The predictability problem

James Kwan Yau Ong

Eingereicht bei der
Mathematisch-Naturwissenschaftlichen Fakultät
der Universität Potsdam

July 2007

Acknowledgements

I thank Professor Jürgen Kurths for accepting me into his group as his doctoral student and providing an environment conducive to learning about the application of nonlinear dynamics in many diverse fields. His group, the Arbeitsgruppe für Nichtlineare Dynamik (AGNLD) in the Physics Department of the University of Potsdam, is truly a melting pot for ideas from many different traditions, and while most of the research performed in this group has nothing to do with my research, I have greatly profited from the many lively academic discussions with other members of the group.

I thank Professor Reinhold Kliegl for accepting me with open arms into his Cognitive Psychology research group. From the first day, I never felt like an outsider in the group. His encouragement and guidance has been very welcome, while never infringing on my independence as a researcher. The weekly forum for students to present and discuss their work has been an invaluable learning experience, particularly for one without an extensive background in cognitive psychology.

I thank Dr. Alexander Geyken for allowing me to use the computing facilities of the Arbeitgruppe "Das Digitale Wörterbuch der deutschen Sprache des 20. Jahrhunderts" to access and preprocess the ZEIT corpus.

I thank the International Max Planck Research School on Biomimetic Systems for supporting me financially, and for exposing me to the application of biological principles to other areas.

Of course, I thank all of the students, researchers and support staff in the many groups for their insights, help and friendship. It has been a pleasure to work with all of you. I must thank Lucia specifically for showing me that sometimes, you just have to slog through what confronts you before you can do what interests you.

Thanks to all those in and outside of Berlin who have continued to keep in touch with me in spite of the horrendously poor efforts on my part, all those who believed in me from the start, all those who were interested in my research, all those who have kept me in their prayers. I especially acknowledge my parents and sister, who have provided all sorts of support in the last three and a half years.

Last, but of course not least, I must thank my lovely wife Ellen for everything that she is and has done for me. Thanks for putting up with my working habits, including the strange hours and the piles of paper scattered all around. Thanks for discussing linguistics with me, even when you were too tired to do so. And thanks for your patience with me, even though I was 'almost finished' for many months. You're the best.

Abstract

We try to determine whether it is possible to approximate the subjective Cloze predictability measure with two types of objective measures, semantic and word *n*-gram measures, based on the statistical properties of text corpora. The semantic measures are constructed either by querying Internet search engines or by applying Latent Semantic Analysis, while the word *n*-gram measures solely depend on the results of Internet search engines. We also analyse the role of Cloze predictability in the SWIFT eye movement model, and evaluate whether other parameters might be able to take the place of predictability. Our results suggest that a computational model that generates predictability values not only needs to use measures that can determine the relatedness of a word to its context; the presence of measures that assert unrelatedness is just as important. In spite of the fact, however, that we only have similarity measures, we predict that SWIFT should perform just as well when we replace Cloze predictability with our measures.

Zusammenfassung

Wir versuchen herauszufinden, ob das subjektive Maß der Cloze-Vorhersagbarkeit mit der Kombination objektiver Maße (semantische und *n*-gram-Maße) geschätzt werden kann, die auf den statistischen Eigenschaften von Textkorpora beruhen. Die semantischen Maße werden entweder durch Abfragen von Internet-Suchmaschinen oder durch die Anwendung der Latent Semantic Analysis gebildet, während die *n*-gram-Wortmaße allein auf den Ergebnissen von Internet-Suchmaschinen basieren. Weiterhin untersuchen wir die Rolle der Cloze-Vorhersagbarkeit in SWIFT, einem Modell der Blickkontrolle, und wägen ab, ob andere Parameter den der Vorhersagbarkeit ersetzen können. Unsere Ergebnisse legen nahe, dass ein computationales Modell, welches Vorhersagbarkeitswerte berechnet, nicht nur Maße beachten muss, die die Relatiertheit eines Wortes zum Kontext darstellen; das Vorhandensein eines Maßes bezüglich der Nicht-Relatiertheit ist von ebenso großer Bedeutung. Obwohl hier jedoch nur Relatiertheits-Maße zur Verfügung stehen, sollte SWIFT ebensogute Ergebnisse liefern, wenn wir Cloze-Vorhersagbarkeit mit unseren Maßen ersetzen.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview of the dissertation

In this dissertation, we consider two hypotheses:

1. Predictability can be generated computationally by combining two simple measures; the first represents semantic relationships, while the second captures short-range morphosyntactic constraints.

2. The form of the lexical processing component of SWIFT, a model that simulates eye movements during reading, can be deduced from real data.

The rest of this chapter contains the motivation for this research and an outline of the measures that we use to study these hypotheses. Chapter 2 details the creation of our semantic measures and then shows a comparison of them to predictability. Chapter 3 is similar, except that it deals with our word *n*-gram measures. Chapter 4 considers whether a combination of semantic and word *n*-gram measures can be used to generate predictability values. In Chapter 5, we explore a method to use real data to deduce the form of the lexical processing component of SWIFT, and in addition, consider whether our derived measures may be useful as alternatives to predictability. Finally in Chapter 6, we bring all the threads together and consider the implications of this research.

1.2 What is predictability?

Predictability is an experimental estimate of a person's ability to predict what is coming up, based on what has come before. The ability to generate correct predictions is reliant on many factors, including a person's specific knowledge and the entropy of the signal.

To generate a quantitative estimate of the predictability of target words during reading, one can use the Cloze task [62]. In this task, a large number of subjects are asked to guess target words given a previous text context. The Cloze predictability value is the proportion of subjects who correctly guess the target word. Equivalently, one can use the responses to build a probability distribution representing the chance that a certain word occurs, given the previous text context. The Cloze predictability measure summarises the many components of predictability into a single probability estimate; this measure specifies the chance that a subject will correctly predict the target word, given the preceding incomplete text context. We will be using Cloze predictability as our quantitative representation of predictability.

In this dissertation, we will be using the Cloze predictability norms collected for the Potsdam Sentence Corpus [33] as a yardstick to allow us to assess our attempts to model predictability. Importantly, predictability norms were collected for every word in the corpus, and not just for a subset of target words.

1.3 Why do we want to compute predictability?

1.3.1 Predictability is useful

In our modern society, the application of predictability has significantly affected the way that most of us live our daily lives. Predictive text on mobile phones, speech recognition, and targeted marketing based on individual shopping patterns are all very direct illustrations of the power of applied predictability. Most applications use rather simple methods to generate predictability estimates, but these crude estimates often extract a surprisingly large amount of information from the raw data.

The applications of predictive text and speech recognition use predictability estimates to guess upcoming words in the face of uncertainty. If we had a way to generate perfect Cloze predictability estimates given a certain context, we would then be able to automatically complete a text fragment in a sensible way. A ‘perfect’ predictive text could suggest words even without the user needing to enter the first few letters, while ‘perfect’ speech recognition would be able to decipher unclear utterances and disambiguate homophones. In addition, such models should also be able to detect syntactic and semantic anomalies.

Another way of using predictability is to consider it to be an estimate for the difficulty or unexpectedness of the current word given the previous context. A highly difficult word is likely to have a low predictability, while an easy word (given the context) is likely to be somewhat predictable. This interpretation is especially suited for the domain of cognitive psychology, where difficulty can be seen as contributing to cognitive load. Predictability has proven to be a useful attribute in the modelling of reading behaviour, since it captures important features that are not captured in simpler first-order word attributes such as frequency, length or sentence position [23, 34]. The fact that we try to anticipate upcoming words is often painfully obvious when we listen to a person speaking unusually slowly.

A far more ambitious use of an autonomously generated predictability estimate occurs in the research area of artificial intelligence. Going back to the roots of artificial intelligence, we come across the question: “Can machines think?” As one means of addressing this question, Turing proposed the “Imitation Game” [63] (now more commonly known as the “Turing Test”), in which a computer is judged on its ability to respond to questions in a human-like manner. An alternative test that is more relevant to the current application, the Editing Test, was proposed by Collins [16] in an attempt to get to the core of the issue, while circumventing many of the tricks that are commonly used to fake understanding of language. Collins proposed that a computer be judged on its ability (relative to a human control) to subedit previously unseen passages of incorrect English. A program that were able to generate predictability values for a sentence would significantly contribute to the ability of a computer to correctly detect errors in text passages, which is the first requirement to pass the Editing Test.

1.3.2 Predictability is difficult to collect

The Cloze procedure is obviously a time and manpower intensive way to derive quantitative estimates of predictability, especially if one wants to end up with statistically stable probability estimates. Some researchers have attempted to reduce the overhead needed to collect predictability values by choosing only one target word per sentence (see [51] for one example), but this severely reduces the utility of the underlying text. In practice, if one considers word predictability to be an important variable in a psychophysical experiment, one is limited to using text that has previously been designed for some other purpose, possibly limiting important experimental manipulations of sentence construction and content.

It would be far cheaper and more convenient to use a method to estimate predictability that did not require the collection of data within an experimental context. Ideally, such a method should also function automatically, without the need for a large amount of human input and without the need for excessively powerful computers.

1.4 A computational model of predictability

Intuitively, the measure of predictability seems to be heavily dependent on a deep understanding of the structure of a language. Adult subjects completing a Cloze task in their native language generally have no trouble in putting forward suggestions that are both grammatically correct and somehow topically related to the context. In addition, they can pick up on cues or idiomatic constructions to propose words that somehow sound ‘normal’. Firth [25] summed up this intrinsic knowledge with the famous remark: “you shall know a word by the company it keeps”.

By creating a computational model that generates predictability estimates and noting the essential elements of the model, one can test the assertion that predictability reflects a deep understanding of language. Griffiths, Steyvers, Blei and Tenenbaum [28] show that a sophisticated generative model that incorporates both morphosyntactic and semantic dependencies can predict many of the fundamental properties of individual words. We will explore in the rest of this dissertation whether it is possible to approximate predictability with far simpler metrics that capture some of the effects of semantics and morphosyntax.

1.4.1 Semantics

The aim of a semantic model is to allow a computer to extract the relevant content or the ‘meaning’ of its input; this ‘understanding’ can then be used to make the computer appear to be smart. Increasingly, web search engines[1] are attempting to use semantic models to anticipate the meaning of search terms in order to improve the relevance of the results given. Indeed, the range of readily-available commercial programs[2] reflects the success of semantic models at extracting ‘knowledge’ from large data stores.

The simplest way to build a semantic model is to create categories (sometimes hierarchical in structure) and then allocate words to each category. Both the creation of categories and the allocation of words to categories may be entirely

[1]A Google search with the search terms “semantic search engine” produces many examples of search engines that incorporate semantic typing; two examples can be found at `http://www.cs.umd.edu/projects/plus/SHOE/search/` and `http://www.semanticwebsearch.com/`. A number of groups that have demonstrated success in integrating semantic modelling into the search paradigm are now fully in the commercial arena, like `http://www.previewseek.com/`; others are supporting an open source approach, like `http://www.alvis.info/`.

[2]A recent article [35] in the popular computer magazine *c’t* is dedicated to the topic of smart search software. All of these software packages first generate a semantic space or network, and then attempt to cluster together concepts that are close together.

or partly dictated by human knowledge, or may be generated automatically from training texts.

In the last few years, there have been many papers outlining more sophisticated ways of capturing semantic relationships. These have included neural networks [42], semantic networks [24, 60] and semantic spaces [40, 59]. After the semantic model is constructed, a similarity metric can be defined between elements, often so that a small distance in the metric implies high similarity. Simple semantic similarity metrics can be either first-order or second-order measures.

First-order semantic measures

First-order semantic measures are based upon the co-occurrence of words. One example of a first-order measure is a synonymity measure, where words are said to be close together if they co-occur in a thesaurus entry; equivalently, one can use a database like WordNet [24]. Using such a measure, one can build a semantic network, in which words are represented by nodes and related words are joined by an arc; once such a network has been constructed, one possible first-order measure is network distance. One problem in generating a synonymity measure is that the underlying resource, like a dictionary or thesaurus, must already exist in the required language. In addition, the translation process from the native format to one useful to the researcher may be rather involved.

Another first-order semantic measure is the probability that two words co-occur on an Internet web page. Surprisingly, the extremely large amount of source data seems to overcome the naïvety of the measure and the inherent noisiness of data drawn from the Internet [64]. Internet search engines allow researchers to access this data quickly and easily, and the provision of Application Programming Interfaces by the major search engines has made bulk automated querying a realistic proposition. Due to the ease of collecting and calculating such a measure, we will use this as our exemplar of a first-order semantic measure.

Second-order semantic measures

Second-order semantic measures attempt to take into account indirect relationships between words. The main idea is that synonyms are unlikely to occur in the same sentence, and thus should not be easy to derive by looking for simple co-occurrences in a text corpus. Instead, one is likely to find similar words in documents that share the same topic. However, if one can allocate the same topic to different documents, then the semantically-important words occurring in these documents should be somehow related.

Latent Semantic Analysis, described below in Section 2.2, places words into a semantic space, where similar words are clustered together by a dimension reduction step. The angle between word vectors then serves as a second-order semantic measure. Our major reason for choosing Latent Semantic Analysis over other methods, like non-negative matrix factorisation [39, 10], is the existence of heuristic methods to perform the procedure approximately; the need for such heuristics is necessitated by the large amount of data that we would like to process.[3]

1.4.2 Idiomatic constructions

One of the main distinguishing features of language use between native and foreign speakers is the use of idiomatic and common combinations of words. Naturally, the presence of such constructions greatly constrains sentence structure, which should result in far higher predictability values than one would expect just from word-based semantic and morphosyntactic considerations.

A natural way to capture such short-range sentence structure is to use word *n*-gram models, described in Section 3.1. One motivation for doing this is some evidence presented by McDonald and Shillcock [44], where they show that eye movements during reading reflect the transitional probabilities between words. Word *n*-gram models must be used carefully, since they exhibit the common problem of having sparse training data. In addition, the purest form of a word *n*-gram model fails to capture long-range idiomatic relations such as the separable constructions commonly found in German[4]. One way of dealing with idiomatic expressions that exceed the span of a word *n*-gram model is to use a skipping word *n*-gram model as mentioned by Goodman [27], but it is not clear how one might select an appropriate skipping pattern.

We will use naïve word *n*-gram models to attempt to capture the effect of idiomatic constructions.

1.4.3 Morphosyntax

The morphosyntactic requirements of a language limit the possible form and syntactic category of words. In German, the form of a word is often highly dependent on its context. However, predictability is highly divorced from pure syntax.

[3]One example given in Berry et al. [10] exemplifies the problem. They apply non-negative matrix factorisation to their Enron corpus to try to classify 289,695 messages into 50 categories, on the basis of a 7,424-word vocabulary. The procedure takes them about 20 minutes to perform, which is roughly comparable to Singular Value Decomposition. In our case, the starting matrix is more than two orders of magnitude larger; even if the procedure scaled up only linearly with size, it would take weeks to calculate the result.

[4]Many idioms split into two parts. One such idiom, “Rolle spielen”, which means “to play a role”, must in some cases split, because of syntactic principles, so that the verb “spielen” is moved forward in the clause, which could be arbitrarily far away from the noun “Rolle”, which remains at the end.

Chomsky [12] contends that it is incorrect to replace “zero probability, and all extremely low probabilities, by *impossible*, and all higher probabilities by *possible*”. He illustrates his point with the example context “I saw a fragile—,”, where the word “whale” is a grammatically correct but highly unpredictable completion, while the word “of” cannot be used to complete the sentence in a grammatical way; the two words may both have predictability zero, although one is allowed and the other is not. This example demonstrates that predictability is only loosely related to acceptability.

Let us, however, for a moment assume that we have a predictive parser that perfectly tells us the possible syntactic categories of the upcoming word and their associated probabilities. Taking Chomsky’s example again, we should now be able to say that “of” must have zero predictability since the probability of a preposition in this context is zero. However, we still have no information telling us that “whale” should also be unpredictable, since a noun works perfectly as a completion. This suggests that we may be able to use syntactic information to predict zero predictability values of ungrammatical instances; this does not help us further, since our corpus is, by design, grammatically correct.

It also makes no sense to use syntactic information to smooth a word *n*-gram model in a simple way by replacing unseen word *n*-grams by the equivalent *n*-gram containing only word class. For example, the 4-gram “I saw a fragile” could be recoded as “Pronoun Verb Article Adjective”. Then, assuming that “whale” has never been seen in the training corpus, we would look at the probability that “Pronoun Verb Article Adjective” was followed by “Noun”. This probability tells us nothing about predictability when the completion is syntactically valid, and we cannot even naïvely use it as an estimate of an upper bound for predictability[5].

In this dissertation, we will not explicitly attempt to accommodate morphosyntax. The use of a word *n*-gram model may capture short-range syntactic constraints well enough to negate much of the contribution provided by an explicit syntactic model. The only obvious void is the ability to predict long-range syntactic constraints, which occur frequently in German; these may be accommodated, for example, by incorporating predictions from a grammar that has a transformational rule allowing a terminal verb to move forward in the sentence. Bellegarda [8] addresses this issue in a brief review of the use of syntactic models for “syntactically driven span extension”. Such a model can be added to our considerations without too much difficulty.

[5]A simple illustration of this, taken from our test sentences, is the trigram “(start of sentence) Man kann”: The predictability of “kann” is high, but the probability that a personal pronoun at the start of the sentence is followed by a modal verb is more than an order of magnitude lower.

1.5 Do we really need predictability?

A totally separate question to those already posed is whether it is actually necessary to compute predictability. In many applications where predictability could be useful, it may actually be better to use a more specific, less all-encompassing measure than predictability, especially if such a measure does not rely on explicit human input. In the case of eye movement research, word predictability has been used as a predictor because it captured lexical features that were not described by the other predictors of word frequency and length, but also because there were no other available predictors that could explain the difficulty of reading a word in its semantic and syntactic context. In Chapter 5, we consider what the role of predictability should be in the SWIFT reading model, and whether our derived measures may be useful alternatives to predictability.

1.6 Chapter summary

Predictability is a useful parameter to have, but it is difficult to collect. This motivates us to ask whether is it possible to compute estimates of predictability. We will explore whether we can approximate predictability by using a combination of simple measures that capture features of semantics and morphosyntax. We use two types of semantic measures: the first is derived from Internet co-occurrence frequency counts, while the second requires the application of Latent Semantic Analysis to a text corpus. We hope to accommodate short-range constraints by using a word *n*-gram measure. As well as doing a direct comparison between our measures and predictability, we will look at the role of predictability in a reading model, and consider whether there are other candidates for this role.

Chapter 2

Semantic measures

In this chapter, we will outline two methods for generating semantic measures, and then examine the relationship of these two measures to predictability. The first method produces a first-order semantic measure based on Internet web page co-occurrence counts, while the second generates a second-order measure based on the method of Latent Semantic Analysis applied to a newspaper corpus.

2.1 Web co-occurrence measures

The Internet is one of the largest training data sets available to researchers in language research. Two important results motivate us to want to use the Internet as a source of training data for language models: Banko and Brill [7] noted that the performance of simple algorithms continues to improve as the size of the training data increases, and Keller and Lapata [31] concluded that the size of the data set outweighs the noisy and unbalanced nature of the data.

2.1.1 Two possible measures

Pointwise mutual information

Turney [64] compared the performance of two methods to correctly recognise synonyms: the first was based on a simple web-based pointwise mutual information measure, while the second was based on Latent Semantic Analysis (see Section 2.2). He found that the web-based method does at least as well as the other method and concludes that the amount of web training data compensates for the simplicity of the mutual information measure.

Following Turney, we adopt pointwise mutual information as a possible first-order semantic measure. For a pair of words w_i and w_j, the pointwise mutual information PMI_{ij} is defined as

$$\mathrm{PMI}_{ij} = \log_2 \left(\frac{p(w_i \cap w_j)}{p(w_i)p(w_j)} \right), \tag{2.1}$$

where $p(w_i \cap w_j)$ is the probability that words w_i and w_j co-occur in the same webpage. If the two words w_i and w_j occur statistically independently of each other, $p(w_i \cap w_j) = p(w_i)p(w_j)$, and thus $\mathrm{PMI}_{ij} = 0$.

We can rewrite the definition of pointwise mutual information in terms of frequency counts:

$$\mathrm{PMI}_{ij} = \log_2 \left(T \frac{f(w_i \cap w_j)}{f(w_i)f(w_j)} \right), \tag{2.2}$$

where T is the total number of web pages being searched. In practice, it is not clear how many web pages are being searched in total, so we approximate this with the frequency of the most common word.

Conditional co-occurrence probability

The other first order semantic measure that we will consider is the simple conditional probability of a word dependent on a previous word in the context:

$$p(w_j|w_i) = \frac{p(w_i \cap w_j)}{p(w_i)} = \frac{f(w_i \cap w_j)}{f(w_i)}. \tag{2.3}$$

This measure, in contrast to pointwise mutual information, is asymmetrical, reflecting the fact that semantic relationships are often asymmetrical. In the case that words w_i and w_j occur statistically independently of each other, the conditional probability $p(w_j|w_i)$ reduces to $p(w_j)$.

2.1.2 Comparison between the different search engines

The Google, Yahoo! and Microsoft (MSN) search engines provide Application Programming Interfaces (APIs) allowing queries to be sent from the command line. At the moment, the daily number of queries per user (regulated by the provision of individual application keys and/or counting the number of queries per IP address) is limited to 1,000, 5,000 and 10,000 queries respectively. Results from the Yahoo! search engine are currently partly drawn from Google, while MSN Search appears to be independent.

The Google and MSN APIs use the Simple Object Access Protocol (SOAP), while the Yahoo! API follows the Representational State Transfer (REST) philosophy; however, in reality, search requests to all of the APIs are submitted via Remote Procedure Call (RPC), where a request created on the local machine causes a remote machine to execute a specified procedure and then return a result message. Creating RPCs in the correct format is a relatively simple matter, and example scripts are either distributed with the API or can be found on the respective user forums dedicated to each API.

In an attempt to make the web-based corpus more relevant, we limited each search to pages in German but did not restrict searches by the country of the server hosting the web page. Although a multitude of information can be supplied in each search result, we were only interested in the estimated number of hits corresponding to the search request.

2.1.3 Practical issues with querying search engines via API

There are a number of issues to be considered when submitting a query to an API:

- An API will occasionally fail to return a valid result to a request (for example, it may time out), meaning that missing requests must be resubmitted at a later time.
- All special characters, such as letters with umlauts and the `ß` character in German, need to be replaced with standard ASCII equivalents. For example, `ä` needs to be replaced with `ae`, `ß` needs to be replaced with `ss`.
- Searches are case insensitive.
- Results are encoded as 32-bit signed integers, meaning that values up to $3^{31} - 1$ can be accommodated. The Yahoo! and MSN APIs replace larger values than this with the maximum value (2147483647), while the Google API instead fails to return a valid response.

Google

Google Search seems to have an anomaly in the way that it estimates the number of hits. There is a suspicious lack of results in the frequency range between 1,000 and 10,000, independent of query type or corpus language[1]; this discrepancy can be seen in Figure 2.1. It may be that Google uses two algorithms for estimating hits, one for high values and one for low values. Results from the MSN search engine do not provide corroborative evidence that the lack of frequencies between

[1]To test whether the missing frequency range was an anomaly or a real finding, we queried the Google API to obtain frequency estimates for the Schilling corpus [58], which is in English. Since the same lack of results between 1,000 and 10,000 occurs for an English corpus, although frequency estimates in English are generally a number of orders of magnitude greater than in German, we attribute the missing frequency range to an anomaly in the Google frequency estimation algorithm.

1,000 and 10,000 is real, but the frequency range to be verified is so low that clean corpora are too small to provide evidence one way or the other.

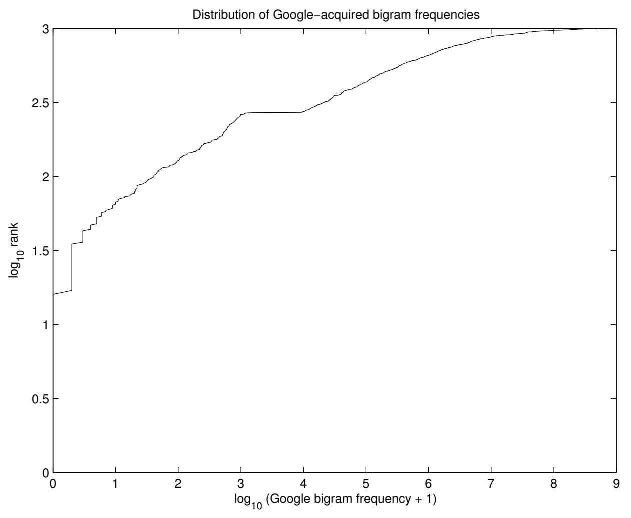

Figure 2.1: A sample plot of the distribution of frequency estimates (in this case, of word bigrams in the Potsdam Sentence Corpus) returned by the Google API. A plateau spanning the frequency range between 1,000 and 10,000 is clearly visible.

In addition, the estimated number of hits for a particular query may vary depending on the particular moment at which the query was sent. After a little testing, it appeared that there could be at least five different values for each query, varying by up to an order of magnitude. This creates problems because it means that it is possible for estimated word bigram frequencies to be *higher* that the estimated word unigram frequencies of the two words in the word bigram.

Going by the many annoyed posts on the Google Search API forum, the Google Search API is not being updated and no support is being provided. Thus, it looks unlikely that Google will resolve these problems in the near future. In September 2006, Google officially released word *n*-gram statistics (up to $n = 5$) for English [2], but has not yet done the same for other languages.

Yahoo!

Because Yahoo! sources some of its search results from Google, their API search results suffer from the same anomalies that are found in the Google API, although less so.

MSN

Language restriction via the MSN API, as specified in the `CultureInfo` field of the `SearchRequest` class, is rather idiosyncratic, since it is not explicitly possible to specify the language German; instead, one must limit requests to a 'culture' like German–Germany (`de-DE`) or German–Austria (`de-AT`). Practically, however, the language restriction submitted in the search query seems to make very little difference, as highlighted by the lack of real difference between results with the `en-US` and the `de-DE` culture restrictions. This suggests that language restriction does not currently work correctly in the MSN API. In spite of this issue, we chose to use the `de-DE` culture restriction, since the `CultureInfo` field cannot be left blank in the search request.

When queried for the frequency of single words, the MSN API sometimes returns significantly less hits than when the word is part of a multiword query. This obviously indicates some fundamental error in the underlying estimation algorithm.

2.2 Latent Semantic Analysis measure

Latent Semantic Analysis[2] is a technique that was originally created to allow the automatic indexing of documents by topic or meaning [17, 11]. There has been so much interest in the research community that Landauer et al. have published a book [38] specific to the topic; as well as addressing technical issues and decisions peculiar to Latent Semantic Analysis, the authors suggest ways to apply and extend the method.

Perhaps one of the most controversial claims about Latent Semantic Analysis was made by Landauer and Dumais [36], who say that the method may reflect the human process of the "acquisition, induction and representation of knowledge". Steyvers and Tenenbaum [60] throw doubt upon this claim by showing that the final semantic space has far fewer isolated words (words with few close neighbours) than empirically-derived human semantic representations—in network terminology, a semantic network generated through Latent Semantic Analysis is small-world but not scale-free; however, they propose no automated method to create a semantic network that *is* scale-free.

In spite of the problem that Latent Semantic Analysis does not seem to produce a 'human' representation of knowledge, it is surprisingly good at automatically clustering together concepts with similar meaning. Papadimitriou, Raghavan, Tamaki and Vempala [47] give a mathematical argument to explain why the

[2]Latent Semantic Analysis is also called Latent Semantic Indexing; these are abbreviated in the literature as LSA and LSI respectively.

procedure is so successful at clustering together semantically similar concepts. The utility of the method is reflected by the multitude of commercial applications of Latent Semantic Analysis that have been patented[3]. Naturally, some variant of Latent Semantic Analysis is probably also used by the major search engines to classify websites[4].

Initially, one forms a matrix, usually sparse, containing the number of occurrences of every token in every document. For the case of m tokens and n documents, this occurrence matrix would have size $m \times n$. Equivalently, each token is described by a vector of length n, while each document is described by a vector of length m.

After obtaining the raw occurrence matrix, one needs to weight the matrix to improve retrieval performance. Dumais [20] recommends the combination of a local and a global weighting: Local weighting reduces the bias introduced by the large range of raw term frequencies, while global weighting reduces the effect of tokens that occur in many documents. The local weighting used by Landauer and Dumais [36] is the log of the raw token frequency, and an entropy-based weighting scheme is used as the global weighting. However, this combination does not satisfactorily depress the importance of function words in our particular case, since there is a very large range of token frequencies in our source corpus. To better address this problem, we choose to use the binary local weighting mentioned by Dumais [20], where raw token frequencies are replaced with a binary coding representing the occurrence of a term in a document. For our global weighting, we follow a proposal by Lowe [40] to transform each entry to the log of its odds ratio.

The heart of the Latent Semantic Analysis technique is Singular Value Decomposition[5], which is used to extract the principal components of the data. The most important principal components are kept, while the rest are discarded in order to reduce the dimensionality of the representation space. The aim of the dimension reduction step is to cluster together data points that are somehow similar.

In the newly reduced space, tokens are considered to be similar to each other if the angle between their representative vectors is small, irrespective of the length of the vectors. Mathematically, one finds the cosine of this angle by taking a normalised dot product; we will refer to this similarity measure as the *LSA measure*.

[3]A search of US Patents on `http://www.freepatentsonline.com/` with the search phrase "latent semantic indexing" reveals more than a hundred patents that refer to the method.

[4]As an example, in 2003, Google acquired Applied Semantics [4], a company specialising in semantic text processing, and currently uses its AdSense product to deliver targeted advertising.

[5]Singular Value Decomposition, often abbreviated as SVD, is the mathematical technique of splitting a matrix into singular values and vectors. Principal Component Analysis is the application of Singular Value Decomposition to a covariance matrix. We adopt some nomenclature from Principal Component Analysis because this context is more likely to be familiar to experimentalists and statisticians.

Another way of visualising this is as follows: Imagine placing the earth at the origin, with each of the tokens being stars in the sky; then similar tokens will appear to be close together in the sky to an observer on the earth. In Section 2.2.6, we discuss the inherent problems of defining 'closeness' in this way.

2.2.1 Preprocessing the source text

The text used is a selection of content taken from "DIE ZEIT", a German weekly newspaper. An electronic version was made available through the Arbeitsgruppe "Das Digitale Wörterbuch der deutschen Sprache des 20. Jahrhunderts" (DWDS), a project of the Berlin–Brandenburg Academy of Sciences (Berlin–Brandenburg Akademie der Wissenschaften); approval for scientific use of the content was given by Mr. Peter Buhr at "DIE ZEIT".

Most of the content is drawn from the period between 1996 and mid-2005, with the addition of articles drawn intermittently (roughly one month per two years) dating back to the founding of "DIE ZEIT" in 1946.

Collating the text

After copying all the available data into a local directory, inspection showed that the raw text would need to be extracted from XML files with irregular filename extensions. Details of the collation process are found in the three files `collect`, `preproc1` and `preproc2` found in Appendix A.1. The major steps in the process are to:

- make a list of all files that could possibly contain text of interest, excluding those that had a high proportion of nonsense words (e.g. chess articles) and those that contained the small print about the day-to-day running of "DIE ZEIT" (e.g. how to subscribe, who the editors were),
- combine all candidate files into one large file, so that paragraphs (everything occurring between the XML tags `<p...>` and `</p>`) are segregrated from the surrounding text by newline characters,
- extract paragraphs,
- remove or replace problematic characters, and
- remove XML tags and, if appropriate, the content between the tags.

Cleaning and preparing the text

The file resulting from the collection process, `master.txt`, now has the right form, with one paragraph on each line, and it is likely that most of the content is the training text of interest. However, the text still has strange characters present that need to be dealt with. In addition, we need to make key decisions that always need to be made when dealing with a text corpus, including considering what to do with punctuation, capitalisation and numbers.

The file `mkmaster1` contains all the necessary commands to clean the corpus. The first major output file, `master.newtxt`, can be considered to be a cleaned version of the corpus, where punctuation has been dealt with appropriately. The two files produced subsequently, `master.lowertxt` and `master.nonum` reflect the decisions to remove capitalisation[6] and numbers from the text corpus.

The cleaning process consists of the following steps:

- remove control characters,
- deal with special character encodings (including XML and Unicode encodings),
- remove World Wide Web addresses (including HTTP, FTP and email addresses),
- remove certain punctuation and special characters,
- deal with abbreviations, initials and other sequences containing full stop characters,
- remove certain punctuation sequences, and
- deal specially with full stops, commas and question marks.

We will refer to the resulting cleaned text as the Zeit corpus.

Conversion into the appropriate format

To create the LSA measure, we require an occurrence matrix detailing the frequency of each token (individual word form) in each paragraph. The file `mkmaster2` produces all the files required for both calculations.

[6]Capital letters in German are highly linked to the word class, but they contribute far less to semantic content. Since it is unlikely that they would contribute anything at all to the semantic clustering carried out by Latent Semantic Analysis, capitalisation was removed to reduce the sparsity of the source data.

Initially, the CMU toolkit [13], self-customised to increase the maximum allowable vocabulary size, is used to generate a vocabulary and collect the frequencies of tokens. The specific executables taken directly from the CMU toolkit were:

- `text2wfreq`, which creates the token frequency list; and
- `wfreq2vocab`, which creates the vocabulary file of tokens.

After this initial processing, we create a 'numerised corpus' by allocating a unique number to every token in the vocabulary and then replacing every token in the text by its corresponding vocabulary number. This conversion process, though simple in principle, is time consuming and requires one to program with memory limitations in mind, due to the size of the corpus and the vocabulary.

Before we can import this numerised corpus into MATLAB, a couple of other small steps are required, including replacing newline characters with line numbers. The file `mkmatlab` contains the commands to carry out these preliminary steps.

2.2.2 Creating the initial term–document matrix

The first step in the Latent Semantic Analysis procedure is to create a term–document matrix, where each entry in the matrix is the raw occurrence frequency of a term (normally a token) in a document. Of course, not all terms contribute to the semantic content of a document, but Quesada [49] suggests that it is unnecessary to manually remove function words, since weighting steps (like those described in Section 2.2.3) should "take care of those high-frequency words". The MATLAB code used to create the initial term–document matrix can be found in Appendix B.1.

Since the Latent Semantic Analysis procedure compares the occurrence pattern of terms in different documents in order to characterise each document, we can remove terms that only occur in one document without altering the principal components of the data. Similarly, we can remove documents that only continue one unique term, because these tell us nothing about the relationship between different terms. The repeated application of these culling rules reduces the size of our term–document matrix without reducing the information content. The implementation of this procedure can be found in Appendix B.2.

2.2.3 Weighting the term–document matrix

Binary coding

The application of binary local weighting causes all term frequencies that are nonzero to be replaced with 1. Without this conversion, fluctuations in the occurrence frequency of function words in different documents contribute to a large spurious increase of their final ‘semantic content’. In other words, binary weighting stops Latent Semantic Analysis from sorting documents into categories like “contains many function words” and “contains few function words”.

Log odds

If f_{ij} is the raw (binary-coded) frequency of the ith term in the jth document, then the odds ratio θ_{ij} can be calculated in the following way:

$$\theta_{ij} = \frac{f_{ij} \sum_{(k,l)\neq(i,j)} f_{kl}}{\sum_{k\neq i} f_{kj} \sum_{l\neq j} f_{il}}. \tag{2.4}$$

The transformed values now reflect how much the frequency of a certain term in a certain document deviates from its appearance in the rest of the corpus. We use the logarithm (base 10) of this odds ratio as our global weighting. A term that occurs very specifically in only certain documents has a significantly positive log odds ratio, while a term that occurs relatively unspecifically has a log odds ratio near zero.

2.2.4 Traditional Singular Value Decomposition

Every matrix $A \in \mathbb{R}^{m\times n}$ can be decomposed as

$$A = U\Sigma V^T \tag{2.5}$$

where $U = [u^1 u^2 \dots u^m]$ is $m\times m$ and unitary[7], $V = [v^1 v^2 \dots v^n]$ is $n \times n$ and unitary, and Σ is $m \times n$ and ‘diagonal’ in that $\Sigma_{ij} = 0$ unless $i = j$. The three matrices U, V and Σ constitute the Singular Value Decomposition of A. The real nonnegative diagonal elements of Σ are called the singular values of A, while the columns of U and V are called the left and right singular vectors of A respectively. The number of nonzero diagonal elements of Σ is the rank of matrix A.

One way of carrying out a Singular Value Decomposition is to find the eigenvalues and eigenvectors of AA^T and A^TA, since

$$\begin{aligned} AA^T &= U\Sigma V^T V\Sigma^T U^T = U\Sigma\Sigma^T U^T = \sum_{t=1}^{r} \sigma_t^2 u^t {u^t}^T, \text{ and} & (2.6)\\ A^TA &= V\Sigma^T U^T U\Sigma V^T = V\Sigma^T\Sigma V^T = \sum_{t=1}^{r} \sigma_t^2 v^t {v^t}^T, & (2.7)\end{aligned}$$

[7]A matrix $P \in \mathbb{R}^{p\times p}$ is unitary if $P^{-1} = P^T$, so that $PP^T = P^TP = I_p$, where I_p is the p-dimensional identity matrix.

where r is the rank of A.

If we take the submatrix $\Sigma_r \in \mathbb{R}^{r\times r}$ of Σ that only includes the r nonzero singular values, and the submatrices U_r and V_r containing the corresponding r left and right singular vectors, then

$$A = U_r \Sigma_r V_r^T = \sum_{t=1}^{r} \sigma_t u^t {v^t}^T. \tag{2.8}$$

This sum reflects the contributions of each of the principal components of A. In the dimension reduction step of Latent Semantic Analysis, it is necessary to discard lesser terms of this sum. If $k \leq r$ and we define

$$A_k = U_k \Sigma_k V_k^T = \sum_{t=1}^{k} \sigma_t u^t {v^t}^T, \tag{2.9}$$

where we retain the terms containing the k largest singular values of A, then A_k is the projection of A onto the space spanned by the top k singular vectors of A. In addition, A_k is an optimal k-rank approximation to A, in that it is the closest rank k matrix to A under the Frobenius norm[8].

2.2.5 Fast Monte Carlo Singular Value Decomposition

Landauer, Foltz and Laham [37] mention that "it is still impossible to perform SVD on the hundreds of thousands by tens of millions matrices that would be needed to truly represent the sum of an adult's language exposure". In his tutorial on Latent Semantic Analysis, Quesada [49] misleadingly dismisses the problem of memory limitations, saying that "nowadays the memory bottleneck is no longer an issue, since a consumer-level PC can be configured with more than enough memory to run a large SVD"; however, that is only strictly true for certain types of problems with a small vocabulary of terms, such as identifying a synonym from a list of candidates [36].

In the current application, the non-specific nature of the problem means that we cannot substantially prune the source corpus or focus in on certain sections of it. The occurrence matrix derived from the rather modestly sized Zeit corpus has 428,132 rows (tokens) and 628,905 columns (paragraphs), and although only one in every ten thousand terms is nonzero, the MATLAB sparse matrix representation is still 345 MB in size. The size of the matrix means that calculation of an exact Singular Value Decomposition is impractical. In addition, most popular methods to generate an approximate Singular Value Decomposition, including Lanczos [9], would require impracticably large amounts of memory for this data set.

[8]The Frobenius norm $||A||_F$ of matrix A is defined as: $||A||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} A_{ij}^2}$.

Drineas, Kannan & Mahoney [19] describe two algorithms that allow the computation of a low-rank approximation to a matrix without the need for large amounts of memory. Both algorithms return a description of the most important principal components, one being $O(m+n)$ in memory and time, the other being $O(1)$. To reconstitute the low-rank approximation to the original matrix requires further multiplications to be carried out, but these can also be streamlined to reduce memory requirements.

In the current application, we will only consider the LINEARTIMESVD algorithm, firstly because it is feasible to carry out, and secondly because it generates an estimate far closer to the exact solution than the CONSTANTTIMESVD algorithm.

The basic algorithm

The algorithm here uses weighted sampling without replacement, as described by Drineas, Kannan & Mahoney [19], but differs in that it returns right singular vectors instead of left singular vectors. This is important to simplify the remultiplication process required to reconstitute the low-rank approximation.

Input: $A \in \mathbb{R}^{m \times n}$, $r, k \in \mathbb{Z}^+$ s.t. $1 \leq k \leq r \leq m$, $p_i = \frac{|A_{(k)}|^2}{||A||_F^2}$.

Output: $H_k \in \mathbb{R}^{n \times k}$ and $\sigma_t(R), t = 1, \ldots, k$.

- For $t = 1$ to r,
  - Pick $i_t \in 1, \ldots, m$ with $\Pr(i_t = \alpha) = p_\alpha$, $\alpha = 1, \ldots, n$.
  - Set $R_{(t)} = A_{(i_t)} / \sqrt{r p_{i_t}}$.
- Compute RR^T and its singular value decomposition; say

$$RR^T = \sum_{t=1}^{r} \sigma_t^2(R) y^t {y^t}^T.$$

- Compute $h^t = R^T y^t / \sigma_t(R)$ for $t = 1, \ldots, k$.
- Return H_k, where $H_k^{(t)} = h^t$, and $\sigma_t(R)$, $t = 1, \ldots, k$.

The rank k approximation to A, which we will call $\tilde{A}_k$, is then calculated as follows:

$$\tilde{A}_k = A H_k H_k^T. \tag{2.10}$$

Practical problems with the implementation

Although the algorithm described is simple, there are a number of practical problems that arise due to the tradeoff between convenience and memory requirements. For convenience, we use MATLAB to carry out the required matrix operations, but this choice means that each matrix used in a calculation needs to be fully available in memory.

Of course, if one wrote code consistent with the Pass-Efficient model of data-streaming computation proposed by Drineas [18], one would avoid most of the memory limitations, but the loss of ease in this case was not justified. However, the current amount of data drove the utilised computer resources[9] to their limit, and a slightly larger input data set would have forced a move away from MATLAB.

The MATLAB code used can be found in Appendix B. On closer inspection, the reader will note a number of work-arounds, including:

- splitting matrices into submatrices and processing these submatrices separately,
- rewriting basic matrix operations like multiplication to operate with submatrices, and
- caching matrices to disk.

With the available resources, it was not possible to carry out the final multiplication in the reconstruction of $\tilde{A}_k$, since it is nonsparse in nature and would have taken up a couple of terabytes of disk space. However, it was not necessary to calculate the entire matrix. Instead, individual rows (representing tokens of interest) were calculated in the obvious way:

$$\tilde{A}_{k(i)} = A_{(i)} H_k H_k^T, \tag{2.11}$$

where the (i) subscript refers to the ith row. Performing a dot product on these rows allowed us to calculate the LSA measure for pairs of words in a space-efficient manner.

Monte Carlo estimation

To deal with the error in $\tilde{A}_k$ caused by the row sampling procedure, the LSA measure was estimated 30 times for every pair of words in each sentence of the

[9]Most of the time, the limiting resource was the 1.5 gigabytes of RAM available. However, disk space was also a consideration, since it was often necessary to store intermediate matrices to disk. With 20 gigabytes of disk space, the process ran without requiring the removal of intermediate files.

Potsdam Sentence Corpus. This number of simulations seems to be enough to allow a meaningful estimate of the LSA measure; this can be seen in Figure 2.14, where the standard error of the estimate is shown graphically.

2.2.6 Understanding the LSA measure

Traditionally, the semantic similarity measure used with Latent Semantic Analysis is the cosine of the angle of two vectors in the reduced high-dimensional space. Because of the high dimensionality of the space, the distribution of this measure is far from uniform. Most studies get around this nonuniformity by simply considering the measure to be ordinal: the higher the measure, the more similar the compared concepts must be. Another approach, proposed by Coccaro and Jurafsky [14], is to raise the cosine of the angle to an exponent in an attempt to "increase the dynamic range of the LSA probabilities".

The current problem requires a more detailed understanding of the meaning of a specific value of the measure, since we need to be able to answer a question like: "If the LSA measure shows that two words have a semantic similarity of 0.3, does that mean that they are related?". To answer such a question, we need to derive the distribution of the LSA measure for random vectors. We do this empirically, following the suggestion of Bellegarda [8]. Then, we use theoretical considerations to explain certain properties of the distribution.

The empirical approach to derive an angle probability distribution between two vectors in our semantic space is simple enough: one randomly selects vectors representing terms and calculates the angle between them. The more one samples, the closer the sampled distribution approximates the actual distribution. Figure 2.2 shows the empirically-derived distribution of semantic similarity between random vectors in the semantic space[10]. This function enables us to determine whether two vectors of interest are significantly closer than two randomly selected vectors, and thus infer whether the respective tokens are semantically related. For example, if the LSA measure of a pair of vectors is greater than 0.37, then these vectors are significantly closer together (with a p-value of 0.01) than would be expected at random. Even though we calculated the angle between 3.3×10^5 random pairs of vectors, we still cannot estimate the p-value reliably down to three decimal places, since very few of the sampled pairs are separated by less than $70°$; these are the cases that help to make up the long tail of the angle distribution.

In the following discussion, we will use the distribution of random vectors in a hypersphere to give us insight into the shape of the empirically derived distribu-

[10]The sampling algorithm used here was stratified due to disk space limitations: Ten randomly chosen token vectors were constructed at a time, and dot products were calculated between each pair of vectors, resulting in 45 values; this procedure was carried out many times, and all results were aggregated before being graphed.

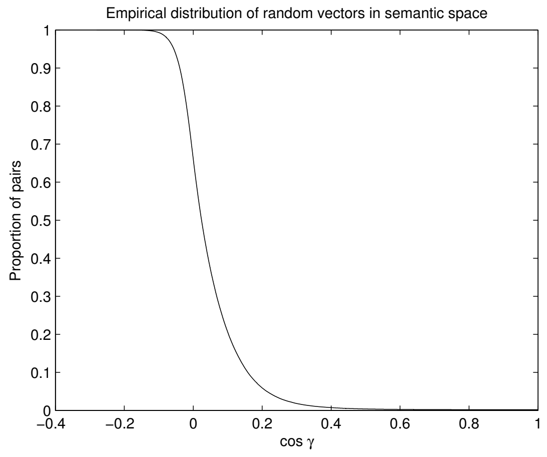

Figure 2.2: Empirically derived distribution of angular distance between two randomly sampled vectors in our semantic space.

tion. We follow the theoretical approach to see how the shape of the distribution reflects the method used to generate the semantic space. The theory will then allow us to predict the effect of altering parts of the method, such as the weighting schemes or the dimension of the semantic space.

Hyperspherical definitions

A hypersphere is the generalisation of the concept of a sphere to arbitrary dimension. To avoid ambiguity, we will use the definition from geometry that an n-hypersphere is defined as the set of points $(x_1, x_2, \ldots, x_n)$ such that

$$x_1^2 + x_2^2 + \ldots + x_n^2 = R^2, \tag{2.12}$$

where R is the radius of the hypersphere. In an extension of spherical coordinates to higher dimensions, we can also describe the n-hypersphere with one radial distance variable and $n-1$ angles in the following way:

- radial distance = R
- one angle resembling longitude, θ, ranging from 0 to 2π
- $n-2$ angles resembling colatitude, ϕ_i, ranging from 0 to π

These specifications in Cartesian and hyperspherical coordinates are related in the following way:

$$\begin{aligned}
x_1 &= R\sin\phi_1\sin\phi_2\ldots\sin\phi_{n-2}\sin\theta \\
x_2 &= R\sin\phi_1\sin\phi_2\ldots\sin\phi_{n-2}\cos\theta \\
x_3 &= R\sin\phi_1\sin\phi_2\ldots\cos\phi_{n-2} \\
&\vdots \\
x_{n-1} &= R\sin\phi_1\cos\phi_2 \\
x_n &= R\cos\phi_1
\end{aligned}$$

For the following discussion, we will arbitrarily consider the unit n-hypersphere, where $R = 1$. In addition, we will arbitrarily choose a 'target vector' to be the high-dimensional generalisation of north, which will be defined to be in the direction of the x_n axis; equivalently, this is the vector that has $\phi_1 = 0$.

Distribution of random vectors in isotropic space

Let us first consider the case of isotropic space, where random vectors are equally likely to be oriented in any direction. Imagine that we pick random position vectors describing points on the surface of the unit n-hypersphere. The proportion of these vectors that are less than an angular distance γ away from our target vector (i.e., those vectors where $\phi_1 \leq \gamma$) is equivalent to the proportion of the hypersurface area of the hypersphere subtended by the angle γ (see Figure 2.3 for an illustration of this in three dimensions). The hypersurface area S_n of the unit n-hypersphere is

$$\begin{aligned}
S_n &= \int_0^\pi d\phi_1 \int_0^\pi \sin\phi_1 d\phi_2 \int_0^\pi \ldots \int_0^\pi \prod_{i=1}^{n-3} \sin\phi_i d\phi_{n-2} \int_0^{2\pi} \prod_{i=1}^{n-2} \sin\phi_i d\theta \\
&= \int_0^\pi \int_0^\pi \ldots \int_0^\pi \int_0^{2\pi} \sin^{n-2}\phi_1 \sin^{n-3}\phi_2 \ldots \sin\phi_{n-2} d\theta d\phi_{n-2} \ldots d\phi_2 d\phi_1,
\end{aligned}$$

and the hypersurface area S_n^γ of the hypersector $\phi_1 \leq \gamma$ is

$$S_n^\gamma = \int_0^\gamma \int_0^\pi \ldots \int_0^\pi \int_0^{2\pi} \sin^{n-2}\phi_1 \sin^{n-3}\phi_2 \ldots \sin\phi_{n-2} d\theta d\phi_{n-2} \ldots d\phi_2 d\phi_1.$$

The proportion can thus be calculated in the following way:

$$\frac{S_n^\gamma}{S_n} = \frac{\int_0^\gamma \sin^{n-2}\phi_1 d\phi_1}{\int_0^\pi \sin^{n-2}\phi_1 d\phi_1}, \tag{2.13}$$

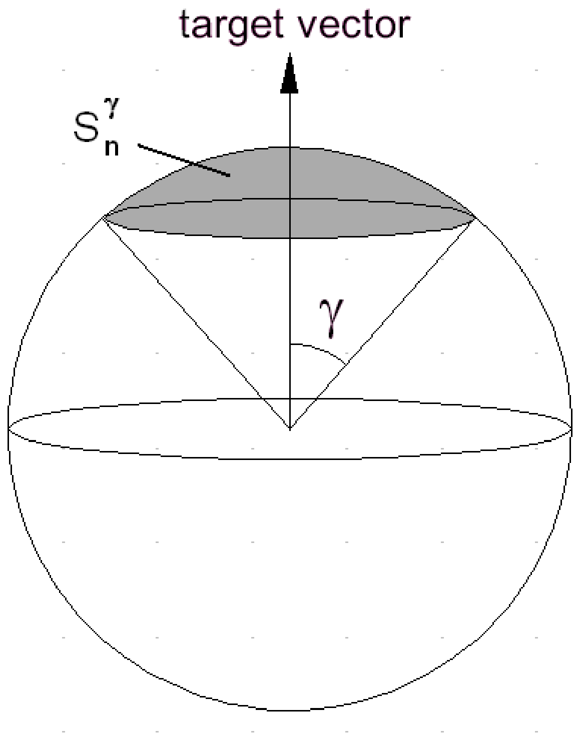

Figure 2.3: Illustration of the hypersurface area S_n^γ.

since all the other terms in the integral are identical in the numerator and denominator. Figure 2.4 shows this relation for $n = 200$. From the figure, we can see that two randomly chosen vectors in a high-dimensional space are very likely to be almost perpendicular.

We can consider the vertical axis to be a p-value of a statistical test with the alternative hypothesis: “Two vectors are significantly closer to each other than can be explained by a uniform random distribution.” If we use this interpretation, we can see that all commonly used significance levels give rather similar critical angular thresholds. Once we are at angular distances less than this critical angle, our p-value criterion quickly loses its ability to distinguish how close together vectors are; for example, if we have one pair of tokens separated by an angular distance of 45° and a second pair separated by 50°, is the first pair significantly more semantically related than the second? In other words, the metric of angular distance in high-dimensional spaces does not allow a meaningful discrimination of semantic closeness, although it is good at detecting whether two tokens are semantically related. This problem is well-known as the “curse of dimensionality” in the context of clustering in high-dimensional spaces, and is an inherent problem of metrics that do not artificially magnify the importance of the region near

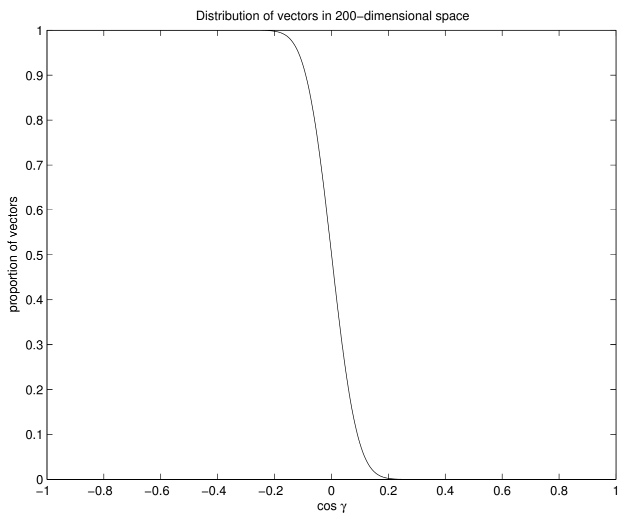

Figure 2.4: Proportion of randomly distributed vectors within angle γ from an arbitrarily chosen target vector in 200-dimensional isotropic space.

the axes; however, if one does use such a metric, like the fractional distance metric[11] suggested by Aggarwal [5], the importance of function words is likely to be artificially inflated, counteracting the effect of the log odds weighting.

Anisotropy generated by the nonnegativity of the word frequency matrix

The semantic space created by our method is embedded in a hyperspace that approximates the positive hyperquadrant. This is a consequence of the fact that Latent Semantic Analysis clusters vectors arising from a nonnegative word frequency matrix. Even after the binary and log odds weightings are applied, only 1.9% of the nonzero entries are negative, and these negative entries are small in magnitude. The biases in direction and length of vectors in our semantic space are displayed in Figure 2.5, where vectors are parametrised by their angular distance from the main diagonal vector $(1,1,\dots,1)$. Since the dimension reduction step of the procedure generally brings term vectors closer together, almost all vectors will be wholly in the positive hyperquadrant, while most others will lie close to the hyperfaces of the hyperquadrant. The trend toward higher norms as vectors approach the main diagonal shows that the binary and log odds weightings have not completely succeeded in removing frequency biases from the data.

This anisotropy is reflected in the empirical distribution (Figure 2.2) as an asymmetric shift away from large angular distances when compared to the iso-

[11]This fractional norm is a simple extension of the standard norm on an L^p space. It is defined as $||x||_f = \left(\sum_{i=1}^{d} |x_i|^f\right)^{1/f}$, where d is the dimension of vector x and $f \in (0,1)$.

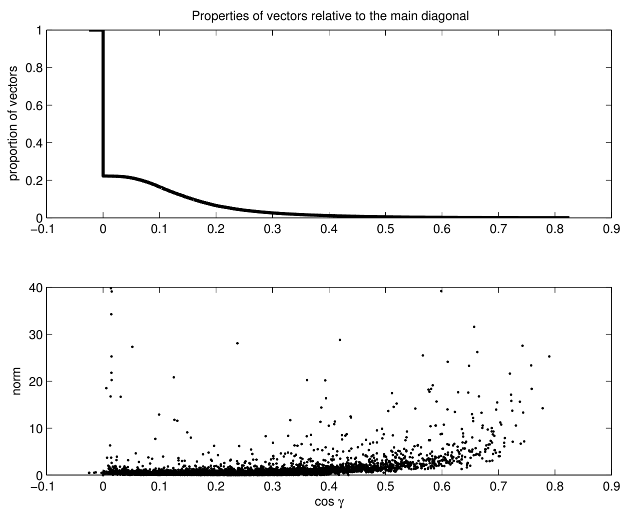

Figure 2.5: Properties of vectors in our semantic space when classified based on their angular distance from the main diagonal. The Euclidean norm axis has been rescaled to show the features of the main bulk of points; as a result, a number of points with extremely high norms have been excluded from the graph. Note that the angle between an axis and the main diagonal in our 600,000-dimensional embedding space is an unintuitively large 89.9°.

tropic case: There are far fewer angular distances greater than 96° represented (i.e. $\cos\theta < -0.1$), while the point of inflection of the empirical distribution still remains at about 90° (i.e. $\cos\theta = 0$).

Effect of anisotropy caused by the singular value distribution

There is another anisotropy introduced by the fact that vectors in our semantic space are likely to be longer in some directions than others; this fact can be deduced from the singular value distribution. Figure 2.6 shows the singular value distribution of the matrix R created in one run of the Fast Monte Carlo Singular Value Decomposition algorithm; we can see that the combination of binary weighting, log odds weighting and row weighting have produced a spectrum that is almost flat, except for the largest singular value. To examine the effect of this type of anisotropy, we can look at a semantic space that is almost isotropic, except that it is stretched in the principal direction corresponding to the largest singular value.

Take the case where the semantic space is isotropic in all directions except one, and this principal direction is at right angles to the target vector. Then, the

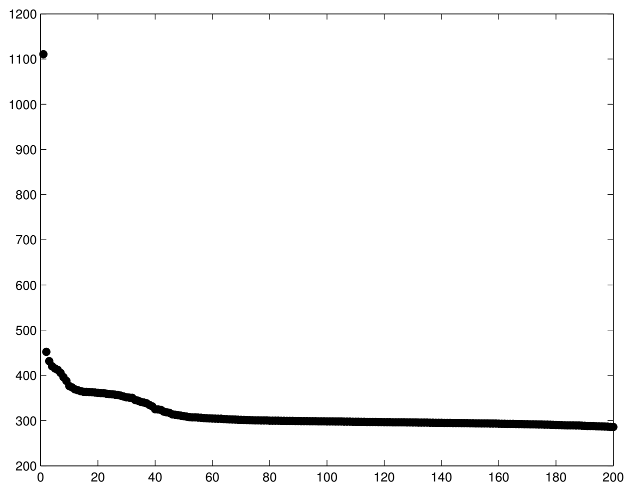

Figure 2.6: Singular value spectrum of the weighted sampled matrix *R* created during one run of the Fast Monte Carlo Singular Value Decomposition algorithm.

hypersurface area S_n^γ of the hypersector $\phi_1 \leq \gamma$ becomes

$$S_n^\gamma = \int_0^{\gamma_a} \int_0^\pi \dots \int_0^\pi \int_0^{2\pi} \sin^{n-2}\phi_1 \sin^{n-3}\phi_2 \dots \sin\phi_{n-2} d\theta d\phi_{n-2} \dots d\phi_2 d\phi_1,$$

where $\gamma_a = \arctan(a \tan\gamma)$ and $a > 1$ is an anisotropy factor. The proportion of random vectors within angle γ from the target vector becomes

$$\frac{S_n^\gamma}{S_n} = \frac{\int_0^{\gamma_a} \sin^{n-2}\phi_1 d\phi_1}{\int_0^\pi \sin^{n-2}\phi_1 d\phi_1}. \tag{2.14}$$

We can see that the stretching serves to rescale the distribution of angular distances, as illustrated in Figure 2.7. This effect accounts well for the stretching of the empirical distribution for angles less than 90° when compared to the isotropic distribution.

Conclusions about the shape of the semantic similarity distribution

To account for the shape of the empirical semantic similarity distribution, we needed to consider three main factors: the dimension, the bias toward positivity caused by starting from a nonnegative frequency matrix, and the stretching of space seen in the singular value spectrum.

Increasing the dimension of the space increases the probability that random vectors will be close to perpendicular, reflected by a steeper central section in the semantic similarity distribution. This certainly does not improve the discrimination of our measure; the only reason that we might consider increasing the

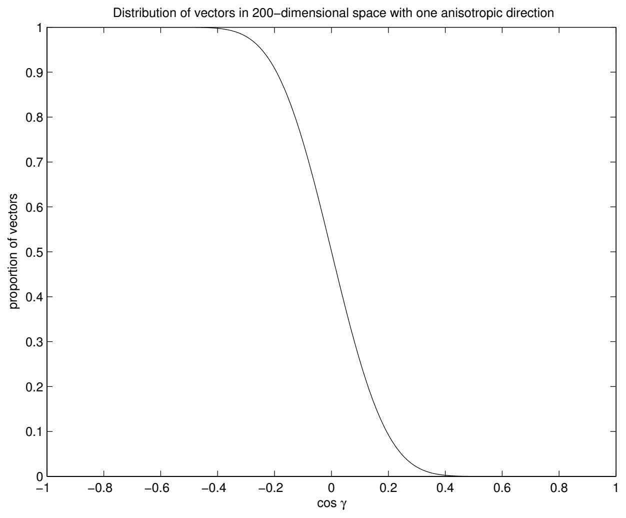

Figure 2.7: Proportion of randomly distributed vectors within angle γ from an arbitrarily chosen target vector in 200-dimensional space, with an anisotropy factor of 2.2, perpendicular to the target vector.

dimension of our semantic space is if we believed that the dimension reduction step causes too great a loss of important information.

The bias toward positivity is unlikely to be affected much by the specific combination of local and global weightings, since we start out with a nonnegative frequency matrix, although some weightings will result in a stricter restriction to the positive hyperquadrant than others. As a result, we would expect that the LSA metric should be either positive or very close to zero.

The stretching of space as seen in the singular value spectrum is specific to the method being used to create the semantic space. The original frequency distribution obeys a power scaling relation reflecting the manifestation of Zipf's law in natural language [41], but the combination of binary local weighting, log odds global weighting and dimension reduction removes much of the heterogeneity in the truncated singular value spectrum. Even small changes to the method may result in significantly different amounts of stretching of the semantic similarity distribution. The more heterogeneous the singular value spectrum, the harder it should be to predict the amount of stretching of the similarity distribution, since the stretching will affect different vectors by different amounts.

Because none of these factors changes the general shape of the semantic similarity distribution, we can see that a semantic similarity measure based on angular distance will always suffer from the curse of dimensionality mentioned in Section 2.2.6, meaning that the measure allows the detection of semantic relatedness, but is poor in discriminating the amount of relatedness. This result may not be so counterintuitive: Miller [45] observes that the human "span of absolute judgement" for a unidimensional variable is quite limited[12]; if we apply this to the unidimensional LSA measure between pairs of words, we may infer that humans are able to classify a word pair as semantically related or not, but have a limited ability to distinguish the degree of relatedness. Since the current work is aimed at replicating human behaviour, it makes no sense for us to artificially increase the dynamic range of the measure—such an increase would only mean that we would have to redefine the critical p-values defining significance in a nonintuitive manner.

2.3 Do the different methods give rise to different semantic measures?

Conceptually, there are many obvious differences between the semantic measures we have calculated; some of these differences are listed in Table 2.1. However, our semantic measures all aim to represent the same attribute, namely the semantic relatedness of two words, and therefore, we should expect that the measures to be somewhat correlated. In Figures 2.8 and 2.9, we can see that the LSA measure is weakly correlated with the web conditional co-occurrence measure, but seems to be unrelated to the web pointwise mutual information measure.

Web measures	**LSA measure**
First order	Second order
May be dependent on word order	Independent of word order
Easy to calculate	Difficult to calculate
Large amount of training text	Small amount of training text
Noisy training text	Clean training text
Heterogeneous subject material	Newspaper articles

Table 2.1: Differences between the semantic measures.

[12]Miller's actual words are "... I maintain that for unidimensional judgments this span is usually somewhere in the neighborhood of seven".

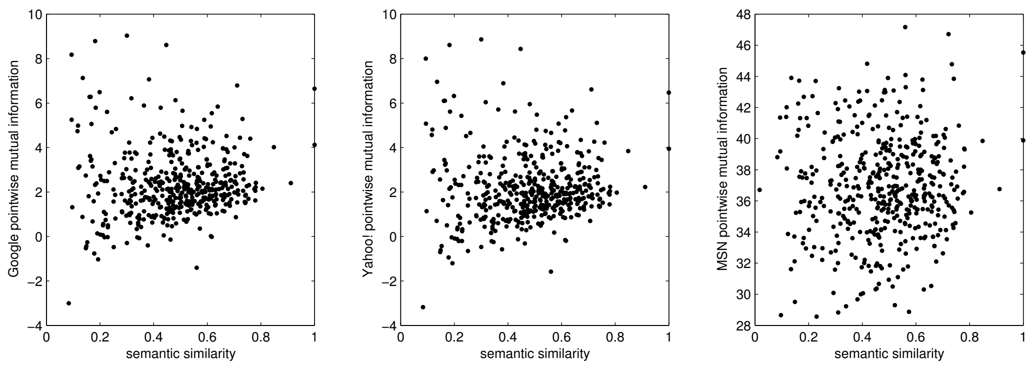

Figure 2.8: Relationship of semantic similarity to pointwise mutual information. These two measures appear to be uncorrelated.

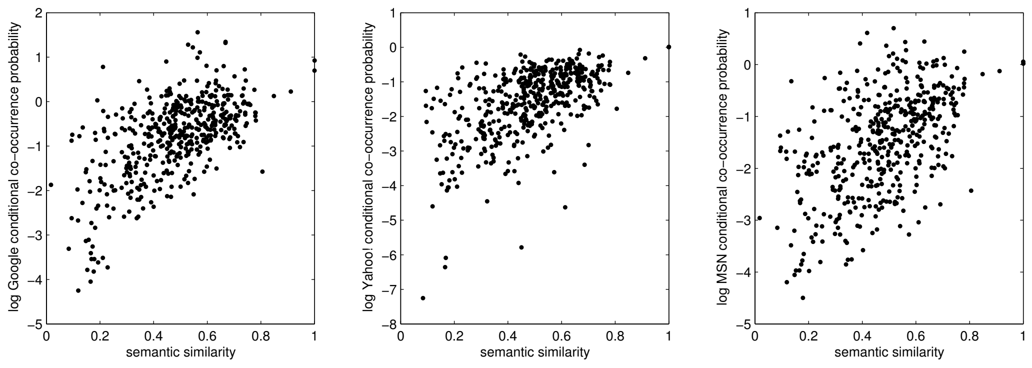

Figure 2.9: Relationship of semantic similarity to log conditional co-occurrence probability. These two measures are weakly correlated.

2.4 Comparison of our semantic measures with predictability

The semantic measures that we have defined give us a way of determining the semantic relatedness of two individual tokens. However, predictability reflects the effect of a multiword context upon a specific target word. It is not obvious how one should generalise similarity measures defined for word pairs to take into account a multiword context.

An interesting feature of Latent Semantic Analysis is the proposition that one may be able to assign vectors in semantic space not only to tokens, but to combinations of tokens, such as word sequences or sentences. Landauer and Dumais [36] propose that the vector for each combination be simply "an appropriately weighted vector average of the condensed vectors of all the events whose local temporal associations constituted it", but it is not clear whether this is a meaningful or sensible thing to do. A more plausible method to derive a vector for a combination of tokens, proposed by Bellegarda [8], is to create a new 'document' containing the required tokens and then use the already-discovered principal components to project this document to the final reduced-dimensional semantic space, as if it were a real document. However, this approach is computationally difficult and is only seems to be worth carrying out if the preceding context is relatively long.

Because contexts in the Potsdam Sentence Corpus never consist of more than ten words, it generally seems reasonable to treat content words[13] found in the context as being almost semantically independent: this means that every content word significantly increases the amount of meaning in the context and that there is little redundancy. As a first approximation, we will also assume that most of the semantic content of the sentence is contained within the content words, although it is clear that function words also add to semantic content by specifying case (especially in the case of a non-default word order), negation, pragmatic nuances or by being part of an idiom. With this simplifying assumption, a context can be meaningfully broken down into its substituent content words. Then, we aggregate the pairwise semantic relatedness values between a particular target content word and all content words occurring in the preceding context, perhaps taking the maximum or a weighted sum, to give the semantic relatedness between the target word and its context. In the following analysis, we have chosen to take the maximum over the pairwise semantic relatedness values to represent the relatedness of the multiword context to the target word.

2.4.1 The effect of function words in the context

Theoretically, the semantic relatedness between a content and a function word should be almost nonexistent, so one should be able to take a maximum over all previous words in the context, not just previous content words. However, this assertion does not necessarily hold for our semantic measures. We consider each case separately in the following discussion.

[13]In linguistics, the terms 'content word' and 'function word' are used in an attempt to distinguish words that refer to a concept from words that serve a purely syntactic function. Clinical observations of agrammatism [26] and semantic dementia [65] support the hypothesis that content words are processed and stored separately from function words. However, the distinction between the two categories is not sharp at the token level; indeed, certain tokens serve as content words in some contexts and function words in other contexts. This duality means that one cannot discriminate between content and function words solely on the basis of syntactic category.

Web pointwise mutual information measure

Let us consider a pair of words where one of the words is a content word and the other is a function word. We expect that the two words should distribute independently, since the probability that a function word occurs in a document should be roughly independent of the type of document. Mathematically, we can rewrite the pointwise mutual information measure to reflect this:

$$\mathrm{PMI}_{ij} = \log_2\left(\frac{p(w_i \cap w_j)}{p(w_i)p(w_j)}\right) = \log_2\left(\frac{p(w_i|w_j)}{p(w_i)}\right) = \log_2\left(\frac{p(w_i)}{p(w_i)}\right) = 0, \quad (2.15)$$

where w_i is a function word and w_j is a content word. Thus, the pairwise mutual information should be approximately zero. This seems to holds well in practice[14], meaning that we do not need to actively remove function words from consideration when considering a multiword context.

Web conditional co-occurrence measure

We cannot assert that the presence of a content word will be independent of a preceding function word, since the exact form of the content word may be strongly bound by morphosyntactic constraints. However, the probability of finding a specific function word in a document is likely to be many orders of magnitude larger than that of the content word, meaning that the resulting conditional probability will be of the order of magnitude of the probability of finding the content word, which is generally small. Another way of stating this rationale is to say that content words are more independent of preceding function words than they are of related preceding content words. The mathematical representation of this argument is as follows:

$$p(w_j|w_i) = \frac{p(w_i \cap w_j)}{p(w_i)} \sim p(w_j) \text{ since } p(w_j) \ll p(w_i). \quad (2.16)$$

If it is true that the target content word occurs far more often when preceded by a related content word, then this relation will mask the presence of any function word in the context, since then $p(w_j|w_i) \gg p(w_j)$. However, if there is no related content word in the context, then it is possible that function words will provide a significant amount of noise for this measure. Equivalently, to be able to ignore the presence of function words, the maximum pointwise mutual information for each target content word would need to be well above zero.

[14]For estimates derived from MSN, the ‘zero’ seems to be shifted upwards because of the inherent underestimation of frequencies when only a single search term is specified, as mentioned in Section 2.1.3.

Since in this case, function words may prove to be a problem, it would be nice to have an automatic way of distinguishing function words from content words. With web-based estimates, the only straightforward way to attempt this is to filter by raw occurrence frequency, which is obviously unsatisfactory. Another way of dealing with function words is to filter them out using a stop-list, parser or hand tagging, but these approaches detract from the automatised nature of the method.

We will include function words in our analysis when we use conditional probability as our measure, cognisant of the possibility that any results in the lower probability ranges may well be overestimated.

LSA measure

The log odds weighting was specifically chosen to reduce the importance of tokens that occur in many documents, while emphasising those tokens that occur in only a few documents. We obtain a simple statistical criterion to distinguish words with high semantic content from those with low semantic content by applying the ∞-norm to each word vector. This criterion does not allow one to directly map words into the categories of function and content words, but we would expect that function words have a low ∞-norm in comparison to content words. In Figure 2.10, we see that points closer to the main diagonal, which represent vectors that inherently have more nonzero elements, generally have a lower ∞-norm. This property is not preserved by the dimension-reduction step of Singular Value Decomposition, as can be seen in Figure 2.11. Thus, we use the ∞-norm obtained before the application of the approximate Singular Value Decomposition technique as our measure of the semantic content of each token. We will refer to this norm as the pre-SVD ∞-norm.

2.4.2 Graphical comparison and interpretation

Web pointwise mutual information measure

Our pointwise mutual information measure is plotted against predictability in Figure 2.12; in the plots, there seems to be no relationship between these two parameters. This finding is a little surprising, since we know that one can use the pointwise mutual information measure to successfully find synonyms [64]. This implies that a measure of synonymity, like that embodied in WordNet [24], is unlikely to be useful in the generation of predictability values.

Web conditional co-occurrence measure

In Figure 2.13, we see the conditional probability that two words co-occur in a web page plotted against predictability. The main feature visible in the plots is

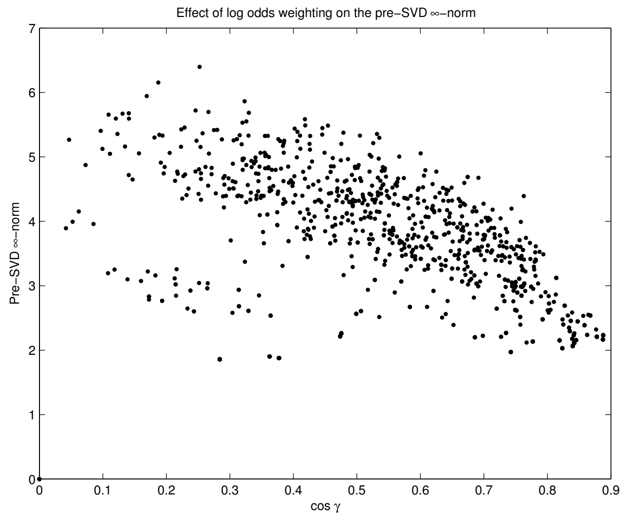

Figure 2.10: Effect of log odds weighting on the ∞-norm of word vectors. Vectors are parametrised on the horizontal axis by the cosine of their angle from the main diagonal. The closer a vector is to the main diagonal, the more nonzero elements it must have. We expect function words to have many nonzero elements, since they occur in many documents. We can see that, in general, the log odds weighting reduces the ∞-norm for words that occur in many documents relative those that only occur in a small number of documents.

the spread of points at zero predictability: If the conditional probability is small enough, we can be sure that the corresponding predictability value will be very close to zero. Another way of stating this is as follows: Words that are totally unrelated to their context are very likely to be unpredictable. The possible overestimation of the conditional co-occurrence probability mentioned in Section 2.4.1 does not change this relation at all—in fact, lower co-occurrence probabilities would strengthen the effect.

The relation between co-occurrence conditional probability and predictability is obviously not one-to-one, implying that a high semantic relatedness of this sort is not sufficient to guarantee a high predictability. Actually, we would not expect any simple relationship, since we believe that predictability reflects far more than just semantic similarity.

LSA measure

A simple plot of the LSA measure against predictability for ‘content’ words can be found in Figure 2.14. Bearing in mind that the LSA measure needs to be above 0.37 to be significant at the 1% confidence level, we can see that there are significantly fewer points in the lower right quadrant of the plot than elsewhere.

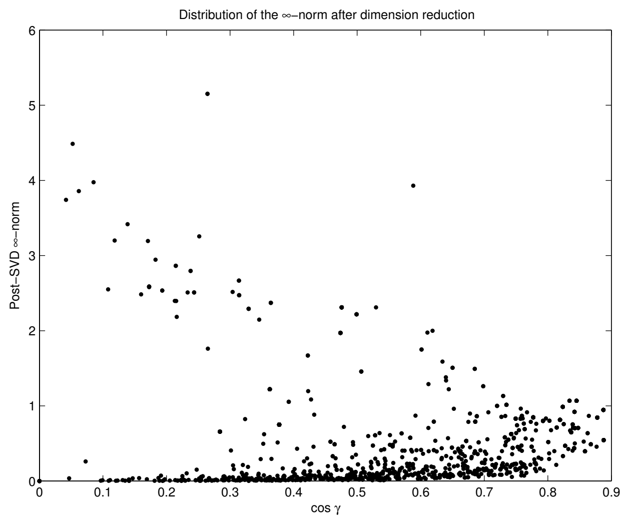

Figure 2.11: Distribution of the ∞-norm of word vectors after application of approximate Singular Value Decomposition. Vectors are parametrised on the horizontal axis by the cosine of their angle from the main diagonal. It is now no longer true that there is a low ∞-norm for words that are near to the main diagonal.

This means that a low value of the LSA measure is almost sufficient to guarantee a low predictability.

In the plot, we see a number of points for which the LSA measure is zero: these represent the first content word of each sentence; under our simplistic assumption, such words have no previous semantic context. The LSA measure is totally uninformative in this case, and the predictability of these points must be modelled in another way.

It is illuminating to study the cases that occur in the lower right quadrant (excepting those on the predictability axis); these represent content words that are predictable but seem to be semantically unrelated to their context. There are two reasons why there are cases where the predictability is larger than 0.1 and the LSA measure smaller than 0.30:

1. The ZEIT corpus did not have the required semantic relationship because it is too small, or
2. The content word occurs as part of an idiom.

As examples, the first reason seems to responsible for the low LSA value for the word `Matratze` in the following sentence:

```
Die Kinder hüpften auf der alten Matratze herum.
```

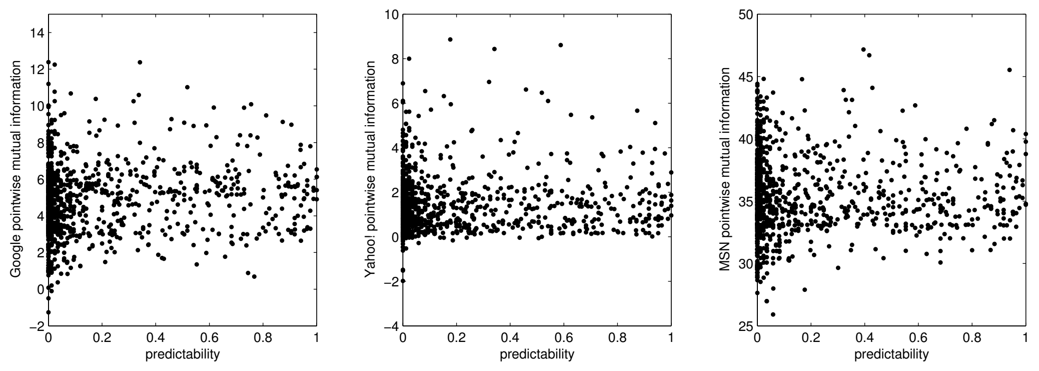

Figure 2.12: Relationship of predictability to pointwise mutual information. The vertical axes have been rescaled so that the resulting plots look similar. Pointwise mutual information appears to be unrelated to predictability.

while the second reason accounts for the high predictability of the word `Lot` in the following sentence:

```
Nach dem Streit schien alles wieder im Lot zu sein.
```

Cases of the first type can be addressed by using a larger training corpus, while cases of the second type are better dealt with by considering word *n*-gram probability.

2.5 Chapter summary

We generate two web co-occurrence measures (pointwise mutual information and conditional co-occurrence probability) and a measure derived from Latent Semantic Analysis. Web-based measures are easy to collect, but are inherently noisy; this noise is partly due to the nature of data on the Internet, but also partly due to anomalies introduced by Internet search engines. The construction of the LSA measure is a far more involved process, and although the source data is cleaner than Internet data, noise is introduced in the use of a Monte Carlo estimation algorithm; it is necessary to use such an an algorithm to overcome computational limitations. A large amount of work was necessary to clean the source corpus and convert it into an appropriate numerical format.

The interpretation of the LSA measure is not a simple task, since token vectors exist in a high-dimensional space. The distribution of angles between vectors in a high-dimensional space is inherently nonuniform. It is possible but not easy to estimate this distribution empirically; alternatively, it may be possible to use

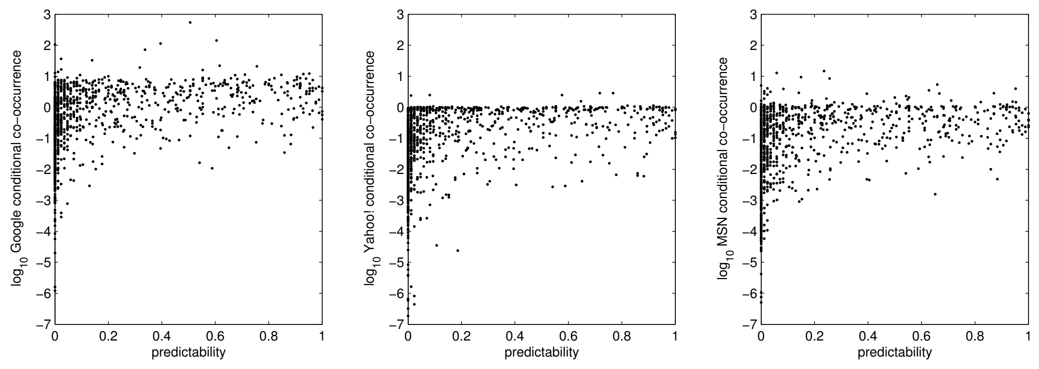

Figure 2.13: Relationship of predictability to web conditional co-occurrence. Points with a low conditional co-occurrence probability are very likely to have a low predictability value. Note the lack of strict adherence to nonpositive values, especially in the case of results from Google.

theoretical considerations to guess the form of the distribution, which may then be used to interpolate the empirical results. Once an angle distribution has been estimated, we are able to transform the LSA measure into a measure that can be interpreted as a probability.

After generalising our pairwise measures to allow us to consider a multiword context, we graphically compare our measures to predictability. The web pointwise mutual information measure seems to be unrelated to predictability. However, the web conditional co-occurrence measure and the LSA measure show the same distinctive pattern: low values of these measures indicate words with low predictability.

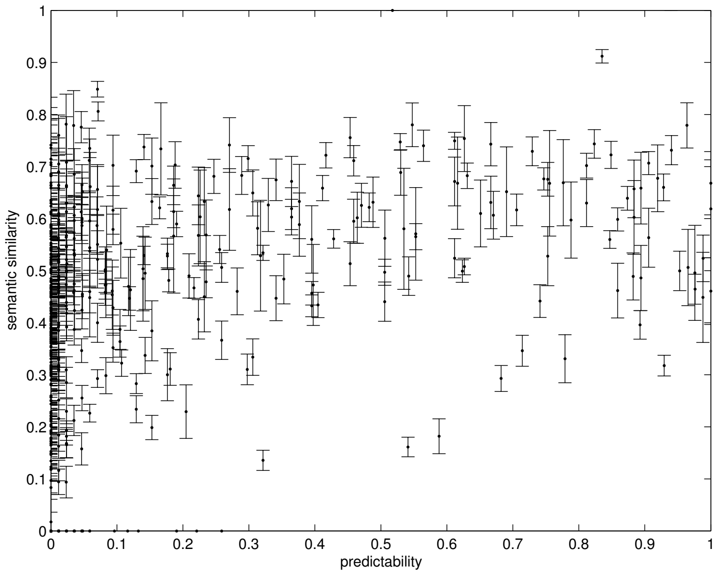

Figure 2.14: Relationship of predictability to the LSA measure. Only words with a pre-SVD ∞-norm greater than 3.2 were included in the analysis leading to this graph. The length of the error bars is twice the standard error. Points with a low conditional co-occurrence probability are very likely to have a semantic similarity that is not significantly different from chance. The points with zero semantic similarity correspond to the first content word of each sentence.

Chapter 3

Word *n*-gram measures

In this chapter, we generate simple word *n*-gram measures and compare them to predictability. These measures are simple to collect and do not suffer from issues of sparsity, which is normally a problem for such measures.

3.1 Using a word *n*-gram model to capture short-range structure

Simple computational implementations of statistical language models date back to the 1970s; one early application of such models was to the area of speech recognition [30]. Rosenfeld [55] gives a review of the major attempts to improve the performance of statistical language models.

At the word level, the simplest of these models are word *n*-gram models, where one attempts to model the probability of observing an *n*-word sequence in the language. For example, a word 2-gram (or bigram) model attempts to specify the probability of coming across a specific two-word sequence in the language, while a word 3-gram (or trigram) model would deal with three-word sequences. More complicated models may consider word bigram and word trigram probabilities simultaneously, as well as looking for other local statistical patterns.

Goodman [27] combined various techniques commonly used in the creation of statistical language models to look for synergies and conflicts between the different techniques; he focussed especially on the tradeoff between the complexity of the model and the quality of the model fit. One important conclusion is that English word *n*-gram models show an optimal fit for a rather small value of *n*, between three and five. This result does not directly carry across to German, but it suggests that it is sensible to use word *n*-gram models with a relatively small value of *n*.

We limit ourselves here to using the raw word bi- and trigram probabilities derived from the World Wide Web as our measures of short-range relatedness. We backoff zero bigram probabilities naïvely to the unigram probability, and backoff trigram probabilities in a similar way. In the following discussion, we justify the validity of such a simple measure.

3.1.1 Training text effects

The traditional method of training statistical language models is to use a standard (and usually clean) training text corpus, derive all required statistics based on it, and then smooth the raw probabilities to account for unseen word patterns.

The resulting statistical language model suffers from a number of intrinsic biases, including those of content and style. These biases can often render the derived model useless for the required application, sometimes quite obviously, and sometimes rather subtly. For instance, it is obvious that if one wanted to build a language model of modern language, it would be nonsense to train the model on the writings of Shakespeare. However, it can also be just as nonsensical, though not nearly as obvious, to train a language model of spoken language on only written texts. Some researchers attempt to circumvent problems of bias by using a 'balanced' corpus, where texts are drawn from many different sources.

Banko and Brill [7] studied the effect of the size and content of the training text on the performance of the resulting language model. They concluded that an increase in size of the training text improves the performance of the language model. However, Rosenfeld et al [56] highlight the need for in-domain training data: the addition of large amounts of out-of-domain training data to a small in-domain training set improves model performance by a disproportionately small percentage.

3.1.2 The problem of sparse data

The great majority of words occur very infrequently, with the frequency distribution approximately following Zipf's law [69, 57, 41]. This means that most sequences of words that one wants to analyse with a language model will never have occurred in the training text. This problem is known as the "sparse data problem", and is a pure consequence of the combinatorial explosion of possibilities with which words can be combined. This problem is particularly marked when one deals with word *n*-gram models with high values of *n*.

One standard way to deal with the sparsity of training data is to use a smoothing algorithm to guess the probability of unseen word patterns, based on the statistical properties of those patterns actually seen. Goodman [27] mentions a number of common smoothing techniques and shows that Interpolated Kneser-Ney smoothing performs well; in addition, he provides a theoretical argument to prefer this particular smoothing technique over others.

Another way to deal with sparse training data is to increase the size of the training corpus to increase the chance that the word pattern will have been present during training. Keller [31] found that a substantial increase in the size of the training text seems to override problems of noisy data, even data as noisy as that found on the Internet.

In practice, most clean corpora (and in particular, the ZEIT corpus) are far too small to give a comprehensive set of estimates of word trigram frequencies, even when smoothed, although their coverage of unigram and bigram word frequencies may be reasonably good. This implies that one has no choice but to rely, at least in part, on Internet-based frequency estimates. We rely solely on the size of the Internet to deal with the sparsity issue. Note that the problems mentioned previously in Section 2.1.3, which were to do with deriving Internet-based frequency estimates from search engines, also need to be taken into consideration here.

3.1.3 Cross-validation of web frequency estimates

To check the validity of web frequency estimates, we compared them to frequency norms derived from the DWDS Kerncorpus [3], which is a balanced corpus made up of literature, journalistic prose, subject-specific prose, technical documentation and transcribed speech. Figures 3.1, 3.2 and 3.3 show the comparisons of word unigram, bigram and trigram frequencies respectively. In general, the web estimates agree with the DWDS norms for high frequencies, where we would expect that the DWDS corpus does not suffer from problems of sparsity. However, the estimates obtained from the Google and Yahoo! APIs have an artificial gap in the frequency band from 10^3 to 10^4, as noted in Section 2.1.3.

3.2 Comparison of word *n*-gram probabilities to predictability

Figure 3.4 shows plots of word bi- and trigram probability against predictability. It is evident that word *n*-grams that are very uncommon are also very likely to be unpredictable. Word trigram probability seems to be a slightly better predictor than word bigram probability, since the word trigram probability plots show a more pronounced positive trend than the word bigram probability plots. However, it is clear that word *n*-gram probability only captures a small part of predictability.

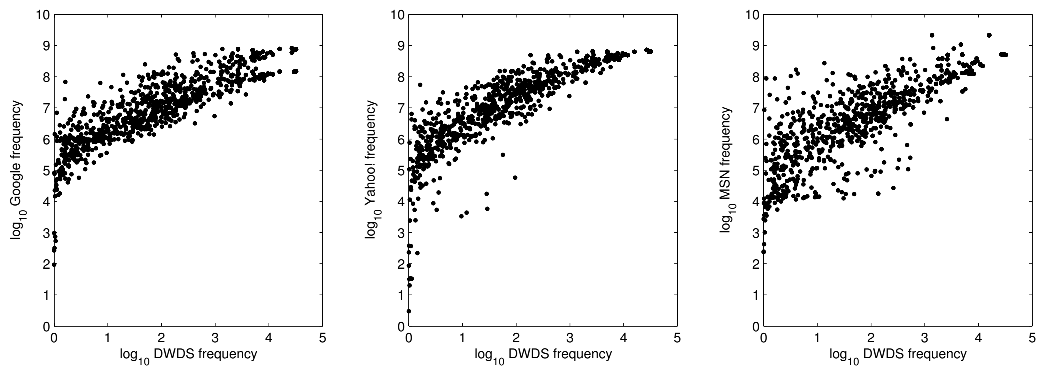

Figure 3.1: Comparison of web word frequency estimates to DWDS word frequency norms. The web norms are roughly in agreement with those derived from the DWDS corpus, with the Google norm giving the best agreement.

Looking more carefully at the cases in the lower right quadrant, which represent highly predictability words that do not occur often after the previous one or two words, we note that the target word is highly semantically related to its context. This is clearly visible in the following sentence, where `Kraftstoff` is highly predictable, but has a low probability of occurrence given the one or two preceding words as context:

`Sogar aus Raps läßt sich Kraftstoff herstellen.`

The relationship here of `Kraftstoff` to `Raps` is better detected by using a semantic similarity measure.

3.3 Chapter summary

Simple word *n*-gram probabilities derived from the Internet serve as our measures of short-range relatedness. The sheer size of the Internet allows us to substantially reduce the problem of sparsity, meaning that there is less need to smooth the raw probabilities. If a two- or three-length sequence of words occurs very improbably, then the last word of the sequence very probably has a low value of predictability.

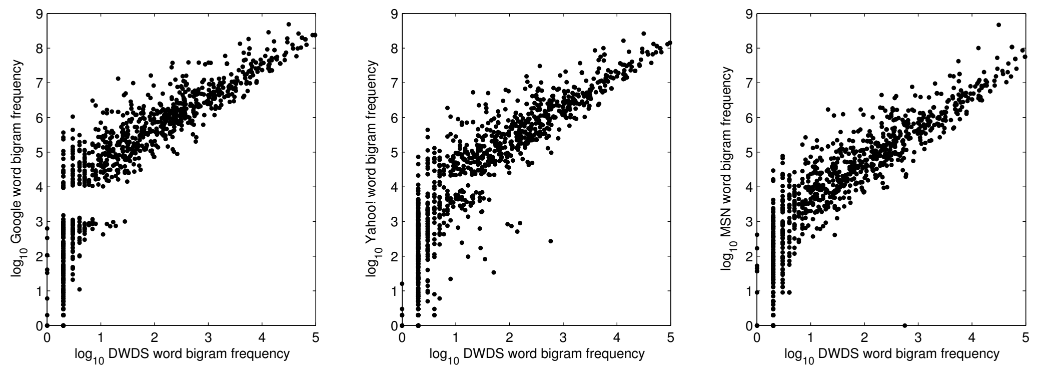

Figure 3.2: Comparison of web word bigram frequency estimates to DWDS word bigram frequency norms. For high word bigram frequency estimates, the web norms are roughly in agreement with those derived from the DWDS corpus. The sparsity of the DWDS corpus in comparison with the web norms is clearly visible. Note the anomalous lack of points with frequency between 10^3 and 10^4 in the Google and Yahoo! estimates.

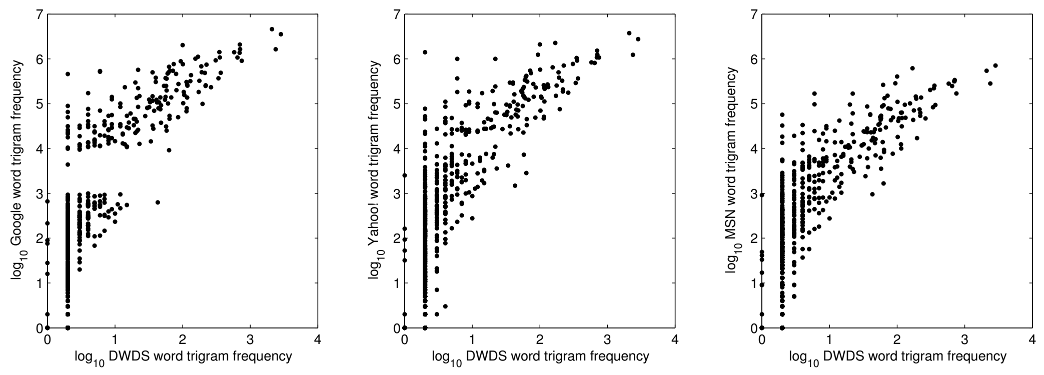

Figure 3.3: Comparison of web word trigram frequency estimates to DWDS word trigram frequency norms. For high word trigram frequency estimates, the web norms are roughly in agreement with those derived from the DWDS corpus. The sparsity of the DWDS corpus in comparison with the web norms is clearly visible. Note the anomalous lack of points with frequency between 10^3 and 10^4 in the Google and Yahoo! estimates.

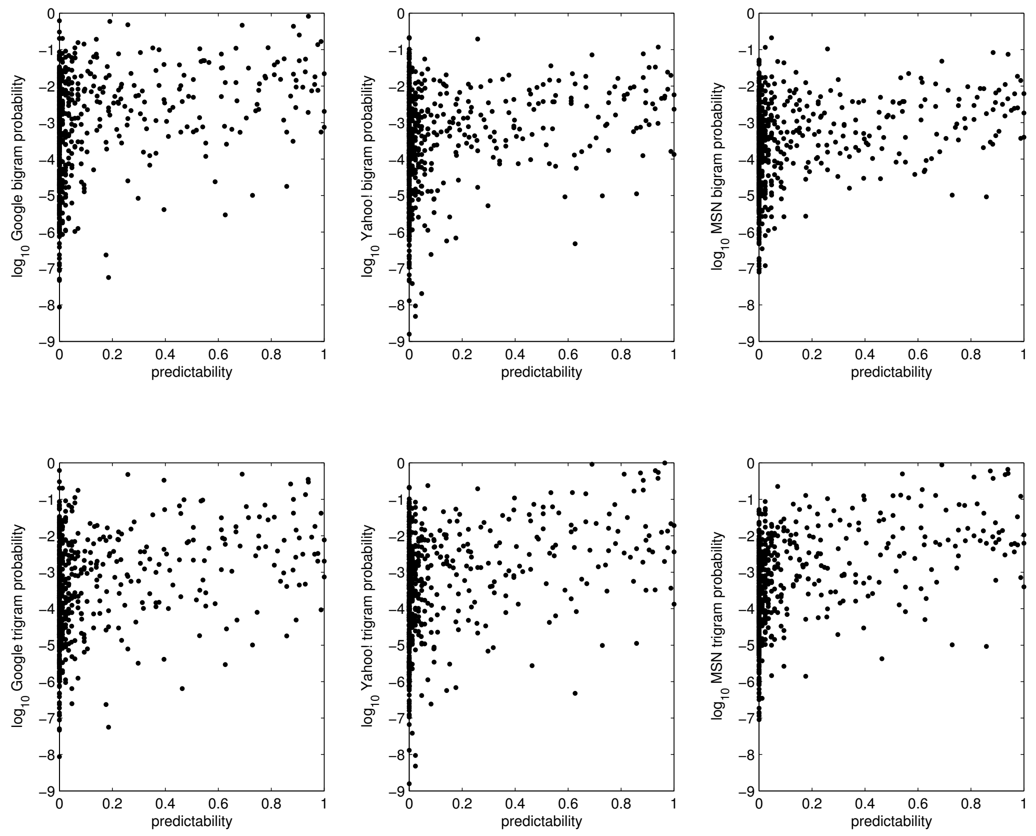

Figure 3.4: Relationship of predictability to word *n*-gram probability. For this plot, we backoff zero probabilities. Points with a low word *n*-gram probability are likely to have a low predictability value.

Chapter 4

Semantic and word *n*-gram measures combined

We have seen that neither the semantic nor the word *n*-gram measures are enough on their own to generate predictability values. The data support the hypothesis that both semantics and local relationships (as captured by word *n*-grams) contribute to predictability. The obvious next step is to determine whether a combination of the two types of measures capture enough information to generate good estimates of predictability.

4.1 Combination of web measures

In Figure 4.1, we can see the relationship of predictability to the web word *n*-gram measure (with a backoff of zero probabilities) and the web conditional co-occurrence measures. Note that the bigram and conditional co-occurrence measures are related by an inequality, since the occurrence of a word bigram implies that the two words co-occur. For the results from the Google and Yahoo! APIs, there is an obvious correlation between the measures following this inequality, while those obtained from the MSN API show a far less pronounced relationship. A lack of correlation implies that the two measures carry different information, and thus that they help in different ways to distinguish words that have a high and a low predictability. This is evident in the plots: In the Google and Yahoo! cases, the web conditional co-occurrence measure seems to carry little additional information, while in the MSN case, the two measures seem to both contribute important information allowing one to distinguish high from low predictabilities.

To assess quantitatively how well our web word *n*-gram and conditional co-occurrence measures model predictability, we can divide predictabilities into two categories: ‘high’ (greater than 0.02) and ‘low’ (less than 0.02). If our measures

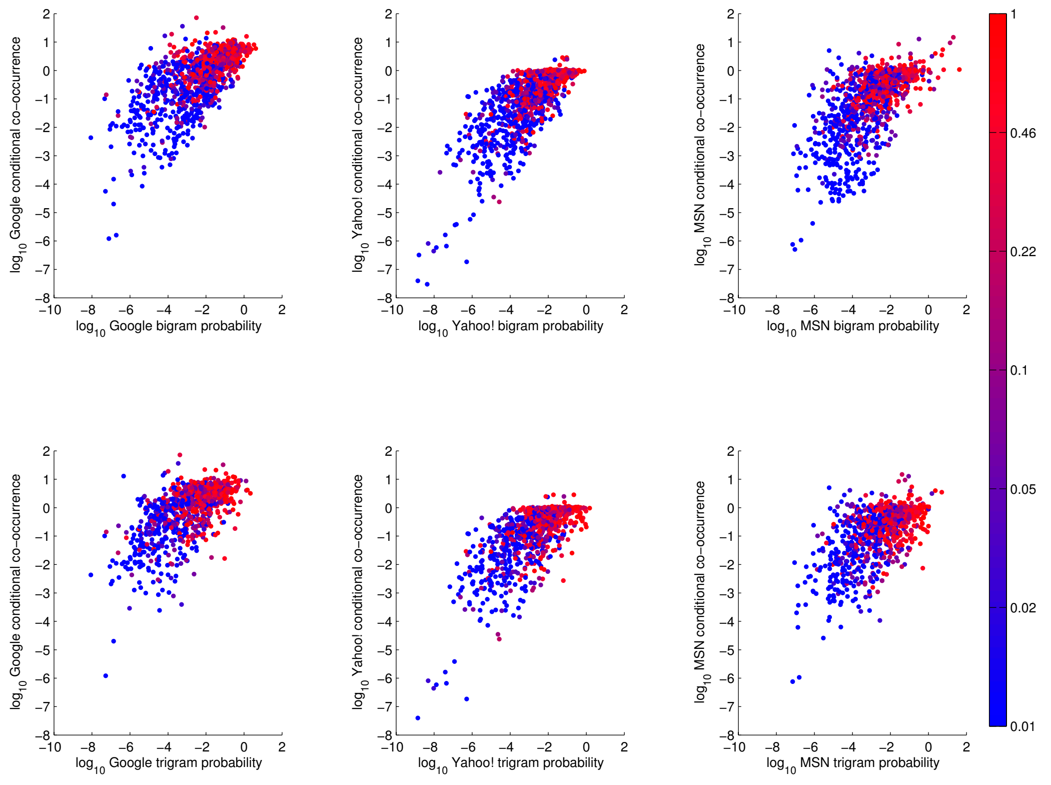

Figure 4.1: Relationship of predictability to web word *n*-gram and conditional co-occurrence measures. Points are coloured according to predictability, where the specific colour, shown on the colour bar, is determined by logarithmic interpolation between the two extremes.

are to make sense, high values of our *n*-gram and conditional co-occurrence measures should imply high predictabilities, while low values of the same measures should imply low predictabilities; if these inferences hold for a particular word, we will say that the word has been correctly classified. When one measure is high and the other is low, we will say that the resulting predictability remains unclassified. We are left to choose where our predictive measures should transition from ‘high’ to ‘low’; in the following analysis, we choose transition points to maximise the number of correctly classified cases.

For the three cases using web word bigram probability and web conditional co-occurrence probability, about 62% of the predictability values can be correctly classified, 15% remain unclassified, and the remaining 23% are incorrectly classified. Using the web word trigram probability and web conditional co-occurrence probability, about 58% can be correctly classified, while about 23% are unclassified, leaving 19% incorrectly classified.

To see what such results really mean, we also consider a reference case where the two predictors are not at all correlated with predictability. In this case, the percentage of cases classified correctly will equal the percentage of cases classified incorrectly. In this case, even if a word has been classified, we only have a 50% chance that the classification is correct. Using the combination of a web word *n*-gram measure and a web conditional co-occurrence probability, we have about 75% chance that a classified word has been correctly classified.

4.2 Combination of the web co-occurrence and the LSA measures

After replacing the web co-occurrence measure with the LSA measure, we obtain the plots shown in Figure 4.2. We apply the same quantitative analysis as the previous section, ignoring those words (mostly function words) where the LSA measure equals zero. The combination of a web word bi- or trigram measure with the LSA measure allows correct classification of roughly 55% of the predictability values, while about 27% remain unclassified, leaving us with 18% incorrectly classified.

From Section 2.2.6, we know that it makes sense to transform the raw LSA measure to an equivalent p-value via the empirical distribution of random vectors in our semantic space. After we transform the vertical axis (see Figure 4.3), we see a similar pattern to that seen with the web-based conditional co-occurrence estimates. The two measures appear to independently separate high from low predictability points. Unfortunately, there is an artifical lower cutoff visible on these graphs, caused by the limited number of random pairs of vectors sampled to generate the empirical distribution discussed in Section 2.2.6; if we had sampled more random pairs, it would have been possible to assign smaller p-values to high semantic similarity values. However, the plots suggest that we would need to sample many more random pairs (or guess an analytical distribution) to obtain a meaningful range of p-values.

Neither of the combinations of the short-range and semantic measures fully separate high from low predictability points, implying that these simple measures used alone cannot be used to perfectly model predictability. However, the fact that the clouds of high and low predictability points do not fully lie on top of each other implies that these measures do capture significant elements of predictability.

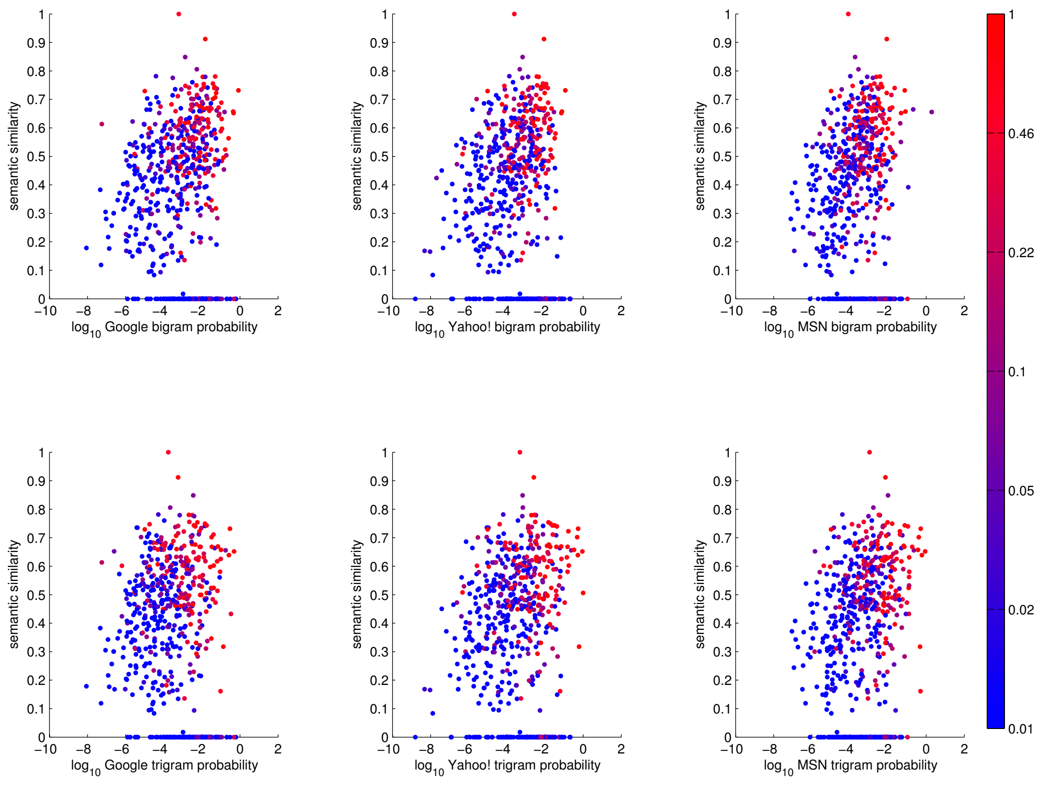

Figure 4.2: Relationship of predictability to the web word *n*-gram measure and the LSA measure. Points are coloured according to predictability, where the specific colour, shown on the colour bar, is determined by logarithmic interpolation between the two extremes.

4.3 Chapter summary

The semantic and word *n*-gram measures capture different aspects of predictability, but the combination is not sufficient to fully describe predictability.

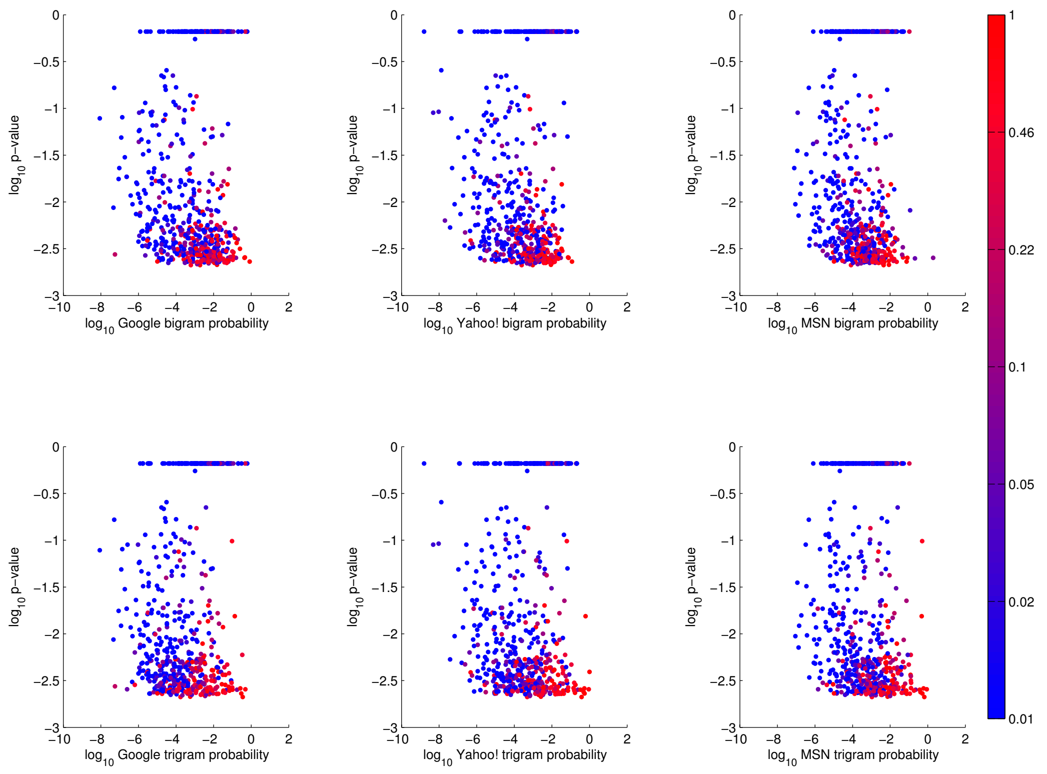

Figure 4.3: Relationship of predictability to the web word *n*-gram measure and the Latent Semantic Analysis p-value. Points are coloured according to predictability, where the specific colour, shown on the colour bar, is determined by logarithmic interpolation between the two extremes. The artificial lower p-value boundary is an artefact of the limited number of random word pairs sampled to generate an estimate of the p-value.

Chapter 5

Reversing SWIFT to test its lexical processing component

In this chapter, we detail a method to deduce the lexical processing component of a reading model, SWIFT, by using real eye movement data. In particular, we examine the role of predictability, and consider whether it is possible for other measures to take its place.

5.1 What is SWIFT?

SWIFT[1] is a mathematical model of eye movement control based on experimental and neurophysiological findings. The latest version, referred to as SWIFT-II [23], is an evolved version of the original SWIFT model [22], which I will refer to as SWIFT-I.

For each input sentence, SWIFT requires a number of parameters to be specified for each word—these parameters allow the model to 'recognise' important word features such as length and difficulty. After a number of internal 'subject-specific' parameters have been chosen, SWIFT generates a sequence of fixations and saccades. Figure 5.1 shows a schematic diagram of SWIFT.

5.2 Implementations of lexical processing

A number of methods have been used to allow lexical features to modulate the behaviour of eye movement models. We outline the lexical processing schemas used in SWIFT-II and SWIFT-I in order to motivate the Reverse SWIFT methodology described in Section 5.3. In addition, we propose a number of other possible ways

[1] Autonomous **S**accade-generation **W**ith **I**nhibition by **F**oveal **T**argets

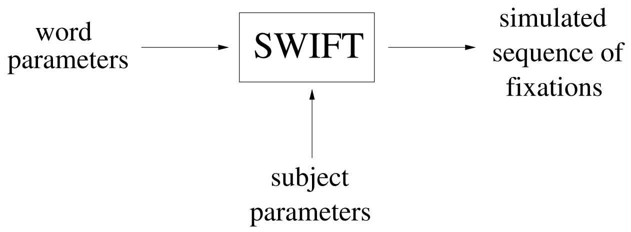

Figure 5.1: Block diagram of SWIFT

of combining word lexical features into a measure of lexical difficulty. These proposals will be tested in Section 5.4.

5.2.1 SWIFT-II

In SWIFT-II, the frequency[2] of word n (f_n) determines the required lexical processing of word n (L_n^{II}; here, the superscript indicates the version of the SWIFT model) in the following way:

$$L_n^{II} = \alpha \left(1 + \beta \frac{\ln f_n}{F} \right) \tag{5.1}$$

where α and β are subject-specific parameters, and F is a normalisation constant. Simplistically, if the amount of lexical processing for word n exceeds L_n^{II}, then this word is considered to have been completely processed. A sentence is considered to be completely processed when each word in it has been completely processed.

One of the important principles that distinguishes SWIFT from other eye movement models is its assumption of parallel processing of words that are present in the visual field. This is seen in the model as an asymmetrical spatial distribution of lexical processing about the fixation point. Two half-Gaussian functions with differing standard deviations (σ_L and σ_R) are used to capture the decrease of letter processing rate (λ) with increasing visual eccentricity (ε). Letter processing rate is normalised so that the total amount of processing (area underneath the function) equals one. The definition of λ is as follows:

$$\lambda(\varepsilon) = \sqrt{\frac{2}{\pi}} \frac{1}{(\sigma_R + \sigma_L)} \exp\left(-\frac{\varepsilon^2}{2\sigma^2} \right) \text{ with } \left\{ \begin{array}{ll} \sigma = \sigma_L, & \text{if } \varepsilon < 0 \\ \sigma = \sigma_R, & \text{if } \varepsilon \geq 0 \end{array} \right. . \tag{5.2}$$

Figure 5.2 shows this function with the most recent estimated values of σ_L and σ_R [23].

[2]The CELEX word frequency norms [6] are used in the SWIFT-II paper. To allow comparison, we will adopt these norms in the first part of our analysis.

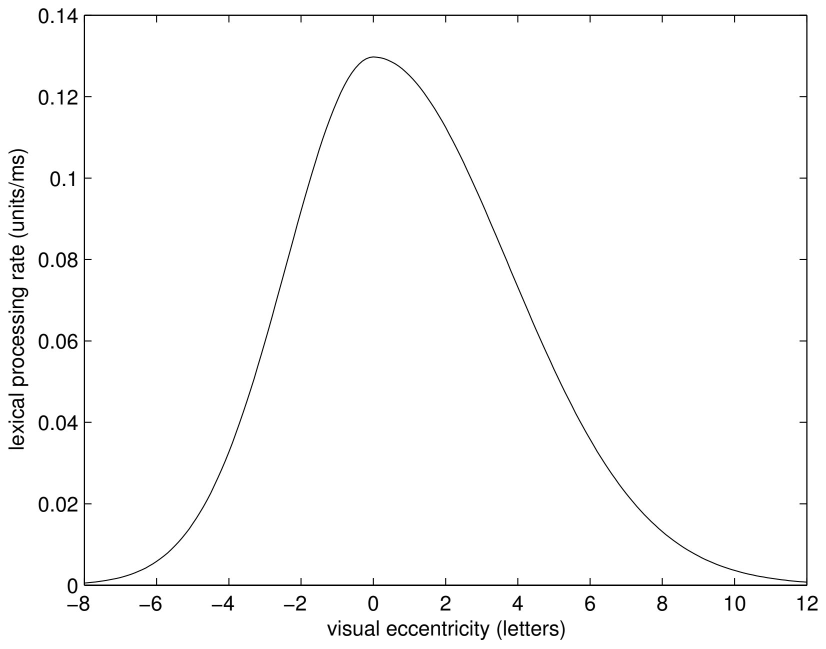

Figure 5.2: Lexical processing rate function with parameter values of $\sigma_L = 2.41$ and $\sigma_R = 3.74$.

Word processing rate (λ_n) is calculated as a parametrised weighted average of the processing rate of all letters in the word:

$$\lambda_n(t) = \frac{1}{(M_n)^{\eta}} \sum_{j=1}^{M_n} \lambda(\varepsilon_{nj}(t)), \tag{5.3}$$

where M_n is the number of letters in word n, and η is a parameter between 0 and 1. Ignoring the Gaussian noise that is subsequently added to include the stochastic nature of word processing, $\lambda_n(t)$ is a piecewise constant function. At some time t_a, we consider that word n has been completely processed[3] if

$$\int_0^{t_a} (1+\theta p_n)\lambda_n(t) \geq L_n^{II}, \tag{5.4}$$

where p_n is the Cloze predictability of word n, and θ is a subject-specific parameter.

[3]This is equivalent to the lexical completion phase detailed in the SWIFT-II paper [23]; the negligible global decay process (represented by parameter ω) has been excluded. We choose to ignore the preprocessing phase, which determines saccade targeting, because the large size of the preprocessing factor (parameter f) ensures that the amount of lexical processing required for preprocessing will always be much less than the amount required for lexical completion.

Taking into account that $\lambda_n(t)$ is piecewise constant and using definitions (5.1) and (5.3), we can rewrite condition (5.4) for the case where the subject has finished reading the sentence after N fixations:

$$\forall n, \sum_{i=1}^{N} \sum_{j=1}^{M_n} d_i \lambda(\varepsilon_{nj})|_i \geq \alpha \left(1 + \beta \frac{\ln f_n}{F}\right) \left(\frac{1}{1+\theta p_n}\right) (M_n)^\eta, \tag{5.5}$$

where d_i is the duration of fixation i, and $\lambda(\varepsilon_{nj})|_i$ is the amount of lexical processing on letter j of word n during fixation i.

5.2.2 SWIFT-I

In SWIFT-I, predictability, instead of modulating lexical processing rate directly, is found in the formula for required lexical processing:

$$L_n^I = (\alpha - \beta \ln f_n)(1 - \theta p_n), \tag{5.6}$$

where α, β and θ are subject-specific parameters. The criterion for complete lexical processing of a word, analogous to equation 5.4, is:

$$\int_0^{t_a} \lambda_n(t) \geq L_n^I \tag{5.7}$$

For the case where the subject has finished reading the sentence after N fixations, we can rewrite this as

$$\forall n, \sum_{i=1}^{N} \sum_{j=1}^{M_n} d_i \lambda(\varepsilon_{nj})|_i \geq (\alpha - \beta \ln f_n)(1 - \theta p_n)(M_n)^\eta. \tag{5.8}$$

Note that this condition is equivalent to taking the first-order Taylor expansion of condition (5.5), the condition used in SWIFT-II, about $p = 0$.

5.2.3 Additive form

One proposal, following Rayner, Ashby, Pollatsek and Reichle [51], is to combine predictability and frequency in an additive manner. Translating this to the SWIFT-II framework, we end up with the following condition:

$$\forall n, \sum_{i=1}^{N} \sum_{j=1}^{M_n} d_i \lambda(\varepsilon_{nj})|_i \geq (\alpha - \beta \ln f_n - \theta p_n)(M_n)^\eta. \tag{5.9}$$

5.2.4 Other possibilities

Following the lead of the statistical analysis of Kliegl, Nuthmann and Engbert [34], we could use a logit transformation[4] on the raw predictability values. Alternatively, we could take the logarithm of predictability or code predictability as a binary variable; the utility of a specific transformation should help us with the interpretation of predictability as a cognitive parameter. In addition, we have the option to altogether abandon the corpus word frequency and Cloze predictability parameters, replacing them with the web word frequency, web word *n*-gram and corpus-based semantic similarity measures derived in Chapters 2 and 3.

We will take a cursory look at these alternatives in Section 5.6 to determine whether it might be useful to use these in a lexical processing function.

5.3 The Reverse SWIFT method

Every sequence of fixations found in a sentence-reading eye-tracking experiment should reflect something about the sentence read. A word fixated more than once may be somehow ‘difficult’ to process—perhaps it is long, uncommon or unexpected given the previous sentence context. Conversely, a skipped word is likely to be somehow ‘easy’ to process—it may be short, have little semantic content or occur extremely frequently. In SWIFT, such considerations are taken into account in the lexical processing component, described above in Section 5.2. However, it is not at all clear that the current way of calculating the “lexical difficulty” parameter is a *correct* representation of word lexical difficulty for SWIFT, especially since SWIFT is a complex dynamical model.

There are many ways that one might validate SWIFT[5]. A typical way to validate the SWIFT model is to compare the properties of its generated fixation sequences with those of real eye movement data. One typical comparison is plotted in the SWIFT-II paper by Engbert, Nuthmann, Richter and Kliegl [23], showing

[4]The raw logit function as taken from Cohen and Cohen [15] is defined as logit $p_n = 0.5\ln[p_n/(1-p_n)]$. However, our predictability values are derived from the responses of only about 80 subjects, meaning that they include the boundary values of zero and one. Our arbitrary method of dealing with this problem, mentioned by Cohen and Cohen, is to "replace $p = 0$ by $p = 1/(2v)$ and $p = 1$ by $(2v-1)/(2v)$", where v is the denominator of the counted fraction (in our case, $v \approx 80$). Another way of dealing with the zero values is to use a backoff strategy such as Katz smoothing [27], but this reduces the independence of the predictability and the frequency estimates.

[5]Pitt [48] divides methods for the validation of models into local versus global and quantitative versus qualitative. Using this classification, Engbert [21] stated that global methods of validation are being neglected, and showed an example of applying global methods to the validation of SWIFT. However, he also mentioned that researchers regularly conduct local methods of validation, including goodness-of-fit, sensitivity analysis, cross-validation and hypothesis testing.

the variation of fixation duration (both real and simulated) with the frequency, predictability and length for the current, previous and next word; the nine plots correspond to nine possible goodness of fit statistics. These goodness of fit statistics could be used to choose between alternative proposals for lexical processing, but an improvement in one of these statistics is often accompanied by a worsening in another statistic. In addition, the inherent stochasticity of SWIFT leads to a certain amount of noise in the goodness of fit statistics.

We will explore another way of validating SWIFT, which is designed to specifically focus on the "lexical difficulty" parameter that is input to SWIFT. We will refer to this method as Reverse SWIFT; the method is effectively solving an *inverse problem* [61] of SWIFT. The idea of the Reverse SWIFT method (see Figure 5.3) is to assume that real experimental data is the result of the SWIFT model. From this data, we deduce the total lexical activation for each word, which should be related to the required lexical activation values. Then, we can test whether these total lexical activation values are related to the underlying lexical features.

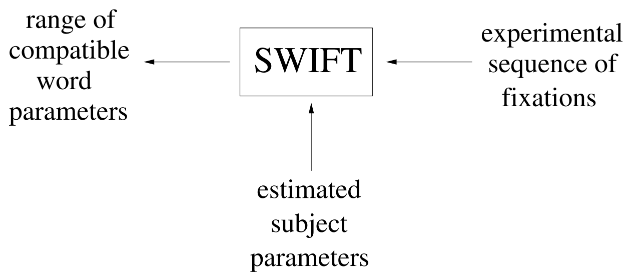
range of compatible word parameters
SWIFT
experimental sequence of fixations
estimated subject parameters

Figure 5.3: Block diagram of Reverse SWIFT

The Reverse SWIFT method requires one to estimate or select a few certain subject-specific parameters—these determine the shape and width of the lexical processing function. For simplicity, we will use the form of this function fitted in the SWIFT-II paper [23], which is supposed to represent an average subject.

It is important to note that the Reverse SWIFT method may not directly give a set of input parameters for SWIFT. Even though results from Reichle and Laurent [52] suggest that a skilled reader may be able to anticipate the amount of time required to process a word and thus program saccades to reduce 'unnecessary' fixation time, experimental data will still give a consistent overestimate of the amount of processing required for each word because subjects do not only look at a word for the minimal time required. This overestimation means that total lexical processing (T_n), which we define in Section 5.3.1, will always be greater than required lexical processing (L_n).

5.3.1 An example of the Reverse SWIFT method

Imagine that an experimental subject reading the sentence

```
Den Ton gab der Künstler seinem Gehilfen.
```

has produced the following sequence of fixations:

position (letters)	1	9	20	30
duration (ms)	192	168	160	180

where letter position 1 is the first letter of the sentence.

Let us assume that our subject is an 'average' subject, with a lexical processing function as postulated in the SWIFT framework, and with a 'normal' visual span:

Parameter	Symbol	Value
visual span, right	σ_R	3.74
visual span, left	σ_L	2.41

We calculate the cumulative sum of $\sum_{j=1}^{M_n} d_i \lambda(\varepsilon_{nj}(t))|_i$ for each fixation, which represents the cumulative value of the lexical processing for each word:

Fixation 1	**D**en 70.5	Ton 31.1	gab 4.6	der 0.1	Künstler 0	seinem 0	Gehilfen 0
Fixation 2	Den 71.9	Ton 62.1	**g**ab 66.3	der 27.4	Künstler 4.4	seinem 0	Gehilfen 0
Fixation 3	Den 71.9	Ton 62.1	gab 66.3	der 31.0	Kün**s**tler 133.3	seinem 13.4	Gehilfen 0
Fixation 4	Den 71.9	Ton 62.1	gab 66.3	der 31.0	Künstler 134.8	sein**e**m 114.0	Gehilfen 54.5

where fixated letters are shown in boldface.

We give this final cumulative sum vector the name *total lexical activation* and denote it T_n. For this particular example,

$$T_n = (71.9, 62.1, 66.3, 31.0, 134.8, 114.0, 54.5).$$

This total lexical activation is what interests us, since it should contain the effects of word lexical features on lexical processing. Note that this vector is exactly the quantity $\sum_{i=1}^{N} \sum_{j=1}^{M_n} d_i \lambda(\varepsilon_{nj})|_i$ on the left hand side of each of the final conditions in Section 5.2.

5.4 Relating total lexical activation to word lexical features

5.4.1 Data

In the initial analysis, eye movement data for subjects reading the Potsdam Sentence Corpus [33] were taken from the Kliegl group; Kliegl, Nuthmann and Engbert [34] have reported on the properties of this data set.

5.4.2 Initial inspection

We start with a cursory visual inspection of the data to detect obvious non-linear trends; we plot the median final lexical activation value for each word[6] against the variables of word length, CELEX word frequency and Cloze predictability (see Figure 5.4). Of course, these variables are highly interdependent; plots including more than one variable simultaneously are shown in Figures 5.5 and 5.6.

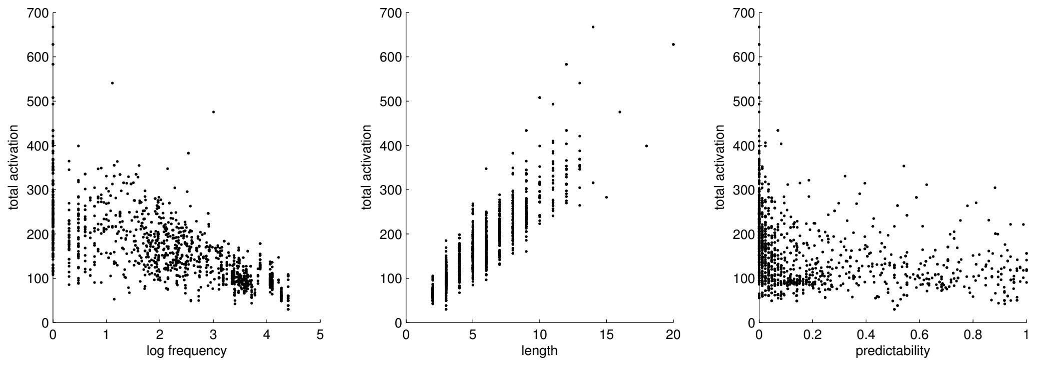

Figure 5.4: Median total lexical activation as a function of CELEX word frequency, length and predictability.

SWIFT-II's lexical activation model, given in condition (5.5), would be consistent with a continuous almost-linear gradient of lexical activation between those words that are long and have low frequency and predictability, and those words that are short and have high frequency and predictability. However, it is quite clear that this does not hold along the predictability axis; instead, we see that words with very high total activation will probably have a predictability value close to zero, but words with a moderate total activation could have any predictability value.

[6]Note that we assume an 'average' reading span by selecting appropriate values of σ_R and σ_L.

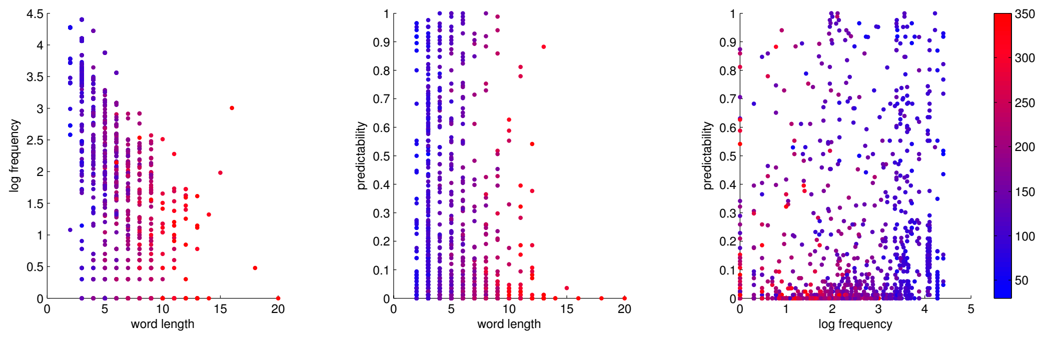

Figure 5.5: Median total lexical activation as a function of CELEX word frequency, length and predictability, taken pairwise. The colour scale denotes the amount of total lexical processing, with the minimum ($T_n = 30$) coded as blue, while the 99th percentile ($T_n = 350$) is coded as red.

5.4.3 Fitting the data to the proposed models

We now assess the appropriateness of each of the proposed lexical activation models by fitting our total lexical activation values according to the final conditions given in Section 5.2. The fitted functions are shown in Table 5.1; robust[7] regression was performed using the *rlm* function in the R environment [1], with recoding as necessary to eliminate nonlinear elements from the equations.

A simple robust regression analysis of log word length vs log total activation suggests a value of 0.900 (standard error 0.002) for the exponent η. This value is significantly different from the extremes of $\eta = 0$ and $\eta = 1$, which gives support to the postulate found in SWIFT-II that the exponent should be an intermediate value [23]. However, it is also significantly different to the $\eta = 0.448$ value obtained for SWIFT-II through the use of a genetic algorithm; this is not surprising, since the estimation of parameters in SWIFT-II is dependent on many factors, not just fixation durations.

It is clear that η, the exponent governing how word activation is derived from letter activation, is a far more important factor than the form of the lexical processing function. In addition, there is no evidence to suggest that the lexical processing function should have an additive form: the SWIFT-II lexical processing formula does not perform worse than either the SWIFT-I or the additive forms with the two tested values of η.

Given a specific value of η, the different forms of the lexical processing func-

[7]A robust regression is necessary because of the fact that fixation durations follow a gamma distribution, and this means that our total lexical activation measure is also far from being normally distributed.

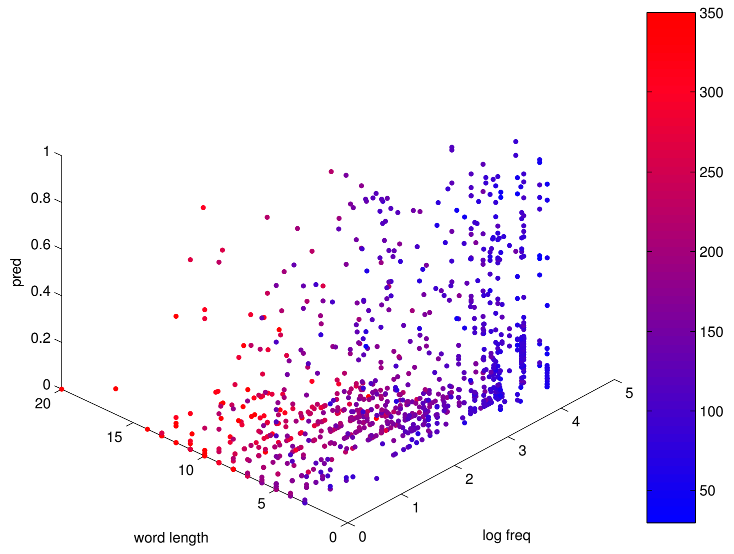

Figure 5.6: Median total lexical activation as a function of CELEX word frequency, length and predictability, shown simultaneously. The colour scale denotes the amount of total lexical processing, with the minimum ($T_n = 30$) coded as blue, while the 99th percentile ($T_n = 350$) is coded as red.

tion have very similar coefficients for frequency and word length, which reflects the fact that the fit of the equations is dominated by data points that have low predictability; when $p = 0$, there is no difference between the different proposals.

5.5 Another look at the form of the lexical processing function

So far, there has been little to distinguish the different proposed forms of the lexical processing formula. Ideally, we would like to vary each parameter in the equation independently and see the effect on the subsequent experimental reading pattern. However, it is not so easy to do this, since all word parameters are highly correlated with each other. There are a number of ways that one might try to manipulate lexical parameters; two ways are to:

- Swap individual words in a sentence, aiming to manipulate word parameters directly.
- Have subjects read the same group of sentences many times.

A problem with the first approach is that even a small change in a sentence may affect predictability values for the entire sentence fragment that follows the change.

proposal	η	**fit**	**SS**
original	0.448	$64\,(1-0.20\ln f_n)\left(\frac{1}{1+0.11p_n}\right)(M_n)^{0.448}$	—
SWIFT-II	0.448	$106\,(1-0.040\ln f_n)\left(\frac{1}{1+0.37p_n}\right)(M_n)^{0.448}$	$2.639e9$
SWIFT-I	0.448	$106\,(1-0.041\ln f_n)\,(1-0.27p_n)(M_n)^{0.448}$	$2.641e9$
Additive	0.448	$105(1-0.039\ln f_n-0.21p_n)(M_n)^{0.448}$	$2.646e9$
SWIFT-II	0.900	$39\,(1-0.004\ln f_n)\left(\frac{1}{1+0.31p_n}\right)(M_n)^{0.900}$	$2.512e9$
SWIFT-I	0.900	$39\,(1-0.005\ln f_n)\,(1-0.24p_n)(M_n)^{0.900}$	$2.513e9$
Additive	0.900	$39(1-0.004\ln f_n-0.24p_n)(M_n)^{0.900}$	$2.514e9$

Table 5.1: The original lexical processing function used in SWIFT-II, and different fits of total lexical activation using the proposed lexical processing functions. The fourth column shows the sum of squares residual left over after fitting.

In addition, a large number of subjects are required in order to distinguish intersubject differences from the particular effects of interest. However, it theoretically allows for a careful control of word parameters at the target word. In contrast, the rereading approach aims to increase predictability (and possibly frequency) values for most words in the sentences, but it is not clear exactly how these values should change quantitatively with repeated presentation. The major advantage of the rereading approach is that each subject generates an independent set of data, each of which should contain the effects of interest, allowing for the reduction of the effects of intersubject variability in a subsequent analysis.

We will test the different forms of the lexical processing formula with rereading data, with the motivation that the form of the formula should not need to change for this slightly unusual, but not abnormal, reading paradigm.

5.5.1 Rereading paradigm

In this study, eighteen subjects were asked to read the first thirty-two sentences from the Potsdam Sentence Corpus [33] a total of fifteen times; these were divided up into three sessions on different days, where each session consisted of five blocks of presentations of the thirty two sentences. It is important to note that subjects were told that they would be asked to reproduce the sentences that they had read at the end of each session[8]. The fact that recall improved across the sessions shows that subjects were successfully learning the sentences that they had read. The rereading data was collected by Kern for her Diplomarbeit [32], under the supervision of Kliegl. The study was carried out to see whether it is possi-

[8]Subjects were given the initial two or three words of each sentence as a cue.

ble to manipulate the effect of frequency and predictability on eye movements by allowing subjects to read the same text material many times.

Kern found a characteristic pattern of differences in the way subjects read familiar material: shorter total fixation time, shorter average fixation duration, increased saccade length, less fixations, less refixated words, more skipped words. Such a pattern implies that the text becomes less difficult with increased familiarity. Kern did not, however, map these results quantitatively to frequency and predictability, and the outcome of such an exercise would be conditional on the choice of the computational model used to generate eye movements from word lexical features.

In the forthcoming analysis, we will only consider the eye movement traces for the last (i.e. fifteenth) presentation.

5.5.2 Fitting rereading data to the proposed models

Similar to Section 5.4.3, we fit the total lexical activation of the rereading data with the proposed forms of the lexical processing formula. Table 5.2 shows the fitted functions. Again, we see that η is far more important than the form of the fit. Even with the manipulation of predictability, there seems to be little evidence to prefer one of the proposed forms of the lexical processing function to another.

proposal	η	**fit**	**SS**
SWIFT-II	0.448	$73\,(1-0.028\ln f_n)\left(\frac{1}{1+0.44p_n}\right)(M_n)^{0.448}$	$3.747e7$
SWIFT-I	0.448	$73\,(1-0.029\ln f_n)\,(1-0.30p_n)(M_n)^{0.448}$	$3.752e7$
Additive	0.448	$73(1-0.028\ln f_n-0.26p_n)(M_n)^{0.448}$	$3.753e7$
SWIFT-II	0.900	$28\,(1-0.011\ln f_n)\left(\frac{1}{1+0.36p_n}\right)(M_n)^{0.900}$	$3.607e7$
SWIFT-I	0.900	$27\,(1-0.011\ln f_n)\,(1-0.27p_n)(M_n)^{0.900}$	$3.609e7$
Additive	0.900	$28(1-0.011\ln f_n-0.30p_n)(M_n)^{0.900}$	$3.609e7$

Table 5.2: Different fits of total lexical activation for the rereading data using the proposed lexical processing functions. The fourth column shows the sum of squares residual left over after fitting.

5.6 Other approaches to forming a lexical processing function

Having found in the previous sections that the SWIFT-II, SWIFT-I and additive forms of the lexical processing function perform roughly the same, we will adopt the additive form for simplicity. In the following manipulations, we will leave η unchanged at 0.900, and attempt to fit the total lexical activation derived from normal reading.

5.6.1 Web vs corpus frequency norms

As a first trivial step to better fitting total lexical activation, we follow the lead of Keller [31] and replace the CELEX word frequency norms with the MSN frequency norms[9]. The resulting fits are only slightly better than the previous fit using CELEX frequency norms. This implies that a better resolution of very low frequencies does not greatly improve the quality of our fit. This may be a consequence of the fact that the robust regression reduces the effect of the extended tail of large values of total lexical activation, which originates from the gamma distribution of fixation durations. However, we have seen in Section 3.1.3 that web frequency norms are consistent with corpus norms at high frequencies, while performing better at low frequencies, so we will continue to use MSN frequency norms for the rest of this section.

5.6.2 Simple transformations of predictability

Next, we attempt to improve our fit of total lexical activation by transforming predictability in some way. Predictability is highly nonuniform in its distribution, with most values close to zero, reflecting the freedom available to express a concept through language. Here, we will try out three alternative transformations:

- logit, implying an underlying sigmoidal distribution of predictability
- log, implying an underlying power law distribution of predictability
- binary (0–1 or low–high coding), implying an inherent distinction between words with high and low predictability

The fits are shown in Table 5.3. The extreme predictability values of zero and one have been set to $1/160$ and $159/160$ before performing the log and logit transformations, following a suggestion in Cohen and Cohen [15]. For the binary transformation, we show the results with a cutoff at 0.03, although other cutoffs were tried, since this value gave the closest fit; with this cutoff, words are considered to have a low predictability if less than three of the seventy-odd Cloze subjects guessed the word correctly.

[9]The MSN web frequency norm was chosen because it does not suffer from the obvious lack of results in the frequency range between 1,000 and 10,000 that the other two web frequency norms have, as mentioned in Section 2.1.3.

variant	fit	SS
logit	$42\left(1-0.0072\ln f_n^{MSN}-0.00024 \text{ logit } p_n\right)(M_n)^{0.900}$	$2.535e9$
log	$32\,(1-0.049\ln p_n)\,(M_n)^{0.900}$	$2.502e9$
binary	$40\left(1-0.0008\ln f_n^{MSN}-0.13 \text{ bin } p_n\right)(M_n)^{0.900}$	$2.507e9$

Table 5.3: Different fits of total lexical activation using different transformations of predictability. The third column shows the sum of squares residual left over after fitting.

We can see that the logit transformation seems to be totally unsuitable. The log transformation performs well; in addition, the transformed predictability parameter absorbs variance otherwise explained by frequency. The fit with the binary predictability parameter outperforms the original fit, which highlights the observation that predictability cannot be treated as a linear parameter.

5.6.3 Semantic and *n*-gram measures

Now, we explore whether our derived measures can replace predictability in the fit of total lexical activation. Table 5.4 shows the fits using the MSN trigram probability with backoff (t_n) and the LSA measure (s_n).

variant	fit	SS
trigram	$30\left(1-0.005\ln f_n^{MSN}-0.022\ln t_n\right)(M_n)^{0.900}$	$2.519e9$
LSA	$48\left(1-0.11\ln f_n^{MSN}-0.228 s_n\right)(M_n)^{0.900}$	$2.478e9$
both	$37\left(1-0.017\ln f_n^{MSN}-0.039\ln t_n-0.29 s_n\right)(M_n)^{0.900}$	$2.462e9$

Table 5.4: Different fits of total lexical activation using our derived measures. The third column shows the sum of squares residual left over after fitting.

There is a significant but small improvement (residual sum of squares: $2.458e9$) when we add log predictability to the last fit, implying that the combination of trigram probability and the LSA measure almost totally explain the variance normally accounted for by predictability. On first glance, this result seems to be surprising, since we know that the combination of trigram and LSA measures do not model predictability perfectly. However, the major discrepancies between our measures and predictability occur for function words, where the LSA measure is almost always zero. Function words seem to have little weight in the current regression analysis since they are generally short. In preliminary investigations, it appears that when the lexical difficulty function is naïvely replaced with the total lexical activation function built with the trigram and LSA measures, SWIFT tends to systematically underestimate fixation durations on short high frequency words.

This suggests that the addition of a variable coding whether a word is a function or content word (for example, the ∞-norm) could greatly improve the estimates of lexical difficulty for SWIFT.

5.7 Chapter summary

SWIFT is a computational model that simulates eye movements during reading. One input required by SWIFT is the lexical difficulty of each word to be read. A number of proposals have been put forward regarding the exact way that word lexical features should be combined into a single lexical difficulty value. We use real eye movement data to calculate the total lexical activation for each word, which is a measure of the maximum amount of processing that could have occurred, assuming that the subject processes visual information like SWIFT does. We look at the relationship of our total lexical activation to word lexical features, finding that there is little difference between the three proposals that we consider. In an attempt to better distinguish between the proposals, we take a different set of real reading data, collected in a rereading paradigm, and apply the same procedure. Again, we find little difference between the three proposals.

Lastly, we try a number of other possible forms of the lexical processing function, firstly to see whether we obtain better results with a transformed version of predictability, and secondly, to test the ability of our derived measures to account for total lexical activation. The results suggest that log predictability may perform better than raw predictability, and that our derived measures may perform well enough to replace predictability altogether.

Chapter 6

Discussion

6.1 Can we compute predictability?

Cloze predictability is an important parameter because it acts as a window into the complex area of cognitive language processing. However, the compression of many different influences into a single parameter means that predictability itself should not be easy to create with a computational model. We have seen that the combination of a simple semantic measure with a simple word *n*-gram measure does not allow us to generate predictability values, although these measures are certainly related to predictability.

The relationship of predictability to our derived measures makes perfect sense in hindsight. We see that content words with high predictability have two properties: firstly, they are almost always semantically related to their context, and secondly, the two- and three-word sequences culminating in the target word occur quite frequently. However, a word may still be unpredictable even if it is semantically related to its context and commonly found following the previous one or two words, perhaps because there are many plausible alternatives, or because subjects guessed the wrong lexical category (e.g. noun instead of adjective). For function words, a very uncommon word sequence implies low predictability, but if a word sequence is moderately common, the word *n*-gram measure has no power to discriminate.

So, can we do better if we add extra information? Ideally, we would like to better predict cases with low predictability that score high with our semantic and word *n*-gram measures, meaning that we want a measure that shows *un*expectedness. One such measure is surprisal, suggested by Hale [29], which may be calculated from a probabilistic context-free grammar. Surprisal is the logarithm of the probability that the syntactic category of the target word follows the preceding syntactic context. Adding surprisal to our selection of measures should elegantly allow us to directly address syntactic constraints while also providing a measure of unexpectedness.

The other major problem is that of long-distance syntactic relationships. Although these occur in English, like in the sentence "He turned all of the lights in the room on.", they are far less frequent than in German, and thus such issues have had little visibility in mainstream computational linguistics. It may be possible to deal with many of these rather simply, by adding a skipping *n*-gram, where the last word in the pattern is at the end of the matrix clause, but not this does not cover all possible long-distance relationships; for example, in the sentence “Er hätte besser am Samstag nicht sollen arbeiten gehen.”, the words “hätte” and “sollen” are separated by a considerable distance, but “sollen” is not at the end of the sentence.

If surprisal and a long-distance syntactic measure were added to our semantic and *n*-gram measures, it is quite possible that we would be able to almost totally model predictability. Such an extension of the current research should not be difficult to pursue and has the potential to bear much fruit.

6.2 Implications for another application of semantic and *n*-gram measures

The application of a combination of semantic and *n*-gram measures is not restricted to the field of reading research. Wu, Berry, Shivakumar and McLarty [68] use such measures to classify protein sequences into families. They show their neural network performs best when they use a combination of the two types of measures. One of the main results that we have seen in language processing also occurs in protein classification: Their method is very good at allocating proteins to a particular family (with a specificity of over 90%), but poor at deciding that a foreign protein does not to belong to any of the trained families (with a sensitivity of around 50%). It seems that their method would also benefit from another measure that shows unrelatedness, although in this case, it is not clear what such a measure should be.

6.3 Is it possible to improve the lexical processing module in SWIFT?

One of the reasons for creating computational models of reading is to test certain hypotheses about the cognitive processing mechanisms underlying reading. In this dissertation, we have considered the proposition that word lexical features should be combined in a certain way to determine lexical difficulty. Our findings indicate that the current proposals about the form of the lexical processing function do not differ much in practice. However, other more fundamental considerations may have a considerable impact on the performance of the reading model: there is evidence that word lexical activation is closer to an average of letter lexical activations than a sum of letter lexical activations, and that predictability should not be treated as a linear parameter. There are other assumptions that can be examined: one currently debated assumption is the use of a logarithm to deal with the nonlinear nature of word frequency. Murray and Forster [46] suggest that rank frequency may be a more suitable measure than log frequency.

One suggestion by Radach and Kennedy [50] is to more closely integrate the development of eye movement and word recognition models, so that the output from a word recognition model can be directly used by an eye movement. One candidate that could probably be integrated with SWIFT is the SERIOL model [67], which uses five processing layers to represent different stages of word recognition; the lowest layer represents retinal detection of a stimulus, while the highest layer chooses the most likely word candidate when given component letter bigrams. The Reverse SWIFT method shows us the type of input that SWIFT requires to produce realistic fixation durations; this gives us a criterion to allow us to compare the suitability of different word recognition models, should it ever happen that it becomes possible to use such a model to provide inputs to SWIFT.

It is important to recognise that our analyses have been based on total viewing time. In contrast, the fitting of parameters in SWIFT is performed by using a genetic algorithm that considers the performance of the model in many different areas, including single and gaze durations and the fixation pattern. The Reverse SWIFT approach allows us to reduce the importance of the fixation pattern and to focus on the lexical difficulty module; once lexical difficulty is determined, the fixation pattern can still be varied by altering the word selection and saccade-performance modules in SWIFT, which are somewhat independent.

The semantic and word *n*-gram measures that we generated seem to perform just as well as predictability in representing lexical difficulty. If SWIFT performs just as well with such measures, this gives us far more freedom to generate simulations for other text passages. Most of the measures are extremely easy to collect; only the creation of the LSA measure is time-consuming and computationally demanding, but even this process is far more convenient than collecting new predictability norms from human subjects in a Cloze procedure.

6.4 Implications for other reading models

There are a number of computational reading models in competition with SWIFT; some of the major competing models are E-Z Reader [53], Glenmore [54] and SERIF [43]. These models are based on different hypotheses of the underlying cognitive processing occurring during reading. However, they are all similar in that they make very simple assumptions regarding the lexical difficulty of words; indeed, their focus on other parts of the reading process is understandable, since the models are not designed to test questions pertaining to lexical difficulty. A more suitable measure of lexical difficulty may in turn make differences between the models more obvious, while also improving model fits to real data.

In the E-Z Reader model (version 9), word lexical difficulty is an additive combination of log frequency and predictability, and the time required to complete word lexical processing is assumed to be proportional to lexical difficulty, with the inherent assumption that the word is processed as if it were a single entity located at the word centre. For this model, the results from our Reverse SWIFT analysis should carry across almost verbatim, in spite of the two lexical processing phases in E-Z Reader.

In the Glenmore model, lexical processing starts at the letter level. Word lexical processing is an average of the amount of processing of the component letters, and increased by a "positive self-recurrent connection that is proportional to the frequency of the word". Because the Glenmore model is implemented as a connectionist model, the core dynamics of the model tend to be hidden within the structure of the connections; it is probably unrealistic to directly apply an analogue of the Reverse SWIFT methodology to it. However, it might be possible to apply our procedure by ignoring the implementation and focussing only on the model's fundamental cognitive hypotheses.

The SERIF model has at its heart the assumption that foveal splitting is an important factor in reading. Apart from lexical processing rate being modulated linearly by log frequency, it is also affected by the visual familiarity of the word fragments on either side of fixation. This emphasis on foveal splitting means that it is almost impossible to apply a method like the Reverse SWIFT method, since reading dynamics are heavily dependent on the exact placement of every individual fixation. Any adaptation of the method would need to include measures like orthographic regularity [66] for variable-length sequences of letters at the beginning and the end of words.

6.5 Further work

One obvious extension of the work is to find out whether we can better compute predictability by adding a measure representing the unexpectedness of lexical category. This should be a simple matter of taking an existing syntactic parser and using it to generate predictions of lexical category given a previous context. As a means of cross-validation, a number of different syntactic parsers should be used to generate independent predictions.

If lexical category fails to significantly improve our ability to fit predictability, then more ingenious measures that reflect unrelatedness are needed. This will require more consideration of individual cases that have low predictability but are highly related through our semantic and word *n*-gram measures.

There is always space for finetuning our existing measures. It may be possible to obtain cleaner web measures by doing some more involved multiword querying; this may also allow us to better enforce a language restriction to the source data. The addition of long-range skipping to our word *n*-gram measure should slightly improve the ability of the measure to account for long-range syntactic constraints; however, some testing will be necessary to determine an appropriate skipping paradigm that does not severely increase the complexity of the model while providing a significant improvement in performance. The transformation of the LSA measure into an equivalent p-value would benefit from a more accurately estimated empirical angle distribution.

To discover shortcomings of the Reverse SWIFT method, it will be necessary to test the suggested forms of lexical difficulty by building them into the SWIFT model, although we must be cautious about whether we can directly specify parameters found in the lexical difficulty formula, since total lexical activation is an overestimate of the amount of processing required. We would expect that the resulting simulated fixation duration distributions should conform better to those distributions found in experimental data, and in particular, we should get more realistic effects of lexical features on fixation duration.

Appendix: Source code

In the following code, the punctuation sequence of five dots in a row, ‘`.....`’, has the special meaning that the line continues without a line break in the original code—the line break in these appendices has been artificially added to allow for the margins. The standard punctuation sequence for an ellipsis, ‘`...`’, has its normal meanings, i.e. either to show that some text or code has been excluded, or as a MATLAB operator to split long lines.

A Preprocessing

A.1 Collation

`collect`

Shown below is one of the `find` commands specific to the folder `2005`. It was repeated (with the obvious customisation) for each data folder. The file concatenates all of the contents of all nonexcluded files from each folder and outputs them to a file `text.X`, where `X` is a representation of the original folder name.

```
#!/bin/sh

echo Extracting 2005.

find ~/DATA/2005/. -type f \
 -not -name "inderzeit" \
 -not -name "*.pdf" \
 -not -name "*.zip" \
 -not -name "*.jpg" \
 -not -name "*.body" \
 -not -name "*.head" \
 -not -name "*.teihead" \
 -not -name "n_*_n" \
 -not -name "*Ihr*" \
 -not -name "*Sie_*" \
 -not -name "*_Sie*" \
 -not -name "*termin*" \
 -not -name "*ZEIT*" \
 -not -name "*ausf_9llen*" \
 -not -name "*Impressum*" \
 -not -name "*Schach*" \
 -exec cat {} \; > ~/datadir/text.2005
```

preproc1

The file `preproc1` reformats the text such that each paragraph (the text between the XML tags `<p...>` and `</p>`) occurs on a single line and is isolated from the text around it. The `sed` command was repeated for each input file `text.X`.

```
#!/bin/sh
# This file puts one paragraph on a line, and then
# combines the files into a large file called
# 'splitfile'.
#
# The order of commands is:
#   - add a space at the end of each line
#   - remove newline characters
#   - add newlines after </p> and before <p...>


echo
echo Creating splitfiles:
echo   2005
sed 's/$/ /g' \
 < ~/datadir/text.2005 | \
 tr -d '\n' | \
 sed -e 's_\(<\/p>\)_\1\n_g' -e 's_\(<p[^>]*>\)_\n\1_g' > \
 ~/datadir/newtext.2005

echo
echo Combining splitfiles.
cat ~/datadir/newtext.* > ~/datadir/splitfile
```

preproc2

The file `preproc2` extracts the required plain text from the original XML file.

```
#!/bin/sh
```

```
# This file preprocesses the splitfile to only leave
# the body text that I want.
#
# The preprocessing does the following:
#  - removes newline characters
#      (i.e. turns the file into one line)
#  - adds newlines before <p...> tags and after </p> tags
#  - only takes lines with both <p...> and </p> tags
#  - removes control characters
#  - takes care of the problematic characters '&<>'
#    - replace &amp; with &
#    - replace &#38 with &
#  - removes <p...> and </p> tags
#  - removes <b> and </b> tags
#  - removes <i> and </i> tags
#  - removes <br/> tags
#  - removes <hr/> tags
#  - removes <line/> tags
#  - removes <em>, </em> and <em/> tags
#  - removes <a...> and </a> tags
#  - removes <span...> and </span> tags
#  - removes <u...> and </u> tags
#  - removes <sub...> and </sub> tags
#  - removes <sup...> and </sup> tags
#  - removes <image...> and </image> tags
#      and all content
#  - removes <raw...> and </raw> tags
#      and all content
#  - removes <div...> and </div> tags
#      and all content
#  - removes <table...> and </table> tags
#      and all content
#  - removes <title...> and </title> tags
#      and all content
#  - removes <intertitle...> and </intertitle> tags
#      and all content
#  - removes <bu...> and </bu> tags
#      and all content
#  - removes <copyright...> and </copyright> tags
#      and all content
#  - removes <tt...> and </tt> tags
#      and all content
#
```

```
# NOTE!!! <strong> and </strong> tags need to be
# given special treatment
#  - removes <strong>, </strong> and <strong/> tags
#
#  - outputs the text to master.txt
#

echo
echo Processing tags.
awk ' /<p[^>]*>.*<\/p>/ \
{gsub( /[[:cntrl:]]/ , " "); \
 gsub( /&amp;/ , "\\&" ); \
 gsub( /&#38;/ , "\\&" ); \
 gsub( /<p[^>]*>|<\/p>/ , "" ); \
 gsub( /<b>|<\/b>/ , "" ); \
 gsub( /<i>|<\/i>/ , "" ); \
 gsub( /<br[^\/>]*\/>/ , " " ); \
 gsub( /<hr[^\/>]*\/>/ , " " ); \
 gsub( /<em[^>]*>|<\/em>|<em\/>/ , "" ); \
 gsub( /<a[^>]*>|<\/a>/ , "" ); \
 gsub( /<span[^>]*>|<\/span>/ , "" ); \
 gsub( /<u[^>]*>|<\/u>/ , "" ); \
 gsub( /<sub[^>]*>|<\/sub>/ , "" ); \
 gsub( /<sup[^>]*>|<\/sup>/ , "" ); \
 gsub( /<line\/>/ , " " ); \
 gsub( /<line[^>]*>/, "\n" ); \
 gsub( /<\/line>/, "\n" ); \
 gsub( /<image[^>]*>/, "\n&" ); \
 gsub( /<\/image>/, "&\n" ); \
 gsub( /<image[^>]*>.*<\/image>/ , "" ); \
 gsub( /<image[^>]*>.*$/ , "" ); \
 gsub( /<raw\/>/, "\n&" ); \
 gsub( /<raw[^>]*>/, "&\n" ); \
 gsub( /<raw[^>]*>.*<\/raw>/ , "" ); \
 gsub( /<\/raw>/, "\n" ); \
 gsub( /<div[^>]*>/, "\n&" ); \
 gsub( /<\/div>/, "&\n" ); \
 gsub( /<div[^>]*>.*<\/div>/ , "" ); \
 gsub( /<table[^>]*>/, "\n&" ); \
 gsub( /<\/table>/, "&\n" ); \
 gsub( /<table[^>]*>.*<\/table>/ , "" ); \
 gsub( /<title[^>]*>/, "\n&" ); \
 gsub( /<\/title>/, "&\n" ); \
 gsub( /<title[^>]*>.*<\/title>/ , "" ); \
 gsub( /<intertitle[^>]*>/, "\n&" ); \
 gsub( /<\/intertitle>/, "&\n" ); \
 gsub( /<intertitle[^>]*>.*<\/intertitle>/ , "" ); \
 gsub( /<bu[^>]*>/, "\n&" ); \
 gsub( /<\/bu>/, "&\n" ); \
 gsub( /<bu[^>]*>.*<\/bu>/ , "" ); \
 gsub( /<copyright[^>]*>/, "\n&" ); \
 gsub( /<\/copyright>/, "&\n" ); \
 gsub( /<copyright[^>]*>.*<\/copyright>/ , "" ); \
 gsub( /<tt[^>]*>/, "\n&" ); \
 gsub( /<\/tt>/, "&\n" ); \
 gsub( /<tt[^>]*>.*<\/tt>/ , "" ); \
# The next lines deal with special cases.
 gsub( /<strong>mann<\/strong>/ , "mann" ); \
 gsub( /<\/strong>mann/ , "mann" ); \
 gsub( /<strong>intel<\/strong>/ , "intel" ); \
 gsub( /re<strong>ise<\/strong>/ , "reise" ); \
 gsub( /AD<\/strong>HS/ , "ADHS" ); \
 gsub( /As<\/strong>sheuer/ , "Assheuer" ); \
 gsub( /Baumgär<\/strong>tel/ , "Baumgärtel" ); \
 gsub( /Benedi<\/strong>kt/ , "Benedikt" ); \
 gsub( /Biller<\/strong>beck/ , "Billerbeck" ); \
 gsub( /Bitt<\/strong>ner/ , "Bittner" ); \
 gsub( /Computer<\/strong>programm/ , .....
       "Computerprogramm" ); \
 gsub( /Coul<\/strong>mas/ , "Coulmas" ); \
 gsub( /Dampfinha<\/strong>lationen/ , .....
        "Dampfinhalationen" ); \
 gsub( /Dries<\/strong>chner/ , "Drieschner" ); \
 gsub( /El<\/strong>tern/ , "Eltern" ); \
 gsub( /Fernseh<\/strong>apparat/ , .....
       "Fernsehapparat" ); \
 gsub( /Fin<\/strong>ger/ , "Finger" ); \
 gsub( /Frei<\/strong>heit/ , "Freiheit" ); \
 gsub( /Gasch<\/strong>ke/ , "Gaschke" ); \
 gsub( /Ge<\/strong>er/ , "Geer" ); \
 gsub( /Har<\/strong>tung/ , "Hartung" ); \
 gsub( /Herzin<\/strong>ger/ , "Herzinger" ); \
 gsub( /Her<\/strong>zinger/ , "Herzinger" ); \
 gsub( /Jes<\/strong>sen/ , "Jessen" ); \
 gsub( /Küm<\/strong>mel/ , "Kümmel" ); \
 gsub( /Kn<strong>ei<\/strong>pe/ , "Kneipe" ); \
 gsub( /Krem<\/strong>pl/ , "Krempl" ); \
 gsub( /Ladur<\/strong>ner/ , "Ladurner" ); \
 gsub( /Michae<\/strong>lis/ , "Michaelis" ); \
 gsub( /Nachricht<\/strong>en/ , "Nachrichten" ); \
 gsub( /Nico<\/strong>demus/ , "Nicodemus" ); \
 gsub( /Pinz<\/strong>ler/ , "Pinzler" ); \
 gsub( /Ra<\/strong>disch/ , "Radisch" ); \
 gsub( /Ran<\/strong>dow/ , "Randow" ); \
 gsub( /Ro<\/strong>gowski/ , "Rogowski" ); \
 gsub( /Schwar<\/strong>zer/ , "Schwarzer" ); \
 gsub( /<strong>Si<\/strong>lja/ , "Silja" ); \
 gsub( /Spe<\/strong>zial/ , "Spezial" ); \
 gsub( /Stil<\/strong>seite/ , "Stilseite" ); \
 gsub( /<strong>Te<\/strong>yssen/ , "Teyssen" ); \
 gsub( /Ul<\/strong>rich/ , "Ulrich" ); \
 gsub( /Vorho<\/strong>lz/ , "Vorholz" ); \
 gsub( /<strong>ZE<\/strong>IT/ , "ZEIT" ); \
 print}' ~/datadir/splitfile |
gawk '\
{a = gensub( /([^[:alpha:]][[:alpha:]]).....
   (<strong>|<\/strong>|<strong\/>)/ , "\\1" , "g" ); \
 b = gensub( /(<strong>|<\/strong>|<strong\/>).....
   ([[:alpha:]]{1,2}[^[:alpha:]])/ , "\\2" , "g" , a ); \
 c = gensub( /<strong>|<\/strong>|<strong\/>/ , .....
   " " , "g" , b ); \
 print c}' > master.txt
```

A.2 Text cleaning and preparation

The extracted plain text needs to be cleaned up, because of the presence of special characters and strings. Then the resulting text is converted to lower case and numbers are replaced by a generic tag. This file is designed to work with a text file not necessary created by our XML parser, which is why control characters are removed again as a precautionary measure.

mkmaster1

```
#!/bin/sh
# This file cleans the original text file
# input:   master.txt
# outputs: master.newtxt
#            Text file with altered punctuation
#          master.lowertxt
#            Text file in lower case
#          master.nonum
#            Text file without numbers
#          master.newidngram
#            Id n-gram file without non-vocab words

clear

#################### !!! CAREFUL !!! ####################
#
# Use the following with great care, since it applies to
# all control characters ...
#
# Remove control characters and change them into spaces.
awk '{ gsub( /[[:cntrl:]]/ , " "); print }' master.txt > \
 master_noctrl.txt
#
# Do nothing.
#cp master.txt master_noctrl.txt
#
#########################################################

echo
echo 'Dealing with & coding.'
echo 'Dealing with &#number coding.'
echo 'Dealing with web and email addresses'
# Note that this | is entered in vim as CTRL-V x a0
# Note that this · is entered in vim as CTRL-V x b7
# Special symbols can be identified using :ascii
echo 'Removing characters !;*?()"«»[]|·ÂÝâ01¦¬'"':"
echo 'Removing miscellaneous free punctuation.'
echo 'Dealing with abbreviations that use full stops.'
echo 'Isolating full stops.'
echo 'Removing free colons.'
echo 'Removing word-bound commas.'

#
# 1. Deal with & coding and special symbol encoding:
#    - replace &lt; with a space
#    - replace &gt; with a space
#    - remove < and >
#    - replace &agrave; with a
#    - replace &copy; with <Copyrightsym>
#    - replace &Copy; with <Companysym>
#    - replace &frac12; with 1/2
#    - replace &Termine; with & Termine
#    - replace &uacute; with u
#    - replace Ã¤ with ä
#    - replace Ã¶ with ö
#    - replace Ã¼ with ü
#    - replace Ã with ß
#    - replace Ã© with é
#    - replace Ã§ with ç
awk '{gsub( /&lt;/ , " " ); \
      gsub( /&gt;/ , " " ); \
      gsub( /[<>]/ , " " ); \
      gsub( /&agrave;/ , "a" ); \
      gsub( /&copy(;)*/ , "<Copyrightsym>" ); \
      gsub( /&Co(;)*/ , "<Companysym>" ); \
      gsub( /&frac12;/ , "1/2" ); \
      gsub( /&Termine(;)*/ , "\\& Termine" ); \
      gsub( /&uacute;/ , "u" ); \
      gsub( /Ã¤/ , "ä" );
      gsub( /Ã¶/ , "ö" ); \
      gsub( /Ã¼/ , "ü" ); \
      gsub( /Ã /, "ß" ); \
      gsub( /Ã©/ , "é" ); \
      gsub( /Ã§/ , "ç" ); \
      gsub( /¼([[:graph:]]|$)/ , " " ); \
      gsub( /([[:graph:]]|^)¼/ , " " ); \
      gsub( /½([[:graph:]]|$)/ , " " ); \
      gsub( /([[:graph:]]|^)½/ , " " ); \
# 2. Deal with &#number coding:
#    - replace 13 with a space
#    - replace 36 with $
#    - replace 37 with %
#    - replace 39 (single quote) with a space
#    - replace 128 with <Eurosym>
#    - replace 130 (baseline single quote) with a space
#    - replace 132 (baseline double quote) with a space
#    - replace 145 (open single quote) with a space
#    - replace 146 (close single quote) with a space
#    - replace 147 (open double quote) with a space
#    - replace 150 (en dash) with a space
#    - replace 160 (&nbsp) with a space
#    - replace 163 with <Poundsym>
#    - replace 167 (section symbol) with s
#    - replace 171 («) with a space
#    - replace 175 (high horizontal bar) with a space
#    - replace 187 (») with a space
#    - replace 228 with ä
#    - replace 252 with ü
#    - replace 261 with c
#    - replace 263 with c
#    - replace 268 with C
#    - replace 269 with c
#    - replace 277 with e
#    - replace 281 with e
#    - replace 283 with e
#    - replace 287 with g
#    - replace 305 with i
#    - replace 321 with L
#    - replace 322 with l
#    - replace 324 with n
#    - replace 328 with n
#    - replace 338 with OE
#    - replace 339 with oe
#    - replace 344 with R
#    - replace 345 with r
#    - replace 346 with S
#    - replace 347 with s
#    - replace 350 with S
#    - replace 351 with s
#    - replace 352 with S
#    - replace 353 with s
#    - replace 376 with Y
#    - replace 378 with z
#    - replace 380 with z
#    - replace 381 with Z
#    - replace 382 with z
#    - replace 402 with f
#    - replace the following spacing symbols with nothing
#      - 710
#      - 711
#      - 728
#      - 729
#      - 730
#      - 731
#      - 732
#      - 733
#    - replace e769 with é
#    - replace a776 with ä
#    - replace u776 with ü
#    - replace 960 with <SmallPisym>
#    - replace 1111 with i
#    - replace the following with a space:
#      - 8211, 8212 (dashes)
#      - 8216, 8217, 8218, 8220, 8221, 8222
#          (apostrophes or quotes)
#      - 8224 (dagger)
#      - 8225 (double dagger)
#      - 8226 (bullet)
#      - 8230 (ellipsis)
#      - 8240 (per hundred)
#      - 8249, 8250 (angle brackets)
#    - replace 8364 with /
#    - replace 8364 with <Eurosym>
#    - replace 8482 with Ö
#    - replace 8486 with <Ohmsym>
#    - replace 8706 (partial derivative d) with d
#    - replace 8710 with <BigDeltasym>
#    - replace 8719 with <BigPisym>
#    - replace 8734 with y
#    - replace 8776 with z
#    - replace 8800 with s
#    - replace 8804 with r
#    - replace 8805 with r
#    - replace 9674 with <Diamondsym>
#    - replace 20250 with <Kanjichar>
#    - replace 21361 with <Kanjichar>
#    - replace 26426 with <Kanjichar>
#    - replace 64257 with fi
#    - replace 64257 with e
      gsub( /&#13;/ , " " ); \
      gsub( /&#36;/ , "$" ); \
      gsub( /&#37;/ , "%" ); \
      gsub( /&#39(;)*/ , " " ); \
      gsub( /&#128;/ , "<Eurosym>" ); \
      gsub( /&#130;/ , " " ); \
      gsub( /&#132;/ , " " ); \
      gsub( /&#145;/ , " " ); \
      gsub( /&#146;/ , " " ); \
      gsub( /&#147;/ , " " ); \
      gsub( /&#150;/ , " " ); \
      gsub( /&#160;/ , " " ); \
      gsub( /&#163;/ , "<Poundsym>" ); \
      gsub( /&#167;/ , "s" ); \
      gsub( /&#171;/ , " " ); \
      gsub( /&#175;/ , " " ); \
      gsub( /&#187;/ , " " ); \
      gsub( /&#228;/ , "ä" ); \
      gsub( /&#252;/ , "ü" ); \
      gsub( /&#261;/ , "a" ); \
      gsub( /&#263;/ , "c" ); \
      gsub( /&#268;/ , "C" ); \
      gsub( /&#269;/ , "c" ); \
      gsub( /&#277;/ , "e" ); \
      gsub( /&#281;/ , "e" ); \
      gsub( /&#283;/ , "e" ); \
      gsub( /&#287;/ , "g" ); \
      gsub( /&#305;/ , "i" ); \
      gsub( /&#321;/ , "L" ); \
      gsub( /&#322;/ , "l" ); \
      gsub( /&#324;/ , "n" ); \
      gsub( /&#328;/ , "n" ); \
      gsub( /&#338;/ , "OE" ); \
      gsub( /&#339;/ , "oe" ); \
      gsub( /&#344;/ , "R" ); \
      gsub( /&#345;/ , "r" ); \
      gsub( /&#346;/ , "S" ); \
      gsub( /&#347;/ , "s" ); \
      gsub( /&#350;/ , "S" ); \
      gsub( /&#351;/ , "s" ); \
      gsub( /&#352;/ , "S" ); \
      gsub( /&#353;/ , "s" ); \
      gsub( /&#376;/ , "Y" ); \
      gsub( /&#378;/ , "z" ); \
      gsub( /&#380;/ , "z" ); \
      gsub( /&#381;/ , "Z" ); \
      gsub( /&#382;/ , "z" ); \
      gsub( /&#402;/ , "f" ); \
      gsub( /&#710;/ , "" ); \
      gsub( /&#711;/ , "" ); \
      gsub( /&#728;/ , "" ); \
      gsub( /&#729;/ , "" ); \
      gsub( /&#730;/ , "" ); \
      gsub( /&#731;/ , "" ); \
      gsub( /&#732;/ , "" ); \
      gsub( /&#733;/ , "" ); \
      gsub( /e&#769;/ , "é" ); \
      gsub( /a&#776;/ , "ä" ); \
      gsub( /u&#776;/ , "ü" ); \
      gsub( /&#960;/ , "SmallPisym" ); \
      gsub( /&#1111;/ , "i" ); \
      gsub( /&#8211;/ , " " ); \
      gsub( /&#8212;/ , " " ); \
      gsub( /&#8216;/ , " " ); \
      gsub( /&#8217;/ , " " ); \
      gsub( /&#8218;/ , " " ); \
      gsub( /&#8220;/ , " " ); \
      gsub( /&#8221;/ , " " ); \
      gsub( /&#8222;/ , " " ); \
      gsub( /&#8224;/ , " " ); \
      gsub( /&#8225;/ , " " ); \
      gsub( /&#8226;/ , " " ); \
      gsub( /&#8230;/ , " " ); \
      gsub( /&#8240;/ , " " ); \
      gsub( /&#8249;/ , " " ); \
      gsub( /&#8250;/ , " " ); \
      gsub( /&#8260;/ , "/" ); \
      gsub( /&#8364;/ , "<Eurosym>" ); \
# Note that 8482 is normally a TM symbol
      gsub( /&#8482;/ , "Ö" ); \
      gsub( /&#8486;/ , "<Ohmsym>" ); \
      gsub( /&#8706;/ , "d" ); \
# Note that 8734 is normally a BigDelta symbol
      gsub( /&#8710;/ , "S" ); \
      gsub( /&#8719;/ , "<BigPisym>" ); \
# Note that 8734 is normally an Infinity symbol
      gsub( /&#8734;/ , "y" ); \
# Note that 8747 is normally an Integral symbol
      gsub( /&#8747;/ , "n" ); \
# Note that 8776 is normally an ApproxEqual symbol
      gsub( /&#8776;/ , "z" ); \
# Note that 8800 is normally a NotEqual symbol
      gsub( /&#8800;/ , "s" ); \
# Note that 8804 is normally a LessThanOrEqual symbol
      gsub( /&#8804;/ , "r" ); \
# Note that 8805 is normally a GreaterThanOrEqual symbol
```

```
      gsub( /&#8805;/ , "e" ); \
      gsub( /&#9674;/ , "<Diamondsym>" ); \
      gsub( /&#20250;/ , "<Kanjichar>" ); \
      gsub( /&#21361;/ , "<Kanjichar>" ); \
      gsub( /&#26426;/ , "<Kanjichar>" ); \
      gsub( /&#64257;/ , "fi" ); \
# note that 64258 is normally a fl symbol
      gsub( /&#64258;/ , "e" ); \
      gsub( /&/ , "<Ampersandsym>" ); \
       print }' master_noctrl.txt |
# 3. Deal with http://... and ftp://... and
#    email addresses
awk '{gsub( /http:\/\/[[:graph:]]+/ , .....
       " <webaddress> " ); \
      gsub( /[[:graph:]]*www\.[[:graph:]]+/ , .....
       " <webaddress> " ); \
      gsub( /[[:graph:]]+\.htm(l)*/ , .....
       " <webaddress> " ); \
      gsub( /ftp:\/\/[[:graph:]]*/ , .....
       " <webaddress> " ); \
      gsub( /([[:graph:]])+@([[:graph:]])+.....
       \.([[:graph:]])+/ , " <emailaddress> " ); \
# 4. Remove specified punctuation.
# Note that this must be altered rather carefully!!!
 gsub( /[\!;*()\"\\«\\»\[\]\|·ÂÝâ01¦¬{}]|:/ , " " );
# 5. Deal with abbreviations that use full stops:
#    - a.d.
#    - a.D.
#    - AG
#    - Anm.
#    - bzw.
#    - dgl.
#    - d.h.
#    - Dr.
#    - d.Red.
#    - e.h.
#    - e.V.
#    - Frankfurt a. M.
#    - GmbH
#    - Ltd.
#    - h.c.
#    - hrsg. v.
#    - i.H.
#    - i.R.
#    - KO
#    - k.u.k.
#    - n.Chr.
#    - OK
#    - p.a.
#    - P. O. Box
#    - Pty.
#    - q.e.d.
#    - s.d.
#    - s.u.
#    - Tel.
#    - u.a.
#    - u.ä.
#    - u.U.
#    - u.s.w.
#    - u.v.a.
#    - v.Chr.
#    - vgl.
#    - v.H.
#    - z.B.
#    - z.Hd.
#    - z.T.
#    - z.Zt.
 gsub( / a\. *d\.([^[:alnum:]]|$)/ , " <ad> " ); \
 gsub( / a\.? *D\.?([^[:alnum:]]|$)/ , " <aD> " ); \
 gsub( / [Aa]\.? *[Gg]\.?([^[:alnum:]]|$)/ , " <AG> " ); \
 gsub( / Anm *\.?([^[:alnum:]]|$)/ , " <Anm> " ); \
 gsub( / bzw\.([^[:alnum:]]|$)/ , " <bzw> " ); \
 gsub( / dgl *\.([^[:alnum:]]|$)/ , " <dgl> " ); \
 gsub( / [Dd]\. *h\.([^[:alnum:]]|$)/ , " <dh> " ); \
 gsub( / Dr *\.([^[:alnum:]]|$)/ , " <dgl> " ); \
 gsub( / [Dd]\. *[Rr]ed\.([^[:alnum:]]|$)/ , .....
  " <dRed> " ); \
 gsub( / [Ee]\. *h\.([^[:alnum:]]|$)/ , " <eh> " ); \
 gsub( / e\. *V\.([^[:alnum:]]|$)/ , " <eV> " ); \
 gsub( .....
  /[Ff]rankfurt *a *\.? *[Mm](ain)?\.?([^[:alnum:]]|$)/ , .....
  "<FrankfurtamMain> " ); \
 gsub( / G\.? *m\.? *b\.? *H\.?([^[:alnum:]]|$)/ , .....
  " <GmbH> " ); \
 gsub( / h\. *c\.([^[:alnum:]]|$)/ , " <hc> " ); \
 gsub( / [Hh]rsg\. *v(on)*\.([^[:alnum:]]|$)/ , .....
  " <hrsgv> " ); \
 gsub( / i\. *H\.([^[:alnum:]]|$)/ , " <iH> " ); \
 gsub( / i\. *R\.([^[:alnum:]]|$)/ , " <iR> " ); \
 gsub( / [Kk]\. *[Oo]\.([^[:alnum:]]|$)/ , " <KO> " ); \
 gsub( / k\. *u\. *k\.([^[:alnum:]]|$)/ , " <kuk> " ); \
 gsub( / n\. *[Cc]hr\.([^[:alnum:]]|$)/ , " <nChr> " ); \
 gsub( / Ltd *\.?([^[:alnum:]]|$)/ , " <Ltd> " ); \
 gsub( /( |$)[Oo]\.? *[Kk]\.?([^[:alnum:]]|$)/ , .....
  " <OK> " ); \
 gsub( / p\. *[Aa]\.([^[:alnum:]]|$)/ , " <pa> " ); \
 gsub( / P\.? *O\.? *Box([^[:alnum:]]|$)/ , " <POBox> " );\
 gsub( / Pty *\.?([^[:alnum:]]|$)/ , " <Pty> " ); \
 gsub( / q\. *e\. *d\.([^[:alnum:]]|$)/ , " <qed> " ); \
 gsub( / s\. *d\.([^[:alnum:]]|$)/ , " <sd> " ); \
 gsub( / s\. *u\.([^[:alnum:]]|$)/ , " <su> " ); \
 gsub( / Tel *\.?([^[:alnum:]]|$)/ , " <Tel> " ); \
 gsub( / u\. *a\.([^[:alnum:]]|$)/ , " <ua> " ); \
 gsub( / u\. *ä\.([^[:alnum:]]|$)/ , " <uä> " ); \
 gsub( / u\.? *s\.? *w\.?([^[:alnum:]]|$)/ , " <usw> " ); \
 gsub( / u\. *v\. *a\.([^[:alnum:]]|$)/ , " <uva> " ); \
 gsub( / u\. *U\.([^[:alnum:]]|$)/ , " <uU> " ); \
 gsub( / v\. *[Cc]hr\.([^[:alnum:]]|$)/ , " <vChr> " ); \
 gsub( / vgl *\.?([^[:alnum:]]|$)/ , " <vgl> " ); \
 gsub( / v\. *H\.([^[:alnum:]]|$)/ , " <vH> " ); \
 gsub( / z\.? *B\.?([^[:alnum:]]|$)/ , " <zB> " ); \
 gsub( / z\. *Hd*\.([^[:alnum:]]|$)/ , " <zHd> " ); \
 gsub( / z\. *T\.([^[:alnum:]]|$)/ , " <zT> " ); \
 gsub( / z\. *Zt*\.([^[:alnum:]]|$)/ , " <zZt> " ); \
       print }' |
# Now process strings of alternating capital letters and
# full stops:
gawk '{.....
 a = gensub( /([[:punct:][:space:]])([[:alpha:]]).....
      (\. *)([[:alpha:]])(\. *)([[:alpha:]])(\. *).....
      ([[:alpha:]])(\. )/ , .....
      " <initials\\2\\4\\6\\8> " , "g" ); \
 b = gensub( /([[:punct:][:space:]])([[:alpha:]]).....
      (\. *)([[:alpha:]])(\. *)([[:alpha:]])(\. *)/ , .....
      " <initials\\2\\4\\6> " , "g" , a ); \
 c = gensub( /([[:punct:][:space:]])([[:alpha:]]).....
      (\. *)([[:alpha:]])(\. *)/ , .....
      " <initials\\2\\4> " , "g" , b ); \
 print c}' |
# Note that the following single-letter replacements must
# come last:
#    - d.
#    - g.
#    - l.
#    - p.
#    - r.
#    - s.
#    - t.
#    - u.
#    - v.
awk '{gsub( / d\. *[^[:lower:][:digit:][:punct:]]/ , .....
       " <d> " ); \
      gsub( / g\. / , " <g> " ); \
      gsub( / l\. / , " <l> " ); \
      gsub( / p\. *[^[:alpha:][:punct:]]/ , " <p> " ); \
      gsub( / r\. / , " <r> " ); \
      gsub( / s\. / , " <s> " ); \
      gsub( / t\. / , " <t> " ); \
      gsub( / u\. / , " <u> " ); \
      gsub( / v\. / , " <v> " ); \
# 6. Remove certain sequences of punctuation.
      gsub( /\+\/-/ , " <plusminus> " ); \
      gsub( /([[:graph:]])*\+-([[:graph:]]*)/ , " " ); \
      gsub( /([[:space:]]|^)-+/ , " " ); \
      gsub( /-+([[:space:]]|$)/ , " " ); \
      gsub( /([[:space:][:punct:]]|^),+/ , " " ); \
      gsub( /,+([[:space:][:punct:]]|$)/ , " " ); \
      gsub( /\.\.+/ , " " ); \
      gsub( /\. \. \./ , " " ); \
      gsub( /([[:space:]\.]|^)'\''+/ , " " ); \
      gsub( /'\''+([[:space:]\.]|$)/ , " " ); \
      gsub( /([[:space:]]|^)\++([[:space:]]|$)/ , " " ); \
      gsub( /([[:space:]]|^)==+([[:space:]]|$)/ , " " ); \
 gsub( /([[:space:]]|^)@+[[:punct:]]*([[:space:]]|$)/,.....
  " " ); \
 gsub( /([[:space:]]|^)#+[[:punct:]]*([[:space:]]|$)/,.....
  " " ); \
      gsub( /\/+[[:punct:]]*([[:space:]]|$)/ , " " ); \
      gsub( /([[:space:]]|^)[[:punct:]]*\/+/ , " " ); \
      gsub( /\+\++/ , " " ); \
      gsub( /[-_]+([[:space:]]|$)/ , " " ); \
      gsub( /([[:space:]]|^)[-_]+/ , " " ); \
      gsub( /([[:space:]]|^)−+([[:space:]]|$)/ , " " ); \
      gsub( /([[:space:]]|ˆ)ª+([[:space:]]|$)/ , " " ); \
      gsub( /([[:space:]]|ˆ)÷+/ , " " ); \
      gsub( /÷+([[:space:]]|$)/ , " " ); \
      gsub( /\/\// , "/" ); \
      gsub( /([[:space:]]|^).....
            [äãåçéèêëíìîïnñóòôöõúùûü°¢£§¶ß®©Ö´‘~].....
            ([[:space:]]|$)/ , " " ); \
      print }' |
# 7. Put spaces before full stops, as well as adding
#    an initial full stop.
gawk '{.....
 a = gensub( /([[:graph:]])(\.)( |$)/ , \\1 . " , "g" ); \
 b = gensub( /([[:graph:]])(:)( |$)/ , "\\1 " , "g" , a );\
# 8. Treat commas specially.
 c = gensub( /([[:alpha:]])(,)([[:alpha:]])/ , .....
      "\\1 \\3" , "g" , b); \
# 9. Recapitalise Sebastian Herzog
 d = gensub( /sebastian herzog/ , .....
      "Sebastian Herzog" , "g" , c); \
# 10. Treat question marks specially.
 e = gensub( /([[:alpha:]])(\?)([^[:lower:]]|$)/ , .....
      "\\1 \\3" , "g" , d); \
 f = gensub( /([^[:alpha:]]|^)(\?)/ , "\\1 " , "g" , e); \
       print f}' > master.temptxt

rm -f master_noctrl.txt

echo 'Adding initial full stop.'
echo . | cat - master.temptxt > master.newtxt

rm -f master.temptxt
#
#########################################################

# Turn the whole file to lower case

echo
echo 'Turning file to lower case and removing extra.....
      spaces and full stops.'
gawk '{a = tolower($0) ; \
       b = gensub ( /( |^)(\.( |$))+/ , " " , "g" , a ); \
       c = gensub ( /[[:space:]]+/ , " " , "g" , b ); \
       print c }' master.newtxt > master.lowertxt

# Remove standalone numbers

echo
echo 'Replacing standalone numbers by <number>'
gawk '{.....
 gsub ( /(\<)([[:digit:][:punct:]])*([[:digit:]]).....
  ([[:digit:][:punct:]])*(\>)/ , "<number>" ); \
 gsub ( /[[:digit:]]+/ , "<number>" ); \
 print }' master.lowertxt > master.nonum
```

A.3 Conversion into the appropriate format

mkmaster2

The file `mkmaster2` uses the cleaned text to generate a vocabulary and frequency counts, as well as a file where all words have been replaced by their corresponding vocabulary number.

```
#!/bin/sh
# This file creates a vocabulary based on the
# preprocessed text file
# input:   master.nonum
# outputs: master.wfreq
#            Word frequency table
#          master.vocab
#            Vocabulary file
#          master.newvocab
#            Vocabulary file minus comments
#          master.numvocab
#            Vocabulary file with line numbers
#          master.tagtxt
#            Text file with tagged numbers
#          master.numtxt
#             Numerised text file


echo 'Creating word frequency table.'
text2wfreq < master.nonum > master.wfreq

echo 'Sorting word frequency table.'
sort -k 2 -n -o master.wfreqsort master.wfreq
```

```
echo
echo 'Creating vocab file.'
wfreq2vocab -gt 1 < master.wfreq > master.vocab


# To relate statistics back to the original corpus,
# remove comment lines from the vocab file.
grep -v '^##' master.vocab > master.newvocab

# The next command adds line numbers to the vocab file
nl master.newvocab > master.numvocab

##########################################################
#
echo
echo 'Substituting vocab numbers for words.'
#
# CAUTION!!!
# Note: The file is written such that long words are
# processed first, to prevent problems arising from
# non-full word replacements.
#
# First step is to tag all remaining digit sequences in
# the source text and vocab list with the tags <nums>
# (start of number) and <nume> (end of number).
echo
echo 'Tagging remaining digit sequences.'
awk '{gsub(/[0-9]+/,"<nums>&<nume>"); print }' .....
 master.nonum > master.tagtxt
awk '{gsub(/[0-9]+/,"<nums>&<nume>",$2); print }' .....
 master.numvocab > vocabtag.tmp
#
# Second step is to sort the vocab list in reverse
# word length order.
echo
echo 'Sorting vocab list in reverse order.'
awk '{print length($2) " " $1 " " $2}' vocabtag.tmp > .....
 length1.tmp
rm -f vocabtag.tmp
sort -r -n length1.tmp > length2.tmp
rm -f length1.tmp
awk '{print $2 " " $3}' length2.tmp > length3.tmp
```

```
rm -f length2.tmp
#
# Third step is to add \ to problem characters.
# The full list of problem characters is /.*[]^$ but
# we don’t have *[]^ any more.
echo
echo ’Adding backslash to problem characters.’
sed ’s:[\/\.\$]:\\&:g’ length3.tmp > length4.tmp
rm -f length3.tmp
#
# Fourth step is to create the basic sed replacement files.
#
echo
echo ’Creating basic sed replacement files.’
#
# 4.1: Create the full replacement file
echo ’Creating full replacement file.’
awk ’{ print "s! " $2 " !" $1 "!g"}’ length4.tmp > .....
 sedproc1.tmp
rm -f length4.tmp
#
# In the following command, it is assumed that the file
# ’finish_sedproc’ exists, which consists of the lines:
#
#  #! /bin/awk -f
#  { currlen = length($2)
#    if (currlen < maxlen && maxlen > 1) {
#      print "s![[:alpha:][:punct:]][[:graph:]]\\{" .....
#       currlen-1 "," maxlen-2 "\\}[[:alpha:][:punct:]]!0!g"
#      maxlen = currlen
#    }
#    print $0
#  }
#
finish_sedproc -v maxlen=1000 sedproc1.tmp > sedproc1_1.tmp
rm -f sedproc1.tmp
sed ’s/ //g’ sedproc1_1.tmp > sedproc1_2.tmp
rm -f sedproc1_1.tmp
#
# In the following command, it is assumed that the file
# ’lastline_sedproc’ exists, which consists of a single line:
#   s![^[:digit:][:space:]]!0!g
#
```

```
cat sedproc1_2.tmp lastline_sedproc > full_sedproc.tmp
rm -f sedproc1_2.tmp
#
rm -f seddir/*
rm -f progress_file
rm -f progress_subfile
rm -f master.numtxt
#
# 4.2: Split the process and the source text file up
#      into manageable pieces.
echo 'Splitting files.'
split -l 10000 full_sedproc.tmp seddir/fsed_
split master.tagtxt seddir/ftxt_
rm -f full_sedproc.tmp
#
# 4.3 Create the executable that will sequentially call
#     the pieces of the process.
echo 'Creating sedproc2.'
ls -1 seddir/fsed_* > sed_list
ls -1 seddir/ftxt_* > txt_list
#
# The following is a hack to allow use of two processors
awk '{print $0 " "}' sed_list | tr -d '\n' > sed_list2.tmp
rm -f sed_list
sed 's/seddir[[:graph:]]* seddir[[:graph:]]* /&\n/g' .....
 sed_list2.tmp > sed_list3.tmp
rm -f sed_list2.tmp

echo '#\!/bin/sh' > sedproc2.tmp
awk '{ \
 if (NF==2) {print "cat temp_in | sed -f " $1 " | .....
  sed -f " $2 " > temp_out; mv -f temp_out temp_in; .....
  echo Finished " $2 " >> progress_subfile ; .....
  date >> progress_subfile"} \
 else {print "cat temp_in | sed -f " $1 " > temp_out; .....
  mv -f temp_out temp_in; echo Finished " $1 " >> .....
  progress_subfile ; date >> progress_subfile"} \
     }' sed_list3.tmp >> sedproc2.tmp
chmod 700 sedproc2.tmp
#
# 4.4 Create the executable that will apply the process to
#     the pieces of the source text file.
echo 'Creating sedproc3.'
echo '#\!/bin/sh' > sedproc3.tmp
awk '{print "rm -f progress_subfile; cp -f " $0 " .....
 temp_in; sedproc2.tmp; cat temp_in >> master.numtxt; .....
 echo Finished " $0 " >> progress_file ; .....
 date >> progress_file" }' txt_list >> sedproc3.tmp
chmod 700 sedproc3.tmp
#
# 4.5 Run the process
echo 'Running process.'
nice time sedproc3.tmp &
#
rm -f sedproc2.tmp
rm -f sedproc3.tmp
#
#########################################################
```

mkmatlab

After we obtain the numerised form of the ZEIT corpus, a few small modifications are required to allow MATLAB to import it correctly.

```
#!/bin/sh
#########################################################
#
# Now we have to turn this file into the correct format
# for MATLAB processing. This requires us to add negative
# line numbers at the beginning of each line, and then to
# concatenate lines so that MATLAB will read in a
# (transposed) vector as input.
#
echo
echo Putting file in MATLAB format.
nl master.numtxt > master_linum.numtxt
awk '{ gsub(/\y./,"-&",$1); print }' .....
 master_linum.numtxt > matinput.tmp
rm -f master_linum.numtxt

paste -d ' ' -s matinput.tmp > matinput.txt
rm -f matinput.tmp
#
# And finally, we need to tell matlab how big the
# SVD matrix needs to be.
#
```

```
wc -l master.newvocab master.numtxt > matsize.tmp
awk '{print $1}' matsize.tmp > matsize.txt
rm -f matsize.tmp

echo
echo 'Creating id n-gram file.'
text2idngram -vocab master.newvocab -n 2 .....
 -write_ascii < master.newtxt > master.idngram

# Remove lines referring to characters not in the vocab.
echo
echo 'Removing non-vocab characters from id n-gram file.'
grep -wv 0 master.idngram > master.newidngram
```

B Approximate Latent Semantic Analysis

B.1 Creating the initial term–document matrix

`mksvd_preproc1.m`

The following code creates a term–document matrix from the original text.

```
clear;

% text is the imported (transposed) vector containing
% the numerised text with (negativised) line numbers

text = load('matinput.txt');

% w is the number of words in the training vocabulary
% l is the number of lines in the training text

matsize = load('matsize.txt');
w = matsize(1);
l = matsize(2);

clear matsize;

% occur is the word frequency matrix in each line
%occur = spalloc(w,l,size(text,2)-l);
occur = spalloc(w,l,30000000);
clear w l;
```

```
for n = 1 : size(text,2)
 if text(n) > 0
  occur(text(n),j) = occur(text(n),j) + 1;
 elseif text(n) < 0
  j = -text(n);
 end
end

save occur.mat occur
```

B.2 Log odds preprocessing

mksvd_preproc2.m

We split the initial term–document matrix into four to allow for memory limitations. After recoding it as a binary occurrence matrix instead of a frequency matrix, we repeatedly 'remove' rows and columns than only have one nonzero entry, since these rows and columns will contribute no useful information to our semantic space. Finally, we apply the log odds transformation.

```
clear;

% Initial sanity check for singleton words and then
% empty documents. Note that singleton words should
% already have been removed, so emptyword_top and
% emptywordmat_bot should be empty.
%
% The recursive removal of all eventually-singleton
% words and documents occurs later.

load occur.mat occur;
docmat = any(occur,1);
emptydoc = find(docmat == 0);
clear docmat;

ns = [1 : size(occur,2)]; ns(emptydoc) = [];

rowsplit = 210000;
colsplit = 314000;
save splits.mat rowsplit colsplit;

% The following lines cause a 0-1 occurrence coding to be
% used instead of frequency. Note that the inclusion of
% these lines is the choice of the person doing the
% analysis.

occur_top = spones(sparse(occur(1 : rowsplit,ns)));
%occur_top = sparse(occur(1 : rowsplit,ns));
save occur_top.mat occur_top; clear occur_top;
occur_bot = spones(sparse(occur(rowsplit + 1 : end,ns)));
%occur_bot = sparse(occur(rowsplit + 1 : end,ns));
save occur_bot.mat occur_bot; clear occur_bot;

clear;


load splits.mat rowsplit colsplit;
load occur_top.mat occur_top;
preproc_topleft = occur_top(:,1 : colsplit);
preproc_topright = occur_top(:,colsplit + 1 : end);
save preproc_topleft.mat preproc_topleft;
save preproc_topright.mat preproc_topright;
clear;

load splits.mat rowsplit colsplit;
load occur_bot.mat occur_bot;
preproc_botleft = occur_bot(:,1 : colsplit);
preproc_botright = occur_bot(:,colsplit + 1 : end);
save preproc_botleft.mat preproc_botleft;
save preproc_botright.mat preproc_botright;
clear;


%
% log odds preprocessing
%

% First, generate sparsity matrices; this takes into
% account the fact that the occurrence matrix may not
% already be 0-1 coded

load preproc_topleft;
sparsity_tl = spones(preproc_topleft);
clear preproc_topleft;

load preproc_topright;
```

```
sparsity_tr = spones(preproc_topright);
clear preproc_topright;

load preproc_botleft;
sparsity_bl = spones(preproc_botleft);
clear preproc_botleft;

load preproc_botright;
sparsity_br = spones(preproc_botright);
clear preproc_botright;

% find rows and columns that have less than two nonzero
% cells; we repeat the process until we hit no
% single-cell rows or columns

singles_t = [];
singles_b = [];
singles_l = [];
singles_r = [];
flag = 0;
row_singles = 0;
col_singles = 0;

while flag == 0
 nonsingles_l = [1:size(sparsity_tl,2)]';
 nonsingles_r = [1:size(sparsity_br,2)]';
 nonsingles_l(singles_l) = [];
 nonsingles_r(singles_r) = [];

 % rows
 elsum_tl = full(sum(sparsity_tl(:,nonsingles_l),2));
 elsum_tr = full(sum(sparsity_tr(:,nonsingles_r),2));
 elsum_bl = full(sum(sparsity_bl(:,nonsingles_l),2));
 elsum_br = full(sum(sparsity_br(:,nonsingles_r),2));
 elsum_t = elsum_tl + elsum_tr;
 elsum_b = elsum_bl + elsum_br;

 singles_t = find(elsum_t <= 1);
 singles_b = find(elsum_b <= 1);
 clear elsum_*;

 new_row_singles = length(singles_t) + length(singles_b);
 if new_row_singles > row_singles
```

```
  row_singles = new_row_singles;
 else
  flag = 1;
  break;
 end

 % columns

 nonsingles_t = [1:size(sparsity_tl,1)]';
 nonsingles_b = [1:size(sparsity_br,1)]';
 nonsingles_t(singles_t) = [];
 nonsingles_b(singles_b) = [];

 elsum_tl = full(sum(sparsity_tl(nonsingles_t,:),1));
 elsum_tr = full(sum(sparsity_tr(nonsingles_t,:),1));
 elsum_bl = full(sum(sparsity_bl(nonsingles_b,:),1));
 elsum_br = full(sum(sparsity_br(nonsingles_b,:),1));
 elsum_l = elsum_tl + elsum_bl;
 elsum_r = elsum_tr + elsum_br;

 singles_l = find(elsum_l <= 1);
 singles_r = find(elsum_r <= 1);
 clear elsum_*;

 new_col_singles = length(singles_l) + length(singles_r);
 if new_col_singles > col_singles
  col_singles = new_col_singles;
 else
  flag = 1;
%  break;
 end
end

save singles.mat singles_*;
save nonsingles.mat nonsingles_*;

clear;


% the following allows us to sample in a way as to
% avoid problematic rows and columns

load nonsingles;
load preproc_topleft;
colsum_topleft = ...
 full(sum(preproc_topleft(nonsingles_t,nonsingles_l),1));
rowsum_topleft = ...
 full(sum(preproc_topleft(nonsingles_t,nonsingles_l),2));
clear preproc_topleft;

load preproc_topright;
colsum_topright = ...
 full(sum(preproc_topright(nonsingles_t,nonsingles_r),1));
rowsum_topright = ...
 full(sum(preproc_topright(nonsingles_t,nonsingles_r),2));
clear preproc_topright;

load preproc_botleft;
colsum_botleft = ...
 full(sum(preproc_botleft(nonsingles_b,nonsingles_l),1));
rowsum_botleft = ...
 full(sum(preproc_botleft(nonsingles_b,nonsingles_l),2));
clear preproc_botleft;

load preproc_botright;
colsum_botright = ...
 full(sum(preproc_botright(nonsingles_b,nonsingles_r),1));
rowsum_botright = ...
 full(sum(preproc_botright(nonsingles_b,nonsingles_r),2));
clear preproc_botright;

colsum_l = [colsum_topleft + colsum_botleft];
colsum_r = [colsum_topright + colsum_botright];
rowsum_t = [rowsum_topleft + rowsum_topright];
rowsum_b = [rowsum_botleft + rowsum_botright];
clear colsum_top* colsum_bot* rowsum_top* rowsum_bot*;
save rowsum.mat rowsum_*;
save colsum.mat colsum_*;

totentries = sum(rowsum_t) + sum(rowsum_b);

load preproc_topleft;
[i,j,s]= find(preproc_topleft(nonsingles_t,nonsingles_l));
[m,n] = size(preproc_topleft);
clear preproc_topleft;
```

```
logodds_els_tl = log10(s .* (totentries - s) ./ ...
 ((rowsum_t(i)-s) .* (colsum_l(j)'-s)));
logodds_tl = sparse(nonsingles_t(i),nonsingles_l(j),...
 logodds_els_tl,m,n);
save logodds_tl.mat logodds_tl;

load preproc_topright;
[i,j,s]= find(preproc_topright(nonsingles_t,nonsingles_r));
[m,n] = size(preproc_topright);
clear preproc_topright;
logodds_els_tr = log10(s .* (totentries - s) ./ ...
 ((rowsum_t(i)-s) .* (colsum_r(j)'-s)));
logodds_tr = sparse(nonsingles_t(i),nonsingles_r(j),...
 logodds_els_tr,m,n);
save logodds_tr.mat logodds_tr;

load preproc_botleft;
[i,j,s]= find(preproc_botleft(nonsingles_b,nonsingles_l));
[m,n] = size(preproc_botleft);
clear preproc_botleft;
logodds_els_bl = log10(s .* (totentries - s) ./ ...
 ((rowsum_b(i)-s) .* (colsum_l(j)'-s)));
logodds_bl = sparse(nonsingles_b(i),nonsingles_l(j),...
 logodds_els_bl,m,n);
save logodds_bl.mat logodds_bl;

load preproc_botright;
[i,j,s]= find(preproc_botright(nonsingles_b,nonsingles_r));
[m,n] = size(preproc_botright);
clear preproc_botright;
logodds_els_br = log10(s .* (totentries - s) ./ ...
 ((rowsum_b(i)-s) .* (colsum_r(j)'-s)));
logodds_br = sparse(nonsingles_b(i),nonsingles_r(j),...
 logodds_els_br,m,n);
save logodds_br.mat logodds_br;

clear;


load logodds_tl.mat logodds_tl;
load logodds_tr.mat logodds_tr;
load logodds_bl.mat logodds_bl;
load logodds_br.mat logodds_br;
```

```
logodds = [logodds_tl logodds_tr; logodds_bl logodds_br];

save logodds.mat logodds;

clear;
```

B.3 Fast Monte Carlo Singular Value Decomposition

`fullapproxsvd1.m`

The file `fullapproxsvd1.m` carries out the row sampling and rescaling steps of the algorithm, and then calculates the singular values and left singular vectors of the sampled matrix.

```
clear;

%%% sampling rows %%%

load nonsingles.mat;

load logodds_tl.mat logodds_tl;
rowsqsum_tl = full(sum(logodds_tl(...
 nonsingles_t,nonsingles_l).^2,2));
nrows_t = size(logodds_tl,1);
clear logodds_tl;

load logodds_tr.mat logodds_tr;
rowsqsum_tr = full(sum(logodds_tr(...
 nonsingles_t,nonsingles_r).^2,2));
clear logodds_tr;

load logodds_bl.mat logodds_bl;
rowsqsum_bl = full(sum(logodds_bl(...
 nonsingles_b,nonsingles_l).^2,2));
nrows_b = size(logodds_bl,1);
clear logodds_bl;

load logodds_br.mat logodds_br;
rowsqsum_br = full(sum(logodds_br(...
 nonsingles_b,nonsingles_r).^2,2));
clear logodds_br;
```

```
nrows = nrows_t + nrows_b;
rowsqsum = [rowsqsum_tl + rowsqsum_tr; ...
 rowsqsum_bl + rowsqsum_br];

clear nrows_* rowsqsum_*;


load splits.mat rowsplit colsplit;

% samptimes: the number of times to do the sampling
samptimes = 5;

% samprows: the number of rows to sample
samprows = 4000;

save samppars.mat samptimes samprows;

rowchoicemat = zeros(samprows,samptimes);

cumrowsqsum = cumsum(rowsqsum);
rowchoiceprob = rowsqsum / cumrowsqsum(end);
rowpropmat = cumrowsqsum / cumrowsqsum(end);
fullrowchoiceprob = zeros(nrows,1);
fullrowchoiceprob([nonsingles_t;...
 nonsingles_b + rowsplit]) = rowchoiceprob;

clear rowsqsum rowchoiceprob;
save choiceprob.mat fullrowchoiceprob;

top_ns = length(nonsingles_t);

for timescount = 1 : samptimes
 rand('state',sum(100*clock));
 rowrandlist = sort(rand(samprows,1));
 rowpropplace = 1;
 for rowplace = 1 : samprows
  while rowpropmat(rowpropplace) < rowrandlist(rowplace)
   rowpropplace = rowpropplace + 1;
  end
  if rowpropplace <= top_ns
   rowchoicemat(rowplace,timescount) = ...
    nonsingles_t(rowpropplace);
  else
```

```
   rowchoicemat(rowplace,timescount) = ...
    nonsingles_b(rowpropplace - top_ns) + rowsplit;
  end
 end
end

clear timescount rowrandlist rowpropplace rowplace;
clear cumrowsqsum rowpropmat top_ns;
save choicemat.mat rowchoicemat;


%%% creating the weighted sampled matrix and %%%
%%% multiplying it with itself transposed    %%%

rowbk = zeros(samptimes,1);

for timescount = 1 : samptimes
 rowbk(timescount) = ...
  max(find(rowchoicemat(:,timescount) <= rowsplit));
end
save rowbk.mat rowbk;

for timescount = 1 : samptimes
 rcb = rowbk(timescount);
 rowmult = 1 ./ sqrt(fullrowchoiceprob(...
  rowchoicemat(:,timescount)) * samprows);
 save(['rowmult' int2str(timescount) '.mat'],'rowmult');
 toprm = sparse(diag(rowmult(1:rcb)));
 botrm = sparse(diag(rowmult(rcb+1:end)));
 clear rowmult;

 load logodds_tl.mat logodds_tl;
 temp_tl = logodds_tl(rowchoicemat(1:rcb,timescount),:);
 clear logodds_tl;
 [i,j,s] = find(temp_tl); [m,n]=size(temp_tl);
 temp_tl = sparse(i,j,s,m,n);

 load logodds_bl.mat logodds_bl;
 temp_bl = logodds_bl(rowchoicemat(...
  rcb+1:end,timescount) - rowsplit,:);
 clear logodds_bl;
 [i,j,s] = find(temp_bl); [m,n]=size(temp_bl);
 temp_bl = sparse(i,j,s,m,n);
```

```
temp_tltl = temp_tl * temp_tl';
temp_tlbl = temp_tl * temp_bl';
temp_blbl = temp_bl * temp_bl';

clear temp_tl temp_bl;

wtemp_tltl = toprm * temp_tltl * toprm;
wtemp_tlbl = toprm * temp_tlbl * botrm;
wtemp_blbl = botrm * temp_blbl * botrm;

clear temp_*;

load logodds_tr.mat logodds_tr;
temp_tr = logodds_tr(rowchoicemat(1:rcb,timescount),:);
clear logodds_tr;
[i,j,s] = find(temp_tr); [m,n]=size(temp_tr);
temp_tr = sparse(i,j,s,m,n);

load logodds_br.mat logodds_br;
temp_br = logodds_br(rowchoicemat(...
 rcb+1:end,timescount) - rowsplit,:);
clear logodds_br;
[i,j,s] = find(temp_br); [m,n]=size(temp_br);
temp_br = sparse(i,j,s,m,n);
clear i j s m n;

temp_trtr = temp_tr * temp_tr';
temp_trbr = temp_tr * temp_br';
temp_brbr = temp_br * temp_br';

clear temp_tr temp_br;

wtemp_trtr = toprm * temp_trtr * toprm;
wtemp_trbr = toprm * temp_trbr * botrm;
wtemp_brbr = botrm * temp_brbr * botrm;

clear temp_*;

diagelem = wtemp_tlbl + wtemp_trbr;

RRT = [wtemp_tltl + wtemp_trtr, diagelem; ...
 diagelem', wtemp_blbl + wtemp_brbr];
 eval(['wrsq' int2str(timescount) ' = RRT;']);
 clear wtemp_* RRT;
end

save wrsq.mat wrsq*;

clear;

load samppars;

% the following is a slightly more complicated form of
%
%[v,d] = eig(full(wrsq#)); d = sparse(d);
%save wb_eig#.mat d v; clear wrsq d v;
%
% where # is replaced by the sample number

for timescount = 1 : samptimes
 load('wrsq.mat',['wrsq' int2str(timescount)]);
 eval(['wrsq = full(wrsq' int2str(timescount) ');']);
 eval(['clear wrsq' int2str(timescount) ';']);
 [v,d] = eig(wrsq); d = sparse(d);
 save(['wb_eig' int2str(timescount) '.mat'],'d','v');
end

clear wrsq d v;

% the following generates the following code
%d#=full(real(sqrt(diag(d))));

for timescount = 1 : samptimes
 load(['wb_eig' int2str(timescount) '.mat'],'d');
 eval(['d' int2str(timescount) .....
  '= full(real(sqrt(diag(d))));']);
end
clear d;
save eigvals.mat d*;
```

fullapproxsvd2.m

The file `fullapproxsvd2.m` reconstructs the desired approximation to the right eigensystem of the original matrix.

```
%%% truncate and find right eigenvectors of R %%%

clear;
load rowbk.mat rowbk;
load choicemat.mat rowchoicemat;
load samppars.mat samptimes;
load splits.mat rowsplit;
load nonsingles.mat nonsingles_l nonsingles_r;


% ntrunc defines the reduced dimension
ntrunc = 200;
% I have set the following parameter manually to
% values between 1 and 5 due to disk space
% limitations
timescount = 1;
rcb = rowbk(timescount);

%for timescount = 1 : samptimes

% calculate R' * v / d in the following way:
%   A(sampled rows)' * diag(row weighting) * v * diag(1/d)
% thus diag(row weighting) * v * diag(1/d) is a
% multiplier for A', which we will call 'rightmult'

load('eigvals.mat',['d' int2str(timescount)]);
eval(['d = d' int2str(timescount) ';']);
eval(['clear d' int2str(timescount)  ';']);
[sortd,indexd] = sort(d,1,'descend');
rec_dtrunc = sparse(diag(1 ./ sortd(1 : ntrunc)));
clear d sortd;

load(['wb_eig' int2str(timescount) '.mat'],'v');
vtrunc = v(:,indexd(1 : ntrunc));
clear v;

load(['rowmult' int2str(timescount) '.mat'],'rowmult');

rightmult = diag(rowmult) * vtrunc * rec_dtrunc;
rightmult_t = rightmult(1:rcb,:);
rightmult_b = rightmult(rcb+1:end,:);
clear rightmult rowmult vtrunc rec_dtrunc;
```

```
load logodds_tl.mat logodds_tl;
temp_tl = logodds_tl(rowchoicemat(1:rcb,timescount),:);
clear logodds_tl;
[i,j,s] = find(temp_tl); [m,n]=size(temp_tl);
clear temp_tl; temp_tlT = sparse(j,i,s,n,m);
clear i j s m n;
topH_t = full(temp_tlT * rightmult_t);
clear temp_tlT;

load logodds_bl.mat logodds_bl;
temp_bl = logodds_bl(rowchoicemat(...
 rcb+1:end,timescount) - rowsplit,:);
clear logodds_bl;
[i,j,s] = find(temp_bl); [m,n]=size(temp_bl);
clear temp_bl; temp_blT = sparse(j,i,s,n,m);
clear i j s m n;
botH_t = full(temp_blT * rightmult_b);
clear temp_blT;

H_t = topH_t + botH_t;
save H_t.mat H_t;
clear topH_t botH_t H_t;

load logodds_tr.mat logodds_tr;
temp_tr = logodds_tr(rowchoicemat(1:rcb,timescount),:);
clear logodds_tr;
[i,j,s] = find(temp_tr); [m,n]=size(temp_tr);
clear temp_tr; temp_trT = sparse(j,i,s,n,m);
clear i j s m n;
topH_b = full(temp_trT * rightmult_t);
clear temp_trT;

load logodds_br.mat logodds_br;
temp_br = logodds_br(rowchoicemat(...
 rcb+1:end,timescount) - rowsplit,:);
clear logodds_br;
[i,j,s] = find(temp_br); [m,n]=size(temp_br);
clear temp_br; temp_brT = sparse(j,i,s,n,m);
clear i j s m n;
botH_b = full(temp_brT * rightmult_b);
clear temp_brT;

H_b = topH_b + botH_b;
```

```
clear topH_b botH_b;
save H_b.mat H_b;
```

`mktruncA`

The file `mktruncA` converts the Potsdam corpus to a vector, where words are replaced by their corresponding vocabulary number.

```
#!/bin/sh
# This file creates a vector containing corresponding
# vocabulary numbers of the words in the Potsdam corpus,
# in order of appearance.
awk '{gsub (/[[:cntrl:]]/,"\n");print}' .....
 POTSTEXTPC.TXT > potslines.txt
# Note that the next command works because the only
# unwanted punctuation is the comma.
awk '{gsub (/\./," -1"); \
      gsub (/ /,"\n"); gsub (/,/,""); \
      a = tolower($0); \
      print a}' potslines.txt > potswords.txt
echo '#\!/bin/sh' > potsnumscript
echo 'rm -f potsnums.txt' >> potsnumscript
awk 'NF > 0 { $1 == -1 ? .....
 a = "echo -1 >> potsnums.txt" : .....
 a = "awk '\''/[[:space:]]" $0 "$/{print $1 + 0}'\'' .....
  master.numvocab >> potsnums.txt; .....
  echo -2 >> potsnums.txt"; \
 print a}' potswords.txt >> potsnumscript
chmod 700 potsnumscript
potsnumscript
```

`mkordnum`

The numerised Potsdam corpus is imported into MATLAB. For each sentence in the Potsdam corpus, the relevant rows of the log odds transformed occurrence matrix are extracted and saved to a separate file in the directory `Adir`.

```
load potsnums.txt
pots_ordnums = zeros(144,11);
i = 1; j = 1; flag = 0;

for m = 1 : length(potsnums)
 curr = potsnums(m);
 if curr == -2
  if flag == 0
   flag = 1;
  else
   flag = 0;
   pots_ordnums(i,j) = 0;
  end
  j = j + 1;
 elseif curr == -1
  i = i + 1;
  j = 1;
  flag = 0;
 else
  pots_ordnums(i,j) = curr;
  flag = 0;
 end
end

clear potsnums i j m flag curr;
save pots_ordnums.mat pots_ordnums;

load logodds.mat logodds;
load nonsingles.mat;
load splits.mat rowsplit colsplit;

ns_lr = [nonsingles_l; nonsingles_r + colsplit];
ns_tb = [nonsingles_t; nonsingles_b + rowsplit];

clear nonsingles_*;

for sentence = 1 : 144
 A = zeros(11,length(ns_lr));
 for word = 1 : 11
  if pots_ordnums(sentence,word) > 0
   A(word,:) = logodds(pots_ordnums(sentence,word),ns_lr);
  end
 end
 A = sparse(A);
 save(['Adir/A' int2str(sentence) '.mat'],'A');
end
```

fullremult.m

The file `fullremult.m` carries out the last multiplication, which constructs an approximate vector representation in the low-dimensional semantic space for each word in the Potsdam corpus.

```
clear;
load H_t.mat H_t;
load H_b.mat H_b;
load splits.mat colsplit;
load nonsingles.mat nonsingles_l nonsingles_r;

for sentence = 1 : 144
 load(['Adir/A' int2str(sentence) '.mat'], 'A');
 AH = A(:,1:length(nonsingles_l)) * ...
  H_t(nonsingles_l,:) + ...
  A(:,length(nonsingles_l)+1:end) * H_b(nonsingles_r,:);
 AHH = ([H_t(nonsingles_l,:) * AH'; ...
  H_b(nonsingles_r,:) * AH'])';
 save(['AHHdir/AHH' int2str(sentence) '.mat'], 'AHH');
end
```

C Reverse SWIFT

The following script calculates the total lexical activation, given an input file containing the sequence of fixations of a number of subjects reading sentences once only.

```
% LOADING DATA

% potsdam.mat contains raw corpus data
% We only use
%  satz.wl:  word length including punctuation
%  satz.wl2: word length excluding punctuation
load potsdam.mat satz;

% allsac.mat contains reading data
% We only use the first five columns, which are
%  subject
%  sentence
%  fixated word
%  fixated letter (of word)
%  fixation duration
```

```
load allsac.mat allsac;
allsac = allsac(:,1:5);

% LEXICAL ACTIVATION FUNCTION

leftletter = -8;
rightletter = 12;

x = [leftletter:0.1:rightletter];
sig_r = 3.74;
sig_l = 2.41;
lamb_0 = 1/(sqrt(pi)*(sig_r+sig_l)/sqrt(2));

lamb = zeros(length(x),1);

for ind1 = 1:length(x)
 if x(ind1) < 0
  lamb(ind1)=lamb_0*exp(-x(ind1)*x(ind1)/(2*sig_l*sig_l));
 else
  lamb(ind1)=lamb_0*exp(-x(ind1)*x(ind1)/(2*sig_r*sig_r));
 end
end

clear sig_r sig_l decay lamb_0 ind1;


% CALCULATE ACTIVATIONS

% missing:holds missing subject data in the following form:
%         subject session block sentence
missing = [];
misscount = 0;

% these maximum values are used to determine how many
% loops are required for the main program, and the size of
% certain matrices
maxsubj=max(allsac(:,1));
maxsent=max(allsac(:,2));
maxword=max(allsac(:,3));

% maxfix:arbitrary estimate of the max number of fixations
%        per sentence
maxfix = 24;
```

```
% totact(word,fixation,subject,sentence) is the total
% processing done on each sentence for each trial
totact = zeros(maxword,maxfix,maxsubj,maxsent);

% fixseq(fixation,subject,sentence) holds the
% fixation position in characters for a particular fixation
fixseq = zeros(maxfix,maxsubj,maxsent);

% fixdur(fixation,subject,sentence) holds the
% fixation duration in msec for a particular fixation
fixdur = zeros(maxfix,maxsubj,maxsent);


for sentence = 1:maxsent
 % wordlen:word lengths (without punctuation) of the
 %         sentence of interest
 wordlen = satz(sentence).wl2;

 % wordlen_p:word lengths (with punctuation) of the
 %               sentence of interest
 wordlen_p = satz(sentence).wl;

 % numword:number of words in the current sentence
 numword = length(wordlen);

 % wordstart:starting positions of each word
 wordstart = ones(1,numword);
 for ind2 = 2:numword
  wordstart(ind2) = wordstart(ind2-1)+wordlen_p(ind2-1)+1;
 end

 % numchar:number of characters in the current sentence
 numchar = wordstart(numword)+wordlen_p(numword)-1;

 % finding the lines that correspond to Sentence Z
 indSZ = find(allsac(:,2) == sentence);

 % taking only the relevant rows of allsac
 SZ = allsac(indSZ);

 subject = 0;
```

```
% read in fixseq and fixdur
for count = 1:length(indSZ)
 if subject ~= SZ(count,1)
  subject = SZ(count,1);
  fixnum = 1;
 else
  fixnum = fixnum + 1;
 end

 fixseq(fixnum,subject,sentence) = ...
  wordstart(SZ(count,3)) + SZ(count,4) - 1;
 fixdur(fixnum,subject,sentence) = ...
  SZ(count,5);
end

clear count subject fixnum indSZ SZ ind2;

% process fixation patterns one at a time
for subject = 1:maxsubj
 subfixseq = fixseq(:,subject,sentence);

 % note if an expected data entry is missing, and move on
 % to the next subject if it is
 if max(subfixseq) == 0
  misscount = misscount + 1;
  missing(misscount,1)=subject;
  missing(misscount,2)=sentence;
  continue;
 end

 % remove terminal zeros from the sequence of fixations
 while (subfixseq(length(subfixseq)) == 0)
  subfixseq = subfixseq(1:length(subfixseq)-1);
 end

 % calculate lexical processing for each fixation
 for fixnum = 1:length(subfixseq)

  % stepact: lexical processing over letters
  % wordact: lexical processing 'averaged' over words
  % leftprocword: leftmost word to be lexically processed
  % rightprocword:rightmost word to be lexically processed
  % minind: leftmost character to be lexically processed
  % maxind: rightmost character to be lexically processed
  % relminind: relative index of minind
  % relminind: relative index of maxind
  stepact = zeros(1,numchar);
  wordact = zeros(1,numword);
  leftprocword = max([1 ...
   max(find(wordstart <= subfixseq(fixnum)+leftletter))]);
  rightprocword = ...
   max(find(wordstart <= subfixseq(fixnum)+rightletter));
  minind = ceil(max(1,subfixseq(fixnum)+leftletter));
  maxind = floor(min(wordstart(numword) + ...
   wordlen(numword),subfixseq(fixnum)+rightletter));
  relminind = max(find(x <= minind-subfixseq(fixnum)));
  relmaxind = max(find(x <= maxind-subfixseq(fixnum)));

  % xref: the x-value in the lambda function that
  %       corresponds to the leftmost letter being
  %       lexically processed
  xref = find(x==minind-subfixseq(fixnum));

  % allocate appropriate lexical processing to letters
  stepact(minind:maxind) = lamb(relminind:10:relmaxind);

  % total lexical processing across each word
  for ind3 = leftprocword:rightprocword
   startchar = wordstart(ind3);
   endchar = startchar + wordlen(ind3) - 1;
   wordact(ind3) = sum(stepact(startchar:endchar));
  end

  % copy previous fixation activations to current fixation
  if fixnum > 1
   totact(:,fixnum,subject,sentence) = ...
    totact(:,fixnum-1,subject,sentence);
  end

  % add current fixation activations
  for ind4 = leftprocword:rightprocword
   totact(ind4,fixnum,subject,sentence) = ...
    totact(ind4,fixnum,subject,sentence) + ...
    wordact(ind4)*fixdur(fixnum,subject,sentence);
  end
end
```

```
  for fixnum = length(subfixseq)+1:maxfix
   totact(:,fixnum,subject,sentence) = ...
    totact(:,fixnum-1,subject,sentence);
  end
 end
end

save totact.mat totact maxsubj maxsent maxword maxfix;
```

Bibliography

[1] `http://www.r-project.org/`.

[2] All Our N-Gram Are Belong To You. `http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html`.

[3] Das Digitale Wörterbuch der deutschen Sprache des 20. Jahrhunderts, Berlin-Brandenburgischen Akademie der Wissenschaften. `http://www.dwds.de/textbasis/kerncorpus/`.

[4] Google acquires Applied Semantics. `http://www.google.com/press/pressrel/applied.html`.

[5] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional space. In *Proceedings of the 8th international conference on database theory*, pages 420–434, 2001.

[6] R. H. Baayen, R. Piepenbrock, and H. van Rijn. The CELEX lexical database (release 1) [CD-ROM], 1993. Linguistic Data Consortium, University of Pennsylvania, Philadelphia, PA.

[7] M. Banko and E. Brill. Mitigating the paucity-of-data problem: Exploring the effect of training corpus size on classifier performance for natural language processing. In *Proceedings of the Conference on Human Language Technology*, 2001.

[8] J. R. Bellegarda. Exploiting latent semantic information in statistical language modeling. *Proceedings of the IEEE*, 88:172–175, 2000.

[9] M. W. Berry. Large scale singular value computations. *International Journal of Supercomputer Applications*, 6:13–49, 1992.

[10] M. W. Berry, M. Browne, A. N. Langville, V. P. Pauca, and R. J. Plemmons. A survey of algorithms and applications for the nonnegative matrix factorization. *Computational Statistics and Data Analysis*, Submitted 2006.

[11] M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37:573–595, 1995.

[12] N. Chomsky. *Syntactic structures*. Mouton de Gruyter, Berlin, 2nd edition, 2002.

[13] P. R. Clarkson and R. Rosenfeld. Statistical language modeling using the CMU-Cambridge toolkit. In *Proceedings ESCA Eurospeech*, 1997. `http://mi.eng.cam.ac.uk/~prc14/toolkit.html`.

[14] N. Coccaro and D. Jurafsky. Towards better integration of semantic predictors in statistical language modeling. In *ICSLP-98, Sydney, Australia*, 1998.

[15] J. Cohen and P. Cohen. *Applied multiple regression/correlation analysis for the behavioral sciences*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1975.

[16] H. M. Collins. The Editing Test for the Deep Problem of AI. *Psycologuy*, 8(1), 1997. `http://www.cogsci.soton.ac.uk/cgi/psyc/newpsy?8.01`.

[17] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41:391–407, 1990.

[18] P. Drineas, R. Kannan, and M. W. Mahoney. Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication. Technical Report YALEU/DCS/TR-1269, Yale University, 2004.

[19] P. Drineas, R. Kannan, and M. W. Mahoney. Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix. Technical Report YALEU/DCS/TR-1270, Yale University, 2004.

[20] S. Dumais. Enhancing performance in latent semantic indexing retrieval. Technical Report TM-ARH-017527, Bellcore, 1992.

[21] Ralf Engbert. Model analysis, selection, and comparison: Theoretical concepts and some illustrations from SWIFT, 2006. ECRP Workshop, Eye Movements in Reading: Computational Models & Corpus Analyses.

[22] Ralf Engbert, Andre Longtin, and Reinhold Kliegl. A dynamical model of saccade generation in reading based on spatially distributed lexical processing. *Vision Research*, 42:621–636, 2002.

[23] Ralf Engbert, Antje Nuthmann, Eike Richter, and Reinhold Kliegl. SWIFT: A dynamical model of saccade generation during reading. *Psychological Review*, 112:777–813, 2005.

[24] C. Fellbaum, editor. *WordNet: An electronic lexical database*. MIT Press, 1998.

[25] J. R. Firth. A synopsis of linguistic theory. In F. R. Palmer, editor, *Selected Papers of J. R. Firth: 1952–1959*. Longman, London, 1968.

[26] H. Goodglass. Agrammatism in aphasiology. *Clinical Neuroscience*, 4(2):51–56, 1997.

[27] J. T. Goodman. A bit of progress in language modeling. Technical Report MSR-TR-2001-72, Machine Learning and Applied Statistics Group, Microsoft Research, 2001.

[28] T. L. Griffiths, M. Steyvers, D. M. Blei, and J. B. Tenenbaum. Integrating topics and syntax. In *Advances in neural information processing systems*, volume 17, 2005.

[29] J. Hale. A probabilistic Earley parser as a psycholinguistic model. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics*, 2001.

[30] F. Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556, 1976.

[31] F. Keller and M. Lapata. Using the web to obtain frequencies for unseen bigrams. *Computational Linguistics*, 29(3):459–484, 2003.

[32] S. Kern. Eye movements during repeated reading, 2002. Diplome thesis, Universität Potsdam.

[33] R. Kliegl, E. Grabner, M. Rolfs, and R. Engbert. Length, frequency, and predictability effects of words on eye movements in reading. *European Journal of Cognitive Psychology*, 16(1/2):262–284, 2004.

[34] R. Kliegl, A. Nuthmann, and R. Engbert. Tracking the mind during reading: The influence of past, present, and future words on fixation durations. *Journal of Experimental Psychology: General*, 135:12–35, 2006.

[35] A. Kramer. Herrscher über das Chaos. *c't magazin für computer technik*, 9:178–182, 2006. `http://ctmagazin.de/`.

[36] T. K. Landauer and S. T. Dumais. A solution to Plato's problem: The Latent Semantic Analysis theory of the acquisition, induction, and representation of knowledge. *Psychological Review*, 104:211–240, 1997.

[37] T. K. Landauer, P. W. Foltz, and D. Laham. An introduction to Latent Semantic Analysis. *Discourse Processes*, 25:259–284, 1998.

[38] T. K. Landauer, D. S. McNamara, S. Dennis, and W. Kintsch, editors. *Handbook of Latent Semantic Analysis*. Lawrence Erlbaum Associates, Mahwah, NJ, 2007.

[39] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

[40] W. Lowe. Towards a theory of semantic space. In *Proceedings of the 21st Annual Conference of the Cognitive Science Society*, 2001.

[41] B. Manaris, L. Pellicoro, G. Pothering, and H. Hodges. Investigating Esperanto's statistical proportions relative to other languages using neural networks and Zipf's law. In *Proceedings of the 2006 IASTED International Conference on ARTIFICIAL INTELLIGENCE AND APPLICATIONS (AIA 2006)*, 2006.

[42] J. L. McClelland and T. T. Rogers. The parallel distributed processing approach to semantic cognition. *Nature Reviews Neuroscience*, 4:310–322, 2003.

[43] S. A. McDonald, R. H. S. Carpenter, and R. C. Shillcock. An anatomically constrained, stochastic model of eye movement control in reading. *Psychological Review*, 112:814–840, 2005.

[44] S. A. McDonald and R. C. Shillcock. Eye movements reveal the on-line computation of lexical probabilities during reading. *American Psychological Society*, 14:648–652, 2003.

[45] G. A. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *The Psychological Review*, 63:81–97, 1956.

[46] W. S. Murray and K. I. Forster. Serial mechanisms in lexical access: The rank hypothesis. *Psychological Review*, 111(3):721–756, 2004.

[47] C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala. Latent semantic indexing: A probabilistic analysis. *Journal of Computer and System Sciences*, 61:217–235, 2000.

[48] M. A. Pitt, W. Kim, D. J. Navarro, and J. I. Myung. Global model analysis by parameter space partitioning. *Psychological Review*, 113:57–83, 2006.

[49] J. Quesada. Creating your own LSA spaces. In T. K. Landauer, D. S. McNamara, S. Dennis, and W. Kintsch, editors, *Handbook of Latent Semantic Analysis*. Erlbaum, Mahwah, NJ, 2007.

[50] R. Radach and A. Kennedy. Theoretical perspectives on eye movements in reading: past controversies, current issues, and an agenda for future research. *European Journal of Cognitive Psychology*, 16(1/2):3–26, 2004.

[51] K. Rayner, J. Ashby, A. Pollatsek, and E. D. Reichle. The effects of frequency and predictability on eye fixations in reading: Implications for the E-Z reader model. *Journal of Experimental Psychology: Human Perception and Performance*, 30:720–732, 2004.

[52] E. D. Reichle and P. A. Laurent. Using reinforcement learning to understand the emergence of "intelligent" eye-movement behavior during reading. *Psychological Review*, 113:390–408, 2006.

[53] E. D. Reichle, A. Pollatsek, and K. Rayner. E-Z Reader: a cognitive-control, serial-attention model of eye-movement behavior during reading. *Cognitive Systems Research*, 7:4–22, 2006.

[54] R. G. Reilly and R. Radach. Some empirical tests of an interactive activation model of eye movement control in reading. *Cognitive Systems Research*, 7:34–55, 2006.

[55] R. Rosenfeld. Two decades of statistical language modeling: where do we go from here? *Proceedings of the IEEE*, 88(8):1270–1278, 2000.

[56] R. Rosenfeld, R. Agarwal, B. Byrne, R. Iyer, M. Liberman, E. Shriberg, J. Unverferth, D. Vergyri, and E. Vidal. Error analysis and disfluency modeling in the switchboard domain. In *Proceedings of the International Conference on Speech and Language Processing*, 1996.

[57] Christer Samuelsson. Handling sparse data by successive abstraction. In *Proceedings of COLING-96*, Kopenhagen, Denmark, 1996.

[58] H. E. H. Schilling, K. Rayner, and J. I. Chumbley. Comparing naming, lexical decision, and eye fixation times: Word frequency effects and individual differences. *Memory and cognition*, 26(6):1270–1281, 1998.

[59] M. Steyvers, R. M. Shiffrin, and D. L. Nelson. Word Association Spaces for predicting semantic similarity effects in episodic memory. In A. F. Healy, editor, *Experimental cognitive psychology and its applications*. American Psychological Association, 2004.

[60] M. Steyvers and J. Tenenbaum. The large scale structure of semantic networks: statistical analyses and a model of semantic growth. *Cognitive Science*, 29(1):41–78, 2005.

[61] A. Tarantola. *Inverse problem theory and methods for model parameter estimation*. Society for Industrial and Applied Mathematics, 2004. Also available for download at `http://www.ipgp.jussieu.fr/~tarantola/`.

[62] W. L. Taylor. Cloze procedure: A new tool for measuring readability. *Journalism Quarterly*, 30:415–433, 1953.

[63] A. M. Turing. Computing machinery and intelligence. *Mind*, 49:433–460, 1950.

[64] P. D. Turney. Mining the web for synonyms: PMI-IR versus LSA on TOEFL. In L. De Raedt and P. Flach, editors, *Proceedings of the Twelfth European Conference on Machine Learning (ECML-2001)*, pages 491–502, 2001.

[65] E. K. Warrington. The selective impairment of semantic memory. *Quarterly Journal of Experimental Psychology*, 27:635–657, 1975.

[66] S. J. White and S. P. Liversedge. Linguistic and nonlinguistic influences on the eyes' landing positions during reading. *The Quarterly Journal of Experimental Psychology*, 59(4):760–782, 2006.

[67] C. Whitney. How the brain encodes the order of letters in a printed word: the SERIOL model and selective literature review. *Psychonomic Bulletin & Review*, 8(2):221–243, 2001.

[68] C. Wu, M. Berry, S. Shivakumar, and J. McLarty. Neural networks for full-scale protein sequence classification: sequence encoding with Singular Value Decomposition. *Machine Learning*, 21:177–193, 1995.

[69] G. K. Zipf. *Human Behavior and the Principal of Least Effort*. Addison Wesley, Cambridge, 1949.