for information call for an automated approach to data mining.

The results of supervised analysis (Mathematics Subject Classification categorization) show feasibility of the machine learning approach to classification of mathematical papers. We used standard algorithms such as linear support vector machines and artificial neural networks, with preprocessing tools of feature selection, stemming, term weighting etc. Automated algorithms of this sort could greatly help with reviewer assignment for new article submissions, to name just one concrete application of our research.

Unsupervised analysis (similarity browsing, data exploration) is notoriously harder to evaluate; nevertheless, the results presented in this chapter look promising. In particular, topics reconstructed by the Latent Dirichlet Allocation show good interpretability and semantic coherence. We are offering an evaluation form to our users (mathematicians accessing the DML-CZ library), but a statistically significant amount of data proves hard to collect. In the future, we plan to conduct controlled human experiments to assess topic quality, following the evaluation framework proposed in Chang et al. (2009).

An extension of this line of research is also part of the European Digital Mathematics Library (EuDML) project[8].

[8] http://eudml.eu/

7

LANGUAGE IDENTIFICATION

Automated language identification of written text is a well-established research domain that has received considerable attention in the past. By now, efficient and effective algorithms based on character n-grams are in use, mainly with identification based on Markov models or on character n-gram profiles. This chapter will investigate the limitations of these approaches when applied to real-world web pages. The challenges to be overcome include language identification on very short text segments, correctly handling texts of unknown language and segmenting documents written in multiple languages.

The work described here was published in (Řehůřek and Kolkus, 2009) and presented at the *10th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)* in Mexico City. It places particular emphasis on classification robustness and efficiency, as it was developed by me and Milan Kolkus for the Seznam.cz company, where it is used in production in its fulltext search engine.

7.1 MOTIVATION

The amount of information available on the Internet is staggering and still growing at a fast pace. To make this information accessible, applications have sprung up to fill the void and gather, process and present Web information to the knowledge-hungry user. Unfortunately, documents on the Web have historically been created with human reader in mind, in formats such as HTML, and are not readily understandable by computers. Although XML and semantic mark-up (e.g. the `xml:lang` attribute, or the `<div lang="en">` construct) have been introduced to alleviate these problems, reality remains that many documents do not make use of metadata tags or, even worse, make use of them incorrectly and provide misleading information.

By not having metadata provided for us, or by deciding not to trust it, we are left with deducing information from the text itself. This is the domain of Natural Language Processing (NLP) and text mining. This article deals with one aspect of text mining, namely telling which language (or languages) is a given Web page written in, if any.

7.2 RELATED WORK

A general paradigm in automated language identification is to create language models during a training phase and compare input documents against these models during language identification. This places the task into the domain of *supervised learning* methods. Another consequence is that the set of target languages needs to be known beforehand, which makes language identification a *classification* problem.

A "common words" approach (Ingle, 1980) is based on the observation that for each language, there is a small class of words that carry little information but make up a large portion of any text. These are called the *function words* or *stop words* and their presence is to be expected, as word distribution in natural languages follows Zipf's law.

In (Dunning, 1994) it is noted that humans need surprisingly little in order to correctly identify a language. Interestingly, this is the case even if they are not proficient in that language or when the text snippet is quite short. This observation leads to a class of algorithms based on character (or even byte) n-grams, as opposed to more linguistically refined syntactic or semantic methods.

- A popular tool called *textcat* (Cavnar and Trenkle, 1994) constructs a ranking of the most frequent character n-grams for each language during the training phase and proclaims this ranking the *language model*. For classification, a ranking is constructed for the input document in the same fashion and is compared against each available language model. The closest model (in terms of ranking distances, see (Cavnar and Trenkle, 1994) for details) wins and is returned as the identified language.

- Another character n-gram approach pioneered by (Dunning, 1994) computes likelihood of generating the observed character sequence explicitly, through use of higher order Markov models. Let S be a sequence which consists of n characters $(s_1, \ldots, s_n)$. Then the probability this sequence was generated by Markov model L of order k is given by

$$P(S \mid L) = p(s_1, \ldots, s_k \mid L) \prod_{i=k}^{n} p(s_{i+1} \mid s_{i-k+1} \ldots s_i, L),$$

  where the first factor is the initial state distribution and the conditional probability describes transitions. Training the language model consists of estimating these transition probabilities. Again, winner is the language with the best likelihood of generating the input text. It is observed that using character trigrams, i.e. Markov models of order 2, already gives optimal results and increasing the model order therefore cannot affect performance much. For a comparison of character trigrams to "common words", see (Grefenstette, 1995).

- A related approach makes use of Shannon's information theory and compares language entropies (Teahan, 2000). Here Markov models are also estimated based on training data. In contrast to (Dunning, 1994), all models of orders $0, \ldots, k$ are used and their relationship explicitly modelled. This allows the algorithm to fall back to lower order models in case of insufficient data through mechanism called *escape probabilities*. Decision function views input as a stream of characters and in accordance with information theory tries to predict the next character in the stream. Success of these predictions is measured by cross-entropy and the model with the lowest cross-entropy after having processed the whole stream wins. Because of its ties to information theory and language compression, this technique is sometimes called the *compression* technique.

Apart from individual algorithms, research into language recognition has also identified key factors which directly influence performance:

- **Size of the training data**. Methods are evaluated based on how quickly their models converge, given differing sizes of training corpora. Note that more is not necessarily better here, as there is a risk of *overtraining* or *overfitting* the model with training data.

- **Size of the input text**. Methods can be distinguished by how much text they need to be given in order to reliably identify its language. Amount of text can be roughly divided into *small* (a phrase, less than 30 characters or up to 5 words), *large* (a paragraph, more than 300 characters or 50 words) and *medium* (a sentence, in between).

7.3 PROPOSED METHOD

Motivation for Change

Summing up the previously mentioned articles, there are several reasons behind the success of language modelling via character n-grams:

- **Fast convergence**. Very small training corpora (in the order of hundreds of kilobytes of text) are required to learn the models. See e.g. (Souter et al., 1994) for a study on speed of model convergence for character bigrams and trigrams.

- **Robust**. As long as the overall letter distribution in input document follows that of training examples, problematic language phenomena such as neologisms (words newly introduced into the language), spelling errors, rare inflections or unknown words are handled gracefully.

- **Domain independent**. In (Dunning, 1994) this approach was applied to a domain as distant as that of genetic sequence identification. Another often highlighted feature is that character n-gram methods do not require tokenization of the input, making them also suitable for Asian languages where tokenization is an interesting challenge in itself.

We implemented, for some time used and then evaluated an algorithm based on the compression technique (Teahan, 2000). We estimated all

i-gram distributions for $i = 0, \ldots, n$ and then combined them through an *Expectation Maximization* (EM) smoothing algorithm on held-out data. We were interested in detecting nine European languages: French (*fr*), English (*en*), Italian (*it*), Spanish (*es*), Slovakian (*sk*), Czech (*cs*), Slovenian (*sl*) and Polish (*pl*). Although this method worked almost perfect on our test data, a number of real-world issues soon became apparent when applied to the task of Web page classification. The main problem was not insufficient accuracy of classification as such, but rather a shift in the formulation of the language identification problem:

- **No *unknown language* option**. All methods listed above based on entropy, Markov processes or n-gram profiles return the nearest, best-fitting language. They assume that a) the set of languages is complete, known and trained for beforehand and b) that the input text is in exactly one of them. While the first assumption can be dismissed as our own choice, the latter is unrealistic for the Web.
- **Multiple languages**. In an also rather frequent scenario, there are parts of the input document which are in different languages. This may stem from a page's logical structuring (menu, text body, copyright notices) but also from the nature of the text body itself. This moves the document away from any one model and the language models become mixed in undesired ways. As a result, the document may even be identified as a completely unrelated language not present in the input text at all. In our experience, multilingual documents somehow often ended up being marked as Slovenian.
- **Close languages (same language family)**. As seen above, our language set includes Slovenian, Slovakian, Czech and Polish, which are all Slavic languages with considerable grammatical as well as lexical overlap. This is exacerbated by the fact that real texts on the Web often come in deaccented version, so that the trained models are unable to even theoretically take advantage of otherwise telling national characters (*ř* for Czech, *ľ* for Slovak etc.).

As a special case of the second point, there are many pages where letter distribution is heavily skewed by repetition of certain words or phrases. This includes discussion lists with "`In reply to:`" fields and so on. This problem does not come up in well-behaved corpora, but quickly becomes a nuisance when dealing with the Web.

To address the first two issues, we tried augmenting our implementation of the n-gram algorithm. We looked for a minimum threshold for each language that a document score has to exceed in order to be identified as that particular language, even if its score is the best available. Note that for language detection, document length is not an issue, as all models are evaluated on the same number of n-grams and the score numbers are thus directly comparable. For the fixed threshold to work, however, the scores need to be normalized to negate the effect of varying document lengths, as adding even one n-gram changes the order of magnitude of the probability scores.

Although we tried setting this threshold automatically, based on held-out training data, the results were not satisfactory. It appears that the per-character cross-entropies are dependent on the content of text n-grams in a way that prohibits direct absolute comparison against any fixed threshold. In other words, it proved impossible to find a threshold that would allow us to tell "this best fitting language is in fact an error". We also tried learning a special *unknown language* model from a hotchpotch of documents in various random languages. This worked better and solved the Slovenian classification problem, but seems rather ad-hoc and theoretically unfounded.

To avoid the headache of further complicating an already complex algorithm, we set out to try a different approach.

Dictionary Method

In (Dunning, 1994), dictionary methods (i.e. methods based on words rather than characters) are discussed and dismissed, based on their best known representative, "common words", being too restrictive and only applicable to longer texts.

Going through the list of n-gram advantages, the benefits of broad domain independence, no required tokenization and fast model convergence will indeed have to go. Since our goal is to tell apart European (Latin character based) natural languages, the first two are not really a concern. The last one, a large amount of training examples required by the dictionary methods, was perhaps a drawback back when these methods were developed. In the present day, with Web as Corpus (Kilgarriff, 2001) projects and NLP advancements, fast indexing and retrieval techniques, the amount of available data is no longer a critical issue. The same cannot be said for runtime performance, as the same reason why there are many documents requires us to process them at increased speed. For these reasons we decided to revisit the dictionary method.

We take a qualitatively different approach to constructing the dictionary language models. Rather than looking for words that are common in a given language (called *function* or *stop* words), we note which words are *specific* for a language, or rather, *how specific* they are. The foundation of our algorithm is a relevance mapping

$$\mathrm{rel}(word, language) : W \times L \mapsto \mathbb{R}$$

where W is a set of all words present in the training data and L the set of considered languages. We call the real-valued score of word $w \in W$ in a language $l \in L$ its *relevance*. In other words, the mapping is not binary as in the case of the "common words" approach, but rather a soft grading of words. Positive relevance hints at the word being indicative of the language, relevance around zero naturally corresponds to "no correlation" and negative values to "this word is indicative of *absence* of the language". We will call these *positive*, *near-zero* and *negative* evidence, respectively.

Naturally, the relevance mapping is constructed automatically from labeled training data. In contrast to character n-gram models, the convergence is much slower and significantly larger training corpora are required. We estimate the word relevance using reasoning detailed in (Kornai, A. et al., 2003). Their idea, although developed for classifying documents into topics, can also be applied to our problem of language identification. Below we give a short overview of the main assumptions and steps behind derivation of the final relevance formula; for a more thorough discussion on various aspects, please see the original article (Kornai, A. et al., 2003).

We start by noting frequencies of words $w_1, w_2, \ldots, w_N$ within each language corpus and compare them to frequencies in a general, *background* corpus. In this context, a corpus C is simply a collection of D documents, $C = (d_1, d_2, \ldots, d_D)$. For each language lang, we have a corpus C_{lang} of documents only in that language, plus one general background corpus which represents a collection of documents of background language lang_0. This background language is ideally completely language neutral, or more realistically represents the distribution of all languages on the Web. To approximate lang_0, we consider the union of all language corpora to be the background corpus, $C_0 = \bigcup C_{\mathrm{lang}}$. The *uncorrected observed frequency* of word w in language lang is then

$$\bar{g}_{\mathrm{lang}}(w) = \frac{\mathrm{TF}(w, C_{\mathrm{lang}})}{\#(C_{\mathrm{lang}})}, \tag{7.1}$$

with $\#(C)$ being the total number of words in corpus C and TF the number of occurences of a word in a corpus, and

$$g_0(w) = \frac{\mathrm{TF}(w, C_0)}{\#(C_0)} \tag{7.2}$$

for the background language.

From the assumption of languages being modelled as Bernoulli (word unigram) sources, the probability that a document d that contains f_i instances of word w_i was produced by language lang is given by the multinomial

$$P(d \mid \mathrm{lang}) = \binom{f_0 + f_1 + \cdots + f_N}{f_0, f_1, \ldots, f_N} \prod_{i=0}^{N} g_{\mathrm{lang}}(w_i)^{f_i}. \tag{7.3}$$

To avoid singularities for zero frequencies, the *Jelinek-Mercer smoothing* correction is introduced

$$g_{\mathrm{lang}}(w) = \alpha g_0(w) + (1 - \alpha)\bar{g}_{\mathrm{lang}}(w) \tag{7.4}$$

for some small value of α, such as 0.1.

After substituting (7.4) into (7.3), we compute logarithm of the probability ratio that a document was emitted by lang rather the background language $\mathtt{lang}_0$ by

$$\log\frac{P(d \mid \mathtt{lang})}{P(d \mid \mathtt{lang}_0)} = \sum_{i=0}^{N} f_i \log\frac{\alpha g_0(w_i) + (1-\alpha)g_{\mathtt{lang}}(w_i)}{g_0(w_i)} \tag{7.5}$$

An interesting observation is that negative and near-zero evidence contributes very little to classification accuracy. In fact, accuracy actually *improves* when near-zero and negative evidence is purposefully omitted. Translated to our language identification problem, we only pay attention to words that are highly indicative of the given language, disregarding near-zero and negative evidence entries. This has the pleasant side-effect of keeping the models reasonably sized, despite there being virtually tens of millions of possible words in each language relevance mapping. With this simplification the formula becomes an elegant and manageable

$$\sum_{g_L(w_i) \ll g_{\mathtt{lang}}(w_i)} f_i \cdot \mathtt{rel}(w_i, \mathtt{lang}), \tag{7.6}$$

where $\mathtt{rel}(w, \mathtt{lang}) = \log(g_{\mathtt{lang}}(w)) - \log(g_0(w))$ is our desired relevance of word w in language $\mathtt{lang}$. Put in words, the relevance of a word measures the orders of magnitude by which it is more frequent in the specific language corpus compared to the background corpus. This a surprisingly simple relationship, given we started only from the assumption of word independence (Bernoulli model). Another way to look at the formula is to realize that f_i corresponds to Term Frequency (TF) and $\mathtt{rel}(w_i, \mathtt{lang})$ to a kind of Inverse Document Frequency (IDF) component, linking this result to the general framework of TF-IDF classifiers. Obviously this is a sharp divergence from the idea of identifying languages by the most "common words".

With all pieces in place, how do we go on choosing which languages a sequence of words belongs to? According to the above derivation, we simply iterate over words that are distinctive of each language and sum their relevancies. We may compare this value to a threshold to immediately see if there was enough evidence to proclaim the document d as belonging to language $\mathtt{lang}$. But to abstract from document length, we first divide this sum by the length of the document in words, that is, we take average of the individual word relevancies. The final decision function which identifies document d as coming from language $lang$ is then

$$\mathrm{score}(d, lang) = \frac{\sum_{g_L(w_i) \ll g_{lang}(w_i)} f_i \cdot \mathrm{rel}(w_i, lang)}{\sum f_i} \geqslant t_{lang}. \tag{7.7}$$

The threshold t_{lang} is found, for each language separately, by optimizing an objective function on held-out data. One candidate for objective function is the F_1 measure, the generalized formula of which is

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

F_1 measure is popular in Information Retrieval and defines an equal trade-off between precision and recall. Other objective functions are possible, depending on the desired application of language identification. If the cost of not identifying the right language (*false negative*) is higher than cost of erroneously identifying an unwanted language (*false positive*), higher preference should be given to recall (e.g. via the F_2 measure) and vice versa. This effectively lowers (resp. raises) the estimated language threshold.

Contrary to results obtained from using thresholds for character n-gram method, detecting unknown language works quite reliably (as will be shown in the Evaluation section below). Because some words may be indicative of several languages (such as the previously mentioned lexical intersection of Slavic languages), more than one language may be recognized, too.

Practical Considerations

As noted earlier, runtime performance of classification is important. Interpreting equation (7.7), the algorithm consists of tokenizing input text, averaging token relevancies and comparing this sum to a precomputed threshold. This can be done extremely fast, using any of the many commonly available data structures which map strings into numbers.

As for memory considerations, the mapping that needs to be stored is very sparse, consisting of only those words which are distinctive for a language. In fact, the size of each language model when stored as

Table 7.1.: Overall data and model statistics.

Language	**Dump size**	**No. unique**	**No. test documents**			**Dictionary model**
code	**[GB]**	**sentences [k]**	small	medium	large	**size [words]**
cs	4.8	2,926	814	907	814	551,126
de	39.4	27,010	461	916	762	944,450
en	208.3	74,926	448	980	998	548,649
es	18.9	10,848	520	891	742	318,423
fr	39.8	18,048	483	852	765	373,432
it	26.0	11,529	469	836	727	378,817
pl	18.0	10,157	286	878	784	799,180
sk	3.3	1,769	275	916	768	474,003
sl	2.8	1,472	249	916	795	288,442

Patricia trie (Morrison, 1968) was in the tens of megabytes, which is comparable to size of our character pentagram models. This is not surprising as character pentagrams already come close in length to whole words.

We solved the practical question of obtaining large and representative language corpora by using Wikipedia dumps[1]. As research into Web corpora (Kilgarriff, 2001) rapidly progresses, it can be expected that assembling large text collections will become less and less of a problem in the future. It must be kept in mind however that common NLP techniques like stemming or lemmatization may not be applied, as these are dependent on language—the very thing we don't know and want to determine in the first place.

Evaluation

To evaluate our algorithm, we trained it on Wikipedia dumps of the nine target languages. As a reminder, these are French (*fr*), English (*en*), Italian (*it*), Spanish (*es*), Slovakian (*sk*), Czech (*cs*), Slovenian (*sl*) and Polish (*pl*). To avoid overfitting the training data, we discarded duplicate sentences and only used each sentence once in our corpus. Sentences with non-Latin (mostly Asian and Arabic) characters were also ignored. Some data statistics can be seen in Table 1, where the number of unique

[1] The static HTML dump as downloaded from `http://static.wikipedia.org`, June 2008.

sentences corresponds to the size of training data. In the same table we also give final model statistics. We put aside three thousand sentences of differing lengths for each language, to be used as test data. These were divided into *small*, *medium* and *large* sub-corpora (with texts of 2–5 words, 6–50 words and over 50 words, respectively), so that each sub-corpus contained exactly 1,000 texts. We manually checked the test corpora and estimated that the ratio of erroneously labeled examples is

- about 10% for medium length documents (mostly municipality and proper name enumerations),
- about 20% for long texts (same reason, plus many texts are in fact English phrases such as song or book titles)
- and as much as 50–70% for the short texts.

Short texts are especially bad because they concentrate "sentences" consisting of formulas, location entries, article headings with a person's name and lifetime and so on. All of these often refer to foreign institutions and have no connection to the language of the main article. Final sizes of test corpora after removing these problematic texts are given in Table 1.

Classification results are summarized in Table 7.2, which also includes results of our implementation of the cross-entropy based character n-gram algorithm described earlier, on the same data. Recall is measured as the ratio of true positives to all available positives (including false negatives), precision is the number of true positives divided by the number of all positives returned (including false positives). Note that this gives more room for precision errors to the dictionary method, which can return multiple false positives for each document, unlike the n-gram method that returns at most one incorrect language per document.

Inspection of results reveals that the errors closely follow the data problems described above. On one hand this is vexing because it prohibits more exact evaluation. On the other hand it shows that despite the considerable amount of noise in training data (which obviously shares the same problems as the test data) and in face of contradictory information, the classifiers are able to generalize. However, we'd like to stress

Table 7.2.: Precision/recall on test data, in percent.

	n-gram method			dictionary method		
language		**text size**			**text size**	
code	small	medium	large	small	medium	large
cs	81.9/75.2	96.8/96.0	100.0/100.0	64.9/84.0	85.2/96.9	97.8/99.6
pl	84.1/67.6	97.4/96.5	97.9/97.2	82.9/90.2	95.5/97.0	96.9/97.5
sk	81.6/77.7	97.7/96.9	99.3/99.0	57.8/82.9	71.4/96.6	87.6/96.7
sl	89.3/79.7	97.8/97.2	99.3/99.2	68.6/88.2	91.9/97.2	98.8/99.0
it	81.9/58.4	98.7/96.4	99.9/99,8	78.6/88.1	95.8/98.0	99.4/99.7
fr	80.1/52.9	98.3/97.3	99.8/99.6	82.7/88.7	98.4/99.0	99.5/99.6
de	85.2/73.6	98.8/98.1	99.0/98.4	85.7/91.8	98.2/99.6	98.8/99.2
es	81.5/61.6	99.0/98.1	100.0/99.9	73.2/86.4	94.3/98.9	99.3/99.8
en	81.4/51.7	99.4/98.2	99.8/99.1	86.1/91.6	99.2/99.7	99.8/99.4

Table 7.3.: Precision/recall on pruned test data, in percent.

	n-gram method			dictionary method		
language		**text size**			**text size**	
code	small	medium	large	small	medium	large
cs	93.5/92.4	98.7/98.7	100.0/100.0	73.9/99.8	86.7/100.0	98.1/100.0
en	93.1/67.3	99.7/98.6	100.0/100.0	98.4/100.0	99.7/100.0	100.0/100.0

the fact that our goal here is not to discuss the absolute numbers, but rather to juxtapose and compare two language identification methods on the same dataset.

To confirm our hypothesis of poor data quality, we manually checked labels of all Czech and English test examples. The labeling error was about 1% for medium and large texts and about 40% for texts of small length. We expect this error to be similar for the seven remaining languages, too. We re-ran language identification experiments on the two manually pruned corpora, with results summarised in Table 7.3. Many short Czech documents are classified as both Czech and Slovak by the dictionary method (reflecting a real ambiguity of the input), which results in lower precision but still excellent recall.

We conclude that the numbers are sufficiently high (in fact, after discounting the test data noise, nearly optimal) for both algorithms. The main difference and actually the reason why we developed our dictionary method in the first place is the added value of being able to return a *set* of languages as the identification result, including the elusive empty set.

7.4 SEGMENTING FOR LANGUAGE

There is one item on our language identification wish-list that has not been covered yet: correct classification of documents that contain blocks from different languages. While our character n-gram method based on cross entropy returns a random language in this case, dictionary method returns an unknown language. Both results are wrong. A practical extension to either algorithm would ideally allow us to locate and identify all compact single-language blocks.

To our knowledge, the only attempt at language segmentation was made in (Teahan, 2000). The authors consider all possible combinations of language change at each character in the input text and measure the resulting entropy on such text blocks. Although they report brilliant results of 99.5% accuracy on *character level*, the method misses the mark in terms of speed by several orders of magnitude. Even with advanced dynamic programming optimizations, it took tens of seconds to segment a text (Teahan, 2000).

Here we describe and evaluate an algorithm that segments input text into monolingual blocks.

Let $S_{lang}(d) = (score(w_1, lang), \ldots, score(w_n, lang))$ be a sequence of individual unit scores (word relevancies or n-gram probabilities) for the n units in document d. We can view this sequence as a real-valued signal and use signal processing to smooth the signal, removing local extrema,

$$(smoothed)_i = fnc_{SIZE}(score(w_{i-SIZE}), \ldots, score(w_{i+SIZE})), \qquad (7.8)$$

for any language $lang$. We use $median$ with sliding window size $SIZE = 2$ as the smoothing function while noting that there is a direct connection between the smoothing window size and robustness to short extra-lingual segments in text. These manifest themselves as sharp local valleys and correspond to proper nouns, typing errors and other text anomalies. Although strictly speaking they really are different from the surrounding text, our task is to identify coherent language blocks that are meaningful on *discourse level*, rather than token level.

Once we have smoothed signals for all available languages, we identify local minima in them. This gives us a first estimate of potential segment boundaries. The proposed segment boundaries are not final though—many of them correspond to local minima in between two segments of the same language. We remerge these back into a single segment. Note that in this way we prohibit having two consecutive segments of the same language, but we may still arrive at segments with no language assigned to them. It is also possible to have a segment with more than one language. This means the text may have been written in either and is indistinguishable. This often occurs with shorter *cs/sk* passages and reflects real ambiguity of the input.

Complexity of the whole procedure is linear in the number of words and languages, $O(|d| \times |L|)$.

Evaluation

To evaluate language segmentation, we constructed an artificial corpus. The corpus contains 1,000 documents, each one of them being a concatenation of 1 to 4 segments in different languages. The numbers were picked to somewhat mimick situation on the Web, with 4 languages in a single document as an extreme case. Language segments are pooled randomly from a collection of medium-length texts in that language (6 to 50 words).

We give this concatenation to our segmentation algorithm, with signal scores based on word relevancies, and mark down languages predicted for each token. This per-token evaluation records $success$ each time a token was assigned precisely the one language that was expected, and $failure$ otherwise. Accuracy is then computed as $\#success/(\#success + \#failure)$. Note that assigning multiple languages or no language at all to a token always equals an error.

The algorithm misclassified 1,420 out of possible 49,943 words. This corresponds to 97.16% accuracy. In 603 cases, the exact boundary was missed by one word, which is still an acceptable error for our purposes. Discounting these off-by-one boundary errors, accuracy climbs to 98.34%. Closer inspection of the 817 misses left shows that some of them come from English collocations like *grand theft auto* which are embedded inside non-English text segments and regrettably "misclassified" as English by the algorithm. The real accuracy is therefore probably slightly higher, depending on mode of application.

Although these results are lower than those reported in (Teahan, 2000), the algorithm enjoys conceptual clarity and an impressive runtime performance.

7.5 DISCUSSION

This chapter's main contribution is revisiting and re-evaluation of some of the assumptions made 15 years ago, when the domain of automated language identification was being shaped. It proposes a straightforward, fully automated method which learns a decision function from training data. The decision function is based on word relevancies and addresses some aching problems of popular character n-gram based methods, while retaining character n-gram's excellent accuracy and actually improving runtime efficiency. The algorithm tends to err on the side of assigning *all* fitting languages when presented with ambiguous input (like the aforementioned short phrases from the close family of Slavic languages). In Information Retrieval parlance, this ensures high recall at the cost of lower precision. Given that the superordinate task is filtering out all documents *not* in a given, small set of languages (so that the search engine doesn't have to index them), high recall is the most important factor and this behaviour is desirable.

An easily overlooked but very important benefit of using words instead of character n-grams is that the system is more open to human introspection, more predictable in ways of interpreting its results ("looking inside the box") or selectively changing its behaviour—something of considerable value in real systems.

Built on top of this algorithm is a language segmentation algorithm, which is based on the notion of language signal strength within the input document. The way this segmentation algorithm was described here makes it an extension in the proper sense of the word (language identification works even without language segmentation), but a careful tuning of the combined algorithm proved critical for the overall IR system performance. Documents on the Web are often composed of segments in multiple languages, even when our human brain is very efficient at picking up the "most interesting" or "salient" part and proclaiming the document to be in a single language (of that one part). The decision on which languages to actually attribute to the document must be based on a reliable and efficient routine for language segmentation. This also opens up the possibility to only index parts of the input document. The segmentation algorithm described here is evaluated to behave acceptably using word relevancies and solves the problem of language identification in multilingual documents.

8

SUMMARY

In this thesis, I examined semantic processing of unstructured text from three perspectives. In the first part, I described several state-of-the-art approaches to modelling semantics, which are based on the idea of creating a *semantic model* from a (training) corpus of unstructered text. Through the trained model, it is then possible to express arbitrary documents (ones from the training corpus as well as others) in a new, semantic representation. In this representation, documents may be evaluated as closely related despite not sharing any common words—a strict departure from the more traditional "keyword" search systems. The advantage of using semantic models is therefore in assessing similarity at a higher, "topical" level.

The common theme among all algorithms was their focus on robustness toward input data noise and computational tractability. The second part of the thesis dealt with my own contributions to the field, which consisted of new training algorithms for Latent Semantic Analysis and Latent Dirichlet Allocation. The novelty lies in their focus on scalability—the algorithms run "online", with constant memory in the number of training documents so that they can process arbitrarily large input, and with computation distributed across a cluster of computers. Also, the input document stream does not require random access. This last point is especially relevant in environments where data is retrieved from slow media (compressed on disk, tape, accessed through the web). In case of LSA, the input stream does not even need to be repeatable—the algorithm only accesses each training document once, in sequential order (a *streamed single-pass algorithm*). This additional constraint allows processing infinite input streams: the model is updated online and the training observations may be immediately discarded. This online, incremental, streamed, distributed algorithm exhibits unparalleled performance and makes LSA processing feasible on web-scale datasets.

In the last part I considered applicability of the general-purpose semantic algorithms to the domain of Information Retrieval. Useful as they are, the algorithms are nevertheless only a first step towards a successful IR system. Issues of document heterogeneity often hurt performance; I presented two novel algorithms for increasing topical consistency among documents: a) by splitting documents into smaller, topically consistent blocks (Chapters 5) and b) by splitting multilingual documents into blocks of the same language (Chapter 7). Chapter 6 presented topic modelling applied to a real digital library, the Czech Digital Mathematics Library, DML-CZ.

OUTLOOK

All these algorithms represent steps in the direction of more automated and intelligent access to the vast digital repositories of today. It has been said that "the devil is in the details", and this is certainly true of IR systems. Despite the theoretical advances and complex data mining methods, expert knowledge in tuning a system is still invaluable. It is my belief that a relatively simple system that is properly tuned by someone with deep understanding of the problem at hand will most of the time outperform a complex, out-of-the-box system, even if the latter utilizes state-of-the-art techniques. In a way, this is a testament and a tribute to the ingenuity of human mind. On the other hand, the amount of raw digital data increases while the amount of human experts stays roughly the same, so despite their imperfections, automated methods considered in this thesis may serve very well in their limited goal of assisting humans during data exploration.

This idea of bringing semantic processing to non-experts (non-linguists and non-computer scientists) was also at the core of the design philosophy of a software package for topical modelling that accompanies this thesis (Řehůřek and Sojka, 2010). It is my hope (already partially fulfilled) that such software will lead to wider adoption of unsupervised semantic methods outside of the academic community. The focus on interface simplicity and clarity has paid off in the sense that users have found creative use for the included semantic algorithms, even outside of the intended domain of NLP.

BIBLIOGRAPHY

Abbasi, Ahmed, and Hsinchun Chen. 2006. Visualizing Authorship for Identification. In *Proceedings of the Conference on Intelligence and Security Informatics ISI 2006*, volume 3975 of *Lecture Notes in Computer Science*, 60–71. San Diego, CA, USA: Springer. [page 25, 82]

Airoldi, E., D. Blei, S. Fienberg, and E. Xing. 2007. Combining Stochastic Block Models and Mixed Membership for Statistical Network Analysis. *Statistical Network Analysis: Models, Issues, and New Directions* 57–74. [page 28]

Allen, James. 1995. *Natural Language Understanding (2nd ed.).* Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc. [page 7]

Anderson, E., Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammerling, J. Demmel, C. Bischof, and D. Sorensen. 1990. LAPACK: A Portable Linear Algebra Library for High-Performance Computers. In *Proceedings of the 1990 ACM/IEEE conference on Supercomputing*, 2–11. IEEE Computer Society Press. [page 34]

Asuncion, A., M. Welling, P. Smyth, and Y.W. Teh. 2009. On Smoothing and Inference for Topic Models. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, 27–34. AUAI Press. [page 32]

Attias, H. 2000. A Variational Bayesian Framework for Graphical Models. *Advances in neural information processing systems* 12:209–215. [page 69]

Batko, M., D. Novák, F. Falchi, and P. Zezula. 2006. On Scalability of the Similarity Search in the World of Peers. In *Proceedings of the 1st international conference on Scalable information systems*, 20. ACM. [page 19]

Beckmann, N., H.P. Kriegel, R. Schneider, and B. Seeger. 1990. The R*-tree: an Efficient and Robust Access Method for Points and Rectangles. *ACM SIGMOD Record* 19:322–331. [page 18]

Bentley, J.L. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM* 18:509–517. [page 18]

Berchtold, S., C. Böhm, B. Braunmüller, D.A. Keim, and H.P. Kriegel. 1997. Fast Parallel Similarity Search in Multimedia Databases. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, 1–12. ACM. [page 18]

Berchtold, S., D.A. Keim, and H.P. Kriegel. 2002. The X-tree: An Index Structure for High-Dimensional Data. *Readings in multimedia computing and networking* 12:451–462. [page 18]

Berry, M.W. 1992. Large-scale Sparse Singular Value Computations. *The International Journal of Supercomputer Applications* 6:13–49. [page 53]

Berry, M.W., and M. Browne. 2005. Email Surveillance using Non-Negative Matrix Factorization. *Computational & Mathematical Organization Theory* 11:249–264. [page 27]

Bestgen, Y. 2006. Improving Text Segmentation using Latent Semantic Analysis: A Reanalysis of Choi, Wiemer-Hastings, and Moore (2001). *Computational Linguistics* 32:5–12. [page 87]

Blei, D.M., and M.I. Jordan. 2003. Modeling Annotated Data. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, 127–134. ACM. [page 28]

Blei, D.M., A.Y. Ng, and M.I. Jordan. 2003. Latent Dirichlet Allocation. *The Journal of Machine Learning Research* 3:993–1022. [page 28, 31, 68, 69, 75]

Bouche, Thierry. 2006. A pdfLaTeX-based Automated Journal Production System. *TUGBOAT* 27:45–50. [page 99]

Bouche, Thierry. 2008. Towards a Digital Mathematics Library? In *CMDE 2006: Communicating Mathematics in the Digital Era*, ed. Eugénio M. Rocha, 43–68. MA, USA: A.K. Peters. [page 99]

Bradford, R.B. 2008. An Empirical Study of Required Dimensionality for Large-Scale Latent Semantic Indexing Applications. In *Proceeding of the 17th ACM conference on Information and knowledge management*, 153–162. ACM. [page 23]

Brand, M. 2006. Fast Low-Rank Modifications of the Thin Singular Value Decomposition. *Linear Algebra and its Applications* 415:20–30. [page 24, 39, 41, 43]

Brooks, R.A. 1990. Elephants Don't Play Chess. *Robotics and autonomous systems* 6:3–15. [page 4]

Cavnar, William B., and John M. Trenkle. 1994. N-gram Based Text Categorization. In *Ann Arbor MI*, 161–175. [page 10, 110]

Chalmers, D.J. 1992. Subsymbolic Computation and the Chinese Room. *The symbolic and connectionist paradigms: closing the gap* 25–48. [page 5]

Chang, J., J. Boyd-Graber, S. Gerrish, C. Wang, and D. Blei. 2009. Reading Tea Leaves: How Humans Interpret Topic Models. In *Neural Information Processing Systems*. Citeseer. [page 108]

Choi, F. 2002. A Speech Interface for Rapid Reading. In *IEE Seminar on Speech and Language Processing for Disabled and Elderly People (Ref. No. 2000/025)*, 2. IET. [page 83]

Choi, Freddy Y. Y., Peter Wiemer-Hastings, and Johanna Moore. 2001. Latent Semantic Analysis for Text Segmentation. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, 109–117. [page 87, 88, 96]

Choi, F.Y.Y. 2000. Advances in Domain Independent Linear Text Segmentation. In *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*, 26–33. Morgan Kaufmann Publishers Inc. [page 86, 93, 94]

Ciaccia, P., M. Patella, and P. Zezula. 1997. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, 426–435. Morgan Kaufmann Publishers Inc. [page 18]

Cobena, G., S. Abiteboul, and A. Marian. 2002. Detecting Changes in XML Documents. In *International conference on data engineering*, 41–52. IEEE Computer Society. [page 15]

Comon, P., and G.H. Golub. 1990. Tracking a Few Extreme Singular Values and Vectors in Signal Processing. *Proceedings of the IEEE* 78:1327–1343. [page 38, 41]

Cresswell, M.J. 1985. *Structured Meanings: The Semantics of Propositional Attitudes*. The MIT Press. [page 4]

Cruse, D.A. 2000. *Meaning in Language: An Introduction to Semantics and Pragmatics*. Oxford University Press Oxford, UK. [page 4]

Curran, J.R., and M. Moens. 2002. Improvements in Automatic Thesaurus Extraction. In *Proceedings of the ACL-02 workshop on Unsupervised lexical acquisition-Volume 9*, 59–66. Association for Computational Linguistics. [page 82]

Deerwester, S., S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman. 1990. Indexing by Latent Semantic Analysis. *Journal of the American society for information science* 41:391–407. [page 20, 143]

Deprettere, E.F. 1989. *SVD and Signal Processing: Algorithms, Applications and Architectures*. North-Holland Publishing Co. Amsterdam, The Netherlands, The Netherlands. [page 11]

Dietz, Laura. 2010. Directed Factor Graph Notation for Generative Models. Technical report, Max Planck Institute for Informatics, Saarbrücken, Germany. [page 29]

Ding, C., T. Li, and W. Peng. 2008. On the Equivalence Between Non-Negative Matrix Factorization and Probabilistic Latent Semantic Indexing. *Computational Statistics & Data Analysis* 52:3913–3927. [page 27]

Doyle, G., and C. Elkan. 2009. Accounting for Burstiness in Topic Models. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 281–288. ACM. [page 14]

Dreyfus, H. 1967. Alchemy and AI. *RAND Corporation P-3244* . [page 4]

Dumais, S.T., and T.K. Landauer. 1997. A Solution to Platos Problem: The Latent Semantic Analysis Theory of Acquisition, Induction and Representation of Knowledge. *Psychological review* 104:211–240. [page 21]

Dunning, T. 1994. Statistical Identification of Language. Technical report, Computing Research Lab (CRL), New Mexico State University. [page 110, 111, 112, 114]

Fei-Fei, L., and P. Perona. 2005. A Bayesian Hierarchical Model for Learning Natural Scene Categories. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, 524–531. IEEE. [page 28]

Feldman, R., and J. Sanger. 2007. *The Text Mining Handbook*. Cambridge, UK: Cambridge University Press. [page 10]

Fields, Ben, Christophe Rhodes, and Mark Inverno. 2010. Using song social tags and topic models to describe and compare playlists. *WOMRAD 2010: Workshop on Music Recommendation and Discovery* 8. [page 11]

Finkel, R.A., and J.L. Bentley. 1974. Quad Trees: a Data Structure for Retrieval on Composite Keys. *Acta informatica* 4:1–9. [page 18]

Firth, J.R. 1957. A Synopsis of Linguistic Theory 1930–1955. *Studies in linguistic analysis* 51:1–31. [page 6]

Foltz, P.W. 1996. Latent Semantic Analysis for Text-Based Research. *Behavior Research Methods Instruments and Computers* 28:197–202. [page 21, 87]

Foltz, P.W., W. Kintsch, and T.K. Landauer. 1998. The Measurement of Textual Coherence With Latent Semantic Analysis. *Discourse processes* 25:285–307. [page 21]

Forman, George, Isabelle Guyon, and André Elisseeff. 2003. An Extensive Empirical Study of Feature Selection Metrics for Text Classification. *Journal of Machine Learning Research* 3:1289–1305. [page 15]

Furnas, G.W., T.K. Landauer, L.M. Gomez, and S.T. Dumais. 1984. Statistical Semantics: Analysis of the Potential Performance of Keyword Information Systems. In *Human factors in computer systems*, 187–242. Ablex Pub. [page 6]

Févotte, C., N. Bertin, and J.L. Durrieu. 2009. Non-Negative Matrix Factorization with the Itakura-Saito Divergence: With Application to Music Analysis. *Neural Computation* 21:793–830. [page 27]

Gaussier, E., and C. Goutte. 2005. Relation Between PLSA and NMF and Implications. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, 601–602. ACM. [page 27]

Gelman, A. 2004. *Bayesian Data Analysis*. CRC press. [page 28]

Golub, G.H., and C.F. Van Loan. 1996. *Matrix Computations*. Johns Hopkins University Press. [page 11, 21, 25, 38, 44]

Gorrell, G., and B. Webb. 2005. Generalized Hebbian Algorithm for Incremental Latent Semantic Analysis. In *Ninth European Conference on Speech Communication and Technology*. [page 39, 54]

Graesser, A.C., K. VanLehn, C.P. Rosé, P.W. Jordan, and D. Harter. 2001. Intelligent Tutoring Systems with Conversational Dialogue. *AI Magazine* 22:39. [page 21]

Grefenstette, G. 1994. *Explorations in Automatic Thesaurus Discovery*. Springer. [page 82]

Grefenstette, G. 1995. Comparing Two Language Identification Schemes. In *Proceedings of the 3rd International Conference on the Statistical Analysis of Textual Data (JADT'95)*. [page 111]

Griffiths, T.L., and M. Steyvers. 2004. Finding Scientific Topics. *Proceedings of the National Academy of Sciences of the United States of America* 101:5228. [page 23, 28, 32]

Griffiths, T.L., M. Steyvers, D.M. Blei, and J.B. Tenenbaum. 2005. Integrating Topics and Syntax. *Advances in Neural Information Processing Systems* 17:537–544. [page 28]

Griffiths, T.L., M. Steyvers, and J.B. Tenenbaum. 2007. Topics in Semantic Representation. *Psychological Review* 114:211–244. [page 7, 21]

Guttman, A. 1984. R-trees: a Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, 47–57. ACM. [page 18]

Halko, N., P.G. Martinsson, and J.A. Tropp. 2009. Finding Structure with Randomness: Stochastic Algorithms for Constructing Approximate Matrix Decompositions. *ArXiv e-prints* 0909.4061v2. [page 38, 39, 50, 66]

Hanks, Patrick. 2011. *Analyzing the Lexicon: Norms and Exploitations*. MIT Press. [page 7]

Harris, Z. 1954. Distributional Structure. *Word* 10:146–162. [page 6]

Hearst, M.A. 1994. Multi-paragraph Segmentation of Expository Text. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, 9–16. Association for Computational Linguistics. [page 85, 86, 92]

Hearst, M.A., and C. Plaunt. 1993. Subtopic Structuring for Full-Length Document Access. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, 59–68. ACM. [page 83]

Hoffman, M.D., D.M. Blei, and F. Bach. 2010. Online Learning for Latent Dirichlet Allocation. In *Advances in Neural Information Processing Systems (NIPS)*. MIT Press. [page 68, 72, 73]

Hofmann, T. 1999. Probabilistic Latent Semantic Indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, 50–57. ACM. [page 27]

Ingle, N. 1980. *A Language Identification Table*. Technical Translation International. [page 110]

Jackson, Allyn. 2003. The Digital Mathematics Library. *Notices Am. Math. Soc.* 50:918–923. [page 99]

Juola, Patrick. 2006. Authorship attribution. *Foundations and Trends in Information Retrieval* 1:233–334. [page 82]

Kan, M.Y., J.L. Klavans, and K.R. McKeown. 1998. Linear Segmentation and Segment Significance. In *Proceedings of the 6th International Workshop of Very Large Corpora*, 197–205. [page 86]

L'uboš Kardoš. 2010. Python Library for Singular Value Decomposition. Bachelor's Thesis. Faculty of Informatics, Masaryk University in Brno, Czech Republic. [page 9]

Katayama, N., and S. Satoh. 1997. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, 369–380. ACM. [page 18]

Kilgarriff, A. 2001. Web as Corpus. In *Proceedings of Corpus Linguistics 2001*, 342–344. [page 115, 119]

Kintsch, W. 2000. Metaphor Comprehension: A Computational Theory. *Psychonomic Bulletin and Review* 7:257–266. [page 21]

Knight, Kevin. 2009. Bayesian Inference with Tears. URL `http://www.isi.edu/natural-language/people/bayes-with-tears.pdf`, accessed June 2010. [page 33]

Kohonen, Teuvo. 1983. PSelf-Organizing Mappings for Two-Dimensional (Visual) Display of High-Dimensional Pattern Spaces. In *Proceedings of 3SCIA, Scandinavian Conference on Image Analysis*, 35–41. Lund, Sweden: Studentlitteratur. [page 26]

Kontostathis, A., and W.M. Pottenger. 2006. A Framework for Understanding Latent Semantic Indexing (LSI) Performance. *Information Processing & Management* 42:56–73. [page 20]

Koren, Y., R. Bell, and C. Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42:30–37. [page 25, 27]

Kornai, A. et al. 2003. Classifying the Hungarian Web. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 1*, 203–210. Association for Computational Linguistics Morristown, NJ, USA. [page 115]

Kozima, H. 1993. Text Segmentation Based on Similarity Between Words. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, 286–288. Association for Computational Linguistics. [page 83]

Lakoff, G., and M. Johnson. 1999. *Philosophy in the Flesh: The Embodied Mind and its Challenge to Western Thought*. Basic Books. [page 4]

Landauer, T.K. 2007. *LSA as a Theory of Meaning*. Lawrence Erlbaum Associates. [page 21]

Landauer, T.K., P.W. Foltz, and D. Laham. 1998. An Introduction to Latent Semantic Analysis. *Discourse processes* 25:259–284. [page 82]

Leach, S. 1995. Singular Value Decomposition—A Primer. Unpublished Manuscript, Department of Computer Science, Brown University, Providence, RI, USA. [page 22]

Leacock, C., and M. Chodorow. 1998. Combining Local Context and WordNet Similarity for Word Sense Identification. *WordNet: An electronic lexical database* 49:265–283. [page 82]

Lee, D.D., and H.S. Seung. 2001. Algorithms for Non-Negative Matrix Factorization. *Advances in Neural Information Processing Systems (NIPS)* 13. [page 26]

Levy, A., and M. Lindenbaum. 2000. Sequential Karhunen-Loeve Basis Extraction and its Application to Images. *IEEE Transactions on Image processing* 9:1371. [page 24, 25, 39, 43]

Madsen, R.E., D. Kauchak, and C. Elkan. 2005. Modeling Word Burstiness using the Dirichlet Distribution. In *Proceedings of the 22nd international conference on Machine learning*, 545–552. ACM. [page 14]

Magistry, P., L. Prévot, and S.K. Hsieh. 2010. Classifier-Noun Composition in Mandarin: A Test Case for Distributional Semantic Models. In *Workshop on Compositionality and Distributional Semantic Models ESSLLI 2010*. [page 11]

Manning, C.D., and H. Schütze. 1999. *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press. [page 10]

Materna, P. 2004. *Conceptual Systems*, volume 1. LOGOS Verlag Berlin. [page 4]

Materna, Pavel, Karel Pala, and Jiří Zlatuška. 1989. *Logická analýza přirozeného jazyka*. Academia, 1st edition edition. [page 4]

Minka, T., and J. Lafferty. 2002. Expectation-Propagation for the Generative Aspect Model. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*, 352–359. [page 31]

Miranda, A.A., Y.A. Le Borgne, and G. Bontempi. 2008. New Routes From Minimal Approximation Error to Principal Components. *Neural Processing Letters* 27:197–207. [page 25]

Mitchell, T.M. 1997. *Machine Learning*. Mac Graw Hill. [page 5]

Moffat, A., and J. Zobel. 1996. Self-Indexing Inverted Files for Fast Text Retrieval. *ACM Transactions on Information Systems (TOIS)* 14:349–379. [page 17]

Morrison, D.R. 1968. PATRICIA—Practical Algorithm to Retrieve Information Coded in Alphanumeric. *Journal of the ACM (JACM)* 15:514–534. [page 119]

Newman, D., A. Asuncion, P. Smyth, and M. Welling. 2007. Distributed Inference for Latent Dirichlet Allocation. *Advances in Neural Information Processing Systems* 20:17–24. [page 68]

Nievergelt, J., H. Hinterberger, and K.C. Sevcik. 1984. The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Transactions on Database Systems (TODS)* 9:38–71. [page 18]

Page, L., S. Brin, R. Motwani, and T. Winograd. 1998. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project. `http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf`. [page 25]

Pantel, P., and D. Lin. 2002. Discovering Word Senses From Text. In *Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining*, 613–619. ACM. [page 82]

Pevzner, L., and M.A. Hearst. 2002. A Critique and Improvement of an Evaluation Metric for Text Segmentation. *Computational Linguistics* 28:19–36. [page 90]

Pinker, S. 2000. *The Language Instinct: How the Mind Creates Language*. Harper Perennial Modern Classics. [page 5]

Rao, CR. 1995. Use of Hellinger Distance in Graphical Displays. *Multivariate statistics and matrices in statistics. Leiden (Netherland): Brill Academic Publisher* 143–161. [page 17]

Řehůřek, R. 2007a. On Dimensionality of Latent Semantic Indexing for Text Segmentation. In *Proceedings of the International Multiconference on Computer Science and Information Technology*, 347–356. Wisła, Poland. [page 23]

Řehůřek, R. 2010. Fast and Faster: A Comparison of Two Streamed Matrix Decomposition Algorithms. In *NIPS 2010 Workshop on Low-rank Methods for Large-scale Machine Learning*, volume 6. Vancouver, Canada. [page 24, 37, 51, 59]

Řehůřek, R., and J. Pomikálek. 2007. The Influence of Preprocessing Parameters on Text Categorization. *International Journal of Applied Science, Engineering and Technology* 1:430–434. [page 15, 20]

Řehůřek, Radim. 2007b. Text Segmentation using Context Overlap. In *EPIA Workshops*, volume 4874 of *Lecture Notes in Computer Science*, 647–658. Guimarães, Portugal: Springer. [page 83]

Řehůřek, Radim. 2008a. Plagiarism Detection through Vector Space Models Applied to a Digital Library. In *Proceedings of the 2nd Workshop on Recent Advances in Slavonic Natural Languages Processing (RASLAN 2008)*, 75–83. Karlova Studánka, CZ: Masaryk University. [page 83, 103]

Řehůřek, Radim. 2008b. Semantic-based Plagiarism Detection. RNDr. thesis, Faculty of Informatics, Masaryk University. [page 81]

Řehůřek, Radim. 2011a. Speeding Up Latent Semantic Analysis: A Streamed Distributed Algorithm for SVD Updates. In *The 3rd International Conference on Agents and Artificial Intelligence (ICAART)*, volume 3. Rome, Italy: INSTICC Press. [page 37]

Řehůřek, Radim. 2011b. Subspace Tracking for Latent Semantic Analysis. In *Advances in Information Retrieval*, ed. Paul Clough, Colum Foley, Cathal Gurrin, Gareth J.F. Jones, Wessel Kraaij, Hyowon Lee, and Vanessa Mudoch, volume 6611 of *Lecture Notes in Computer Science*, 289–300. Springer. [page 37, 39, 59]

Řehůřek, Radim, and Milan Kolkus. 2009. Language Identification on the Web: Extending the Dictionary Method. In *Computational Linguistics and Intelligent Text Processing (CICLing)*, ed. Alexander F. Gelbukh, volume 5449 of *Lecture Notes in Computer Science*, 357–368. Mexico City, Mexico: Springer. [page 109]

Řehůřek, Radim, and Petr Sojka. 2008. Automated Classification and Categorization of Mathematical Knowledge. In *AISC/MKM/Calculemus*, volume 5144 of *Lecture Notes in Computer Science*, 543–557. Birmingham, UK: Springer. [page 99, 101]

Řehůřek, Radim, and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 45–50. Valletta, Malta: ELRA. [page 128]

Reynar, J.C. 1998. Topic Segmentation: Algorithms and Applications. Doctoral Dissertation, University of Pennsylvania. [page 83, 86]

Rijsbergen, C. J. Van. 1979. *Information Retrieval*. Butterworth. [page 15]

Robert, C.P., and G. Casella. 2004. *Monte Carlo Statistical Methods*. Springer Verlag. [page 30, 32]

Rosen-Zvi, M., T. Griffiths, M. Steyvers, and P. Smyth. 2004. The Author-Topic Model for Authors and Documents. In *Proceedings of the 20th conference on Uncertainty in Artificial Intelligence*, 487–494. AUAI Press. [page 28]

Roweis, S., and Z. Ghahramani. 1999. A Unifying Review of Linear Gaussian Models. *Neural computation* 11:305–345. [page 26]

Russell, S.J., and P. Norvig. 2009. *Artificial Intelligence: a Modern Approach*. Prentice Hall. [page 5]

Salton, G. 1971. *The SMART Retrieval System—Experiments in Automatic Document Processing*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA. [page 13, 14, 38]

Salton, G. 1989. *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1988. [page 13]

Sojka, Petr. 2005. From Scanned Image to Knowledge Sharing. In *Proceedings of I-KNOW'05: Fifth International Conference on Knowledge Management*, ed. Klaus Tochtermann and Hermann Maurer, 664–672. Graz, Austria: Know-Center in coop. with Graz Uni, Joanneum Research and Springer Pub. Co. [page 99]

Song, X., C.Y. Lin, B.L. Tseng, and M.T. Sun. 2005. Modeling and Predicting Personal Information Dissemination Behavior. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 479–488. ACM. [page 68]

Souter, C., G. Churcher, J. Hayes, J. Hughes, and S. Johnson. 1994. Natural Language Identification using Corpus-Based Models. *Hermes Journal of Linguistics* 13:183–203. [page 112]

Teahan, W.J. 2000. Text Classification and Segmentation using Minimum Cross-Entropy. In *Proceedings of the 6th International Conference "Recherche d'Information Assistee par Ordinateur", Paris, FR*, 943–961. [page 111, 112, 122, 124]

Teh, Y.W., D. Newman, and M. Welling. 2007. A Collapsed Variational Bayesian Inference Algorithm for Latent Dirichlet Allocation. *Advances in Neural Information Processing Systems (NIPS)* 19:1353. [page 32]

Tesic, J., S. Bhagavathy, and BS Manjunath. 2004. Issues Concerning Dimensionality and Similarity Search. In *Image and Signal Processing and Analysis, 2003. ISPA 2003. Proceedings of the 3rd International Symposium on*, volume 1, 272–277. IEEE. [page 19]

Tichý, P. 1969. Intension in Terms of Turing Machines. *Studia Logica* 24:7–21. [page 4]

Tichý, P. 1980. The Logic of Temporal Discourse. *Linguistics and Philosophy* 3:343–369. [page 4]

Turian, J., L. Ratinov, and Y. Bengio. 2010. Word Representations: A Simple and General Method for Semi-Supervised Learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, 384–394. Association for Computational Linguistics. [page 11]

Turney, P.D., and P. Pantel. 2010. From Frequency to Meaning: Vector Space Models of Semantics. *Journal of Artificial Intelligence Research* 37:141–188. [page 7, 8, 13, 15]

Varadarajan, J., R. Emonet, and J.M. Odobez. 2010. Probabilistic Latent Sequential Motifs: Discovering Temporal Activity Patterns in Video Scenes. In *British Machine Vision Conference (BMVC), Aberystwyth*. [page 28]

Vigna, S. 2008. Distributed, Large-Scale Latent Semantic Analysis by Index Interpolation. In *Proceedings of the 3rd international conference on Scalable information systems*, 1–10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). [page 37, 41, 68]

Wallach, H., D. Mimno, and A. McCallum. 2009. Rethinking LDA: Why Priors Matter. *Advances in Neural Information Processing Systems (NIPS-09)* 22:1973–1981. [page 32]

Wang, Y., H. Bai, M. Stanton, W.Y. Chen, and E. Chang. 2009. PLDA: Parallel Latent Dirichlet Allocation for Large-Scale Applications. *Algorithmic Aspects in Information and Management* 301–314. [page 69]

Weber, R., and S. Blott. 1997. An Approximation Based Data Structure for Similarity Search. Technical report, Report TR1997b, ETH Zentrum, Zurich, Switzerland. [page 18, 19]

Weber, R., H.J. Schek, and S. Blott. 1998. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of the International Conference on Very Large Data Bases*, 194–205. Institute of Electrical & Electronics Engineers. [page 18]

Weber, Roger, Klemens Böhm, and Hans-Jörg Schek. 2000. Interactive-Time Similarity Search for Large Image Collections using Parallel VA-Files. In *ECDL*, 83–92. Lisbon, Portugal: Springer. [page 19]

William, H., S.A. Teukolsky, and W.T. Vetterling. 1988. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge university press New York, NY, USA. [page 38]

Wilson, K.W., B. Raj, and P. Smaragdis. 2008. Regularized Nonnegative Matrix Factorization with Temporal Dependencies for Speech Denoising. In *Proceedings of 9th Annual Conference of the International Speech Communication Association (Interspeech)*, 411–414. Brisbane, Australia: ISCA. [page 27]

Yang, Yiming, and Jan O. Pedersen. 1997. A Comparative Study on Feature Selection in Text Categorization. In *Proceedings of the fourteenth International Conference (ICML'97)*, 412–420. Nashville, Tennessee: Morgan Kaufmann Pub. [page 15, 20]

Yao, L., D. Mimno, and A. McCallum. 2009. Efficient Methods for Topic Model Inference on Streaming Document Collections. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 937–946. ACM. [page 68]

Young, S.S., P. Fogel, and D. Hawkins. 2006. Clustering Scotch Whiskies using Non-Negative Matrix Factorization. *Joint Newsletter for the Section on Physical and Engineering Sciences and the Quality and Productivity Section of the American Statistical Association* 14:11–13. [page 27]

Yuret, D., and M.A. Yatbaz. 2010. The Noisy Channel Model for Unsupervised Word Sense Disambiguation. *Computational Linguistics* 36:111–127. [page 82]

Zha, H., O. Marques, and H. Simon. 1998. Large-scale SVD and Subspace-based Methods for Information Retrieval. *Solving Irregularly Structured Problems in Parallel* 29–42. [page 37, 48, 53]

Zha, H., and H.D. Simon. 1999. On Updating Problems in Latent Semantic Indexing. *SIAM Journal on Scientific Computing* 21:782. [page 38, 39, 41, 43]

Zha, H., and Z. Zhang. 2000. Matrices With Low-Rank-Plus-Shift Structure: Partial SVD and Latent Semantic Indexing. *SIAM Journal on Matrix Analysis and Applications* 21:522. [page 48]

Zhang, J., Y. Yan, and M. Lades. 2002. Face Recognition: Eigenface, Elastic Matching, and Neural Nets. *Proceedings of the IEEE* 85:1423–1435. [page 25]

Zobel, J., A. Moffat, and R. Sacks-Davis. 1992. An Efficient Indexing Technique for Full Text Databases. In *Proceedings of the 18th International Conference on Very Large Data Bases*, 352–362. Morgan Kaufmann Publishers Inc. [page 17]

A

SAMPLE CORPUS

Table A.1 shows a toy corpus of nine short documents, replicating the example from Deerwester et al. (1990). Documents c_1–c_5 relate to human-computer interaction, documents m_1–m_4 are about graphs. Words that are part of the dictionary are in *italics*; other words are either stop-words or only appear once in the corpus, and are ignored in any subsequent processing. The bag-of-words (word frequency) matrix representation of this corpus is in Table A.2.

Table A.1.: Example corpus (replicated from Deerwester et al. (1990)).

Label	**Text**
c_1	*Human* machine *interface* for Lab ABC *computer* applications
c_2	A *survey* of *user* opinion of *computer system response time*
c_3	The *EPS user interface* management *system*
c_4	*System* and *human system* engineering testing of *EPS*
c_5	Relation of *user*-perceived *response time* to error measurement
m_1	The generation of random binary unordered *trees*
m_2	The intersection *graph* of paths in *trees*
m_3	*Graph minors* IV: Widths of *trees* and well-quasi-ordering
m_4	*Graph minors*: A *survey*

Table A.2.: The same corpus as a 12×9 bag-of-words matrix $\mathbf{A}$.

	c_1	c_2	c_3	c_4	c_5	m_1	m_2	m_3	m_4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

Table A.3.: The same corpus, with each document represented by its adherence to two latent topics.

document	**topic 1**	**topic 2**
c_1	0.20	−0.06
c_2	0.61	0.17
c_3	0.46	−0.13
c_4	0.54	−0.23
c_5	0.28	0.11
m_1	0.00	0.19
m_2	0.01	0.44
m_3	0.02	0.62
m_4	0.08	0.53

(a) Latent Semantic Analysis (truncated left singular vectors)

document	**topic 1**	**topic 2**
c_1	0.87	0.13
c_2	0.92	0.08
c_3	0.90	0.10
c_4	0.90	0.10
c_5	0.87	0.13
m_1	0.25	0.75
m_2	0.17	0.83
m_3	0.13	0.87
m_4	0.13	0.87

(b) Latent Dirichlet Allocation (variational topic parameters γ)

B

DIFFERING DOCUMENTS

Below are two articles mentioned in Chapter 6.4, on which the TF-IDF and Latent Semantic Analysis similarity scores differ dramatically. Note the many OCR errors present in the text. Such low-level errors would likely make application of more refined, logic-based semantic inference methods problematic. Statistical processing proves rubost towards this type of "input noise".

1. **Czechoslovak Mathematical Journal, vol. 24 (99) 1974, Praha**

```
NEWS and NOTICES IN MEMORIAM PROF. RNDr.  KAREL
CERNY On 15 January 1974, RNDr.  Karel Cerny,
Associated Professor of Mathematics at the Czech
Technical University, died in Prague.  Prof.
Cerny was born on 6 July 1909 at Zbyslavice near
Caslav.  After completing his mathe- matical
studies at Charles University in 1933 he became
lecturer at the Faculty of Mechanical
Engineering.  He remained member of staff of the
Faculty till 1953 except for the years 1942--45
when he suffered from Nazi persecution.  In 1953
he was appointed Associated Professor (Dozent)
first at the Faculty of Architecture and later at
the Faculty of Civil Engineering of the Czech
Technical University.  Prof.  Cerny spared no
effort in his educational activity which may be
characterized by his full devotion and
responsible approach.  The scientific interest of
```

K. Cerny, who had been a pupil of Prof. V. Jarnik, was concentrated on the theory of numbers, particularly on the metric theory of diophantine approximations. A more detailed biography of Prof. Cerny is published in Cas. pest. mat. 99 (1974), 321 -- 323. Editorial Board

2. **ARCHIVŮM MATHEMATICUM (BRNO) Vol. 26, No. 2-3 (1990), 65-66**

THIS ISSUE OF ARCHIVUM MATHEMATICUM IS DEDICATED TO THE NONAGENERIAN OF ∗ ACADEMICIAN OTAKAR BORtFVKA Academician Otakar Boruvka, Nestor and legend of the Brno mathematicians, long ago one of the leaders of the Czechoslovak mathematical life, a prominent representative of our science abroad, excellent teacher and oiganizer of the scientific life was ninety on May 10, 1989. In full mental freshness, creating activity, in enviable spirit, in constant interest in mathematical events. In 1920-as a student-he passed from the Czech Technical University to the newly founded Faculty of Science of the Brno University and here he passed a state examination in mathematics and physics in 1922. From the year 1921he was a lecturer in the year 1928 he became an associate professor, from the year 1934 he was a professor assistant and’from the year 1946 (with the effectivness from the year 1940) he was a regular professoi of our faculty. From the year 1970 he is a member of the Mathematical Institute of the Czechoslovak Academy of Sciences’ in Brno. For the time being he is an author of 84 original scientific papers from the area of differential geometry, general algebra and differential equations and 50 further popular and bibliografical papers. For his results he

was awarded a State Prize of Klement Gottwald in the year 1959 and Order of Labour in the year 1965, ⟨hr id="0072"/⟩ from the year 1953 he was a corresponding member and from the year 1965 a regular member of the Czechoslovak Academy of Sciences, he is an honourable doctor of the Komensky University in Bratislava, and honourable member of the Association of the Czechoslovak Mathematicians and Physicists and he received a number of medals and diplomas of the universities and scientific associations in our country and abroad. Last but not least, he gave rise to this journal (25 yeai ago, in 1965) and was its first editor-in-chief. The rare life anniversary of the Academician Otakar Boruvka is of course associated with a numbei of summary publications in professional and popular press (e.g. Czech. Math. Journal, vol. 39 (113) 198?, 382-384). To us, belonging to the generations of his students, members of scientific seminars, founded or oriented by him, to those, inspired by his work, to his younger collaborators and colleagues and to those esteeming his character, is, however, this reality not only a reason for valorizing his admirable work but also for an oportunity to express our homage to our honoured person by the results of our works. We wish to Academician Boruvka health and good humour in ordei to be able to give away, in further years, from the treasury of his wisdom and experience.
Photo: J. France