Knowledge Discovery From Natural Language Corpora - Mining For Relatedness In Textual Data

Schriftliche Hausarbeit
für die Masterprüfung der Fakultät für Philologie
an der Ruhr-Universität Bochum
(Gemeinsame Prüfungsordnung für das Bachelor/Master-Studium
im Rahmen des 2-Fach-Modells an der RUB vom 7.1.2002)

vorgelegt von

Björn Wilmsmann
Stefan-George-Str. 15a
46117 Oberhausen
bjoern@wilmsmann.de

abgegeben am 27. Mai 2008

Erstgutachter: Prof. Dr. Tibor Kiss
Zweitgutachter: Jun.-Prof. Dr. Stefanie Dipper

Table of Contents

As textual information available on the Web and other resources like company intranets keeps growing more complex, discovering knowledge from unstructured data steadily becomes more important for information retrieval systems in order to keep up with that growth. This is accomplished by using text data mining techniques that derive connections between pieces of information which are seemingly unrelated on a superficial level.

These techniques have to put into consideration both the structure of the various kinds of natural language resources and properties of natural language itself.

Building upon text normalisation and statistical language models, such techniques include statistical hypothesis testing, document indexing and clustering.

In terms of this paper an application for creating and using statistical language models has been developed under the name TextSieve. This application makes use of some of the text data mining techniques that will be described in this paper.

The availability of huge natural language corpora like newspaper corpora or the Google corpus (Brants / Franz, 2006) as well as the usage of the Web as a corpus (Grefenstette / Kilgariff, 2003), (Liu / Curran, 2006) also lends itself to linguistic research. Two more recent issues deal with the productivity of determinerless PPs and finding semantically similar words from a corpus and draw heavily upon the use of text mining methods to empirically underpin theoretical assumptions.

1. Introduction: Knowledge Discovery From Natural Language Corpora

Natural language resources play an important role nowadays, both in production use information systems and for linguistic research. Information retrieval systems need to process raw textual data in order to be capable of understanding the meaning of and relations between pieces of unstructured information stored as natural language. Such systems, like the ones employed by popular Internet search engines or their Intranet counterparts rely upon the quality of their underlying data models, which to a large extent consist of language models.

These models currently mainly are composed of statistical data (Chakrabarti, 2003), although there are recurrent attempts to integrate rule-based grammars as well in order to improve system performance, which usually is expressed in terms of recall and precision (Baeza-Yates / Ribeiro-Neto, 1999: pp. 73-97).

In addition to that, information extraction systems require even more intricate knowledge about the data they process than do information retrieval systems, as these systems are designed for not only retrieving the documents that are most likely to contain the information requested by a user, but to extract specific bits of information from such documents. This typically necessitates a detailed model of the data that is to be processed.

The same models that can be used for finding a document that contains requested information from a repository of textual data or extracting specific information, can also be applied for gaining knowledge about the structure of language itself. Statistical language models and algorithms building on these models provide useful information for example when it comes to determining the relatedness of words or possible collocations.

Though related, the aspect of gaining novel information from unstructured data is to be distinguished from information retrieval or information extraction and hence is defined by another term, that is text mining or text data mining. Text mining means data mining techniques applied to textual data.

While information extraction and text mining share the objective of extracting useful information from unstructured textual data, text mining just like its parent field, data mining has a more exploratory leaning. As for information extraction the categories sought for or slots to fill appropriate information into are known and well-defined. Text mining seeks to derive useful information from textual data without any predefined notion of what this information is to look like.

In more general terms, data mining means knowledge discovery by recognising common patterns and relations in huge amounts of data. Other areas data mining algorithms are frequently used in are business intelligence and image processing (Bishop, 2006).

According to Dey et al. (2007: p. 177) the aim of text mining is to provide the following facilities:

- distil the meaning of text in a concise form
- view accurate summaries before plunging into full documents
- navigate efficiently through large textbases
- perform natural language information retrieval

Thus, text mining can be seen as a requirement for advanced information retrieval systems. Without proper in-depth knowledge about the nature of the information that is supposed to be retrieved or extracted and the relations between specific pieces of information, retrieval of textual information that goes beyond mere keyword discovery is not feasible.

I shall give a brief description of the nature of the textual resources and the several kinds of text corpora and the formats they come in.

I will further discuss normalisation techniques that can be used to map related tokens to the same underlying concept. These include, amongst others, stemming and phonetic indexing algorithms.

By stemming we understand the algorithmic removal of inflexional and derivational morphemes from surface word forms. This process maps the different word forms a lexical item can expose to a single concept (or lemma), thus effectively reducing a surface language model to a much more compact and accurate underlying model. The most well-known algorithm of this kind probably is the Porter Stemming Algorithm or Porter Stemmer, named after its creator Martin Porter (2006).

Although very different from a linguistic point of view, phonetic indexing algorithms are also similar to stemming algorithms, since both describe surjective functions that map a set of surface entities to a set of underlying entities. In contrast to stemming algorithms, phonetic indexing does not subsume word forms that belong to the same underlying concept, but relates tokens in terms of how similar their phonetic structure is. That is, depending on the actual algorithm, tokens that – given some broad assumptions about the accent used – sound the same or even only similar to a certain degree will be mapped to the same type.

In order to lay the roots for creating statistical language models, we shall briefly consider basic concepts like token and type.

Statistical language models can be deduced from natural language corpora by applying basic concepts of statistics to natural language data. As for natural language data specific properties like the Zipf distribution of type frequencies and the sparse data problem have to be considered. Smoothing methods that try to address this problem will be presented in order to improve the models derived from raw textual data.

Various statistical measures of association will be addressed in order to outline how statistical language models can be applied for deriving knowledge about linguistic issues like the relatedness of words as given by their respective probability to co-occur with each other.

Furthermore, I shall elaborate on how statistical data gathered from a document collection can be used for document indexing and clustering in a vector space model.

Grounded on previous work on Text-NSP (see Banerjee / Pedersen (2003) for a detailed description of this package), a text statistics module for the Perl programming language, I have developed a software framework for text mining called TextSieve. This software and its design will be outlined and explained. I shall outline methods for efficiently counting n-grams of variable length and creating a language model from this data.

Finally, a few practical applications of the techniques shown in this paper will be presented:

First of all, we shall deal with a current issue in linguistic research, that is the question if determinerless PPs (Baldwin et al., 2006) are regular and productive linguistic structures or if they are to be considered idiomatic (Kiss, 2006 / 2007).

Finally, we shall turn to lexicography and show how statistical hypothesis tests can be used in this particular area for finding semantically similar words.

2. Natural Language Resources

Natural language resources come in many different flavours. Each of them has a different purpose. Some were created for allowing specific linguistics research on a certain topic or as general purpose linguistic tools. The bulk of the textual resources available today is not even made for being processed automatically, but simply for being consumed by human readers. The format they are delivered in frequently is as different as their respective purposes. Some of them come in plain text, while some are provided as complex XML feature structures. The vast majority of textual data on the Internet is, amongst other file formats like PDF, supplied in the form of hyperlinked (X)HTML documents.

Each of these ways of delivering textual information has its advantages and sometimes disadvantages over one another, one being more amenable to specific algorithms, while others being less so. These different methods of delivering textual information that can be processed by text mining systems give rise to equally distinctive aspects that can be exploited by using such resources. This in turn leads to the various resources being more or less appropriate for different aspects of text mining, linguistic research and information retrieval. In the following section we shall elaborate on the currently most common formats for delivering textual data and in which ways each of them can be used for applications in these areas.

2.1 Plain Text Corpora

The most obvious and structurally simplest way of representing textual information certainly is plain text. As plain text does not necessitate any additional preprocessing by the software handling it, it is the most readily processable textual resource, which can be dealt with by most of the programming languages in use today just by using the respective core language features.

However, this certainly has some drawbacks as well. First of all, as it hardly exposes any structure apart from sentences, paragraphs and maybe page markers or various other markers used in languages other than those from the Indo-European language family, it cannot provide as much information as feature-rich or hyperlinked formats. Secondly, processing plain text might even be more difficult than processing more complex structures, since plain text usually does not abide by a well defined format and therefore does not provide any structure that can be used as rules by processors.

Another issue that comes into play when dealing with plain text data (and actually any format that builds upon plain text like XML or HTML) is character encoding. Most languages of the world use various numbers of special characters that are unique to that specific language or a small set of languages. These special characters used to be encoded in special character sets for each language. This can still impose serious problems for systems that have to deal with texts from different languages or even multi-lingual texts, because texts need to be converted from one encoding into another, which, depending on the languages that have to be covered, can result in conversion errors that render the data useless. Moreover, many languages like Chinese or Japanese contain so many different symbols that they simply cannot be encoded in standard 7-bit ASCII anymore. In order to amend this, Unicode has been introduced in 1991 and further improved from then on. Unicode offers a by far wider range for representing characters than does ASCII. Therefore, it can be used as a universal encoding for every human language.

2.2 Corpora With Rich Feature Structures

In addition to plain text corpora, there are also quite a few corpora that supply additional information alongside with the raw text. The scope of this information can vary quite a bit. There are spoken language corpora which might simply provide information about how a specific word is pronounced as well as complex lexical resources that attach a great deal of additional morphological, syntactic and semantic information to a given word.

Typically these corpora serve as resources for linguistic information that exceeds the mere statistical information hidden in plain textual data. This can include detailed morphological specifications like part of speech, word formation rules and word origins, syntactic structures like for instance complement and adjunct structures and last not least semantic information about a word like hypernyms, hyponyms and related concepts.

The information is usually encoded in XML formats that build complex tree structures by referencing other nodes or possibly referring to other data sources[1].

This kind of corpus is created manually for the purpose of allowing machines to process huge amounts of language data alongside with linguistic knowledge that otherwise is unavailable to machines. By making use of such corpora a language processing system can build upon intricate human knowledge about linguistic structures in order to, for example, parse sentences syntactically, tag words with their respective part of speech or cluster words according to the environment they occur in.

Although these corpora have been proven to be expedient for several applications in natural language processing they also display some drawbacks that detract from their general usefulness:

First of all, as of now, this type of corpus has to be hand-crafted, which involves a lot of cumbersome work being done by human researchers. For this very reason, only a limited amount of the text available can and will be enriched with feature structures until there is a way for having machines do all or most of the work implied by this process.

Another possible problem is the variation in formats for encoding rich corpora. Fortunately, most researchers in this area have opted for the use of XML as an open standard (in contrast to proprietary formats) for defining their respective file formats. However, as the format definitions vary from corpus to corpus, each has to be parsed using a slightly different implementation, which can impose a considerable development effort.

2.3 Hyperlinked Documents

By hyperlinked documents we understand documents that besides the textual information and alongside with more detailed information about visual representation and document-internal semantic structure also display links to other resources as a

1 For example, see WordNet: http://wordnet.princeton.edu/ and ConceptNet: http://web.media.mit.edu/~hugo/conceptnet/ .

way of organising complex relations between distinct pieces of information. The nowadays most commonly used incarnation of this concept is (X)HTML[2].

This particular way of organising information on the Web has brought about completely new approaches for filtering relevant information, the most well-known one probably being the PageRank algorithm used by the Google search engine (Brin / Page, 1998).

The additional information provided by links between otherwise isolated chunks of information can be used to add another layer of analysis on top of the methods that can be used for plain textual data.

PageRank for instance ranks pages from a result set for a search query according to the number of incoming and outgoing links (Chakrabarti, 2003: pp. 209-212).

3. Statistical Language Models

In this section I shall address various issues that are relevant to the creation of statistical language models that can be used for finding relations between words on the grounds of their statistical properties.

For this we first have to define the entities we want to examine statistically. In the case of computational models of language these typically are tokens, types and building upon that, n-grams and collocations on a more abstract level.

Secondly, we have to know about the margins within which a particular event will be subsumed under a specific entity. As for language this means dealing with phonetic and morphological variation, that is to say a word can be spelled slightly different (the most common case simply being spelling errors) or expose a variety of different word forms while still belonging to the same lexeme. This process of assigning different surface forms to a single entity is called text normalisation. A specific case of this kind of normalisation is stopword elimination, which means discarding all the words from corpus that are considered to bear no actual meaning in the corpus language.

2 See http://www.w3.org/html/ and http://www.w3.org/MarkUp/ respectively.

We shall see how a language model actually is created and how we can use measures of association like X^2 (chi-square), mutual information or the t-test in order to gain knowledge about the inherent properties of word frequency distributions.

3.1 Tokens And Types, N-Grams And Collocations

When we talk about statistical language models and natural language corpora the most basic and readily available concepts are those of *type* and *token*.

By a single token we understand an occurrence of a particular word in a corpus. Conversely the set of types of a corpus is built from the distinct words that occur in this corpus and hence can be thought of as the lexicon of that corpus – in fact the term *distinct* is essential to understand the conceptual difference between token and type (Manning / Schütze, 1999: p. 22):

> "In general in this way one can talk *tokens*, individual occurrences of something, and *types*, the different things present."

The *n-gram* concept builds upon this in that n-grams consist of several consecutive single word tokens (or types for that matter). The idea of single word (or unigram) types and tokens can be extended to types and tokens of size *n* yielding a string consisting of *n* (distinct) words.

Another way to understand n-grams is to interpret them as Markov chains. A Markov chain is a sequence of random variables for which the following properties hold (Manning / Schütze, 1999: p. 318):

Limited Horizon:

$$P(X_{t+1}=s_k|X_{1,}\ldots,X_t)=P(X_{t+1}=s_k|X_t)$$

Time invariant (stationary):

$$P(X_{t+1}=s_k|X_{1,}\ldots,X_t)=P(X_2=s_k|X_1)$$

This means that an n-gram is an approximation of the probability of a word given the *n - 1* words preceding it (Jurafsky / Martin, 2000: p. 197).

An n-gram with *n = 2* is called a first order Markov model as it has a horizon of one previous token, an n-gram with *n = 3* has a horizon of two previous tokens. In general an n-gram is an *(n - 1)*th order Markov model (Jurafsky / Martin, 2000: p. 198).

While n-grams can assume any size, only smaller-sized n-grams are used in practice in order to avoid combinatorial explosion and keep the model computationally tractable. The n-grams used in practice usually are 2-grams (or bigrams) and 3-grams (or trigrams), while 4-, 5-grams or even larger ones tend to be rarely used due to the sparse data problem (Manning / Schütze, 1999: p. 194):

> "So we quickly see that producing a five-gram model [...] may well not be practical, even if we have what we think is a very large corpus. For this reason, n-gram systems currently use bigrams or trigrams (and often make do with a smaller vocabulary)."

Collocations, a term for corpus linguistics, is a more abstract concept referring to any sequence of words whose components occur together more frequently than would be expected when assuming statistical independence.

According to Choueka (1988), collocations are defined as follows:

> "A sequence of two or more consecutive words that has characteristics of a syntactic and semantic unit, and whose exact and unambiguous meaning or connotation cannot be derived directly from the meaning or connotation of its components."

Formally, collocations can be defined using the following criteria (Manning / Schütze, 1999: p. 184):

1. Non-compositionality

2. Non-substitutability

3. Non-modifiability

The first criterion signifies that the meaning of a collocation cannot be compositionally derived by the meanings of its components alone, which seems to violate the compositionality principle, which in turn is a strong argument for treating them as atomic entities similar to words.

The second feature results from the first criterion: A component of a collocation cannot be substituted for another without breaking up the collocation.

Finally, the last criterion suggests that collocations cannot be modified freely be it by adding lexical items or by grammatical transformations.

All of these criteria can be considered strong arguments for categorising collocations as idiomatic expressions.

Collocations can range from two words directly adjacent to each other to structures composed of more entities within a window of a specified size (Banarjee / Pedersen, 2003: p. 373-376). This means that the components of a collocation do not have to be directly adjacent to each other like in 'The doctor performs an operation.' Here, 'perform' and 'operation' constitute a collocation although there is another word in-between them (Manning / Schütze, 1999: p. 184):

> “But in most linguistically oriented research, a phrase can be a collocation even if it is not consecutive.”

As we shall see later, the latter aspect of collocations is rather important and hence is reflected in the text mining software developed for this paper by allowing an n-gram to appear in a window of an arbitrary size m (instead of size n).

Another notion that can lead to confusion is a concept called *term* that sometimes is used interchangeably with *word* or *type*. In information retrieval *term* simply stands for a keyword that occurs within a document or a search query, regardless of its count, and hence is equivalent to our notion of *type*.

3.2 Text Normalisation

Text normalisation means converting the surface representation of textual data into actual entities that can be used consistently during further processing steps. This is done by mapping the surface forms of words to the lexemes they belong to.

Depending on the application this can range from stemming to error correction methods like phonetic indexing.

3.2.1 Stemming

Stemming means a surjective mapping of grammatical word forms to the actual abstract word (lemma) behind them. This typically is done by morphologically analysing the input words and sequentially applying numerous rules in order to reduce a given word to its stem. Depending on the irregularities exposed by the language at hand and the actual algorithm this process tends to be error-prone.

Stemming is usually done in order to increase the recall of information retrieval systems which only have access to a limited amount of data. Stemming allows a user to enter a query like 'reading' and retrieve all the documents containing the term 'read' or morphological variations thereof. Depending on the setting, especially in terms of the Web, where information is readily available in abundance, stemming might however contribute to decreasing system performance (Chakrabarti, 2003: p. 49):

> "When in doubt, it is better not to stem, especially for Web searches, where aliases and abbreviations abound. [...] Owing to the variety of abbreviations and names coined in the technical and commercial sectors, polysemy is rampant on the Web."

According to Baeza-Yates / Ribeiro-Neto (1999: p. 168) there are four basic types of stemming strategies: Table lookup, affix removal, stemming based on successor variety and stemming based on n-grams.

Table lookup means a brute force method that simply tries to map each word form to a stem from a predefined table. This approach appears to be rather impractical for any real-world application as it is "dependent on data on stems for the whole language" (Baeza-Yates / Ribeiro-Neto, 1999: p. 168), which would require both considerable storage space and a seemingly endless amount of work for creating the table in the first place and then keeping it up-to-date.

Stemming by affix removal can be understood as a set of heuristic affixation rules that is applied to a word in order to conflate it to its stem

Successor variety stemming by contrast makes use of the morphological features of words (Baeza-Yates / Ribeiro-Neto, 1999: p. 168):

> "Successor variety stemming is based on the determination of morpheme boundaries, uses knowledge from structural linguistics, and is more complex than affix removal stemming."

Finally, n-gram-based stemming is based upon finding bigrams and trigrams of characters that frequently co-occur and hence at a certain probability make up a suffix or prefix. This approach disregards underlying linguistic structures, as it only takes the surface form of the input stream into consideration. Nevertheless, this approach appears to be quite successful and a similar idea therefore is used in the TnT part-of-speech tagger as a means of heuristically tagging words unknown to the system (Brants, 2000: pp. 25-26).

Probably the most well-known stemming algorithm is the one implemented by the affix removal stemmer known as the Porter Stemmer, which has come to be used in a vast variety of applications, because "despite being simpler" it "yields results comparable to those of the more sophisticated algorithms" (Baeza-Yates / Ribeiro-Neto, 1999: p. 169).

Building upon his works on stemming algorithms Martin Porter, the eponymous developer of the Porter Stemming Algorithm, introduced the Snowball stemming framework due to the evident "lack of readily available stemming algorithms for languages other than English" (Porter, 2001).

3.2.2 Phonetic Indexing

Phonetic indexing basically is a surjective mapping from terms that would sound the same or at least similar within predefined limits when pronounced in a given phonetic system to the underlying entities these terms refer to. Phonetic indexing is especially useful in cases where one wants to gloss over spelling errors that otherwise would deteriorate the language model by falsely introducing a new term for every misspelled word.

3.2.3 Removal Of Stopwords

Another aspect of text normalisation that usually is conducted before any further processing takes place is stopword filtering, which means filtering those words from a document that convey little to no specific semantics in the document language and hence can be neglected during further processing, thereby reducing both processing and storage costs. Commonly, these include pronouns, prepositions and the different word forms of functional verbs like 'to be' or 'to have'.

Removing stopwords is an important preprocessing step in information retrieval systems. According to Zipf's Law (see section 3.3.2), which states that in a natural language model the frequency of the *i*-th most frequent word roughly is 1 / *i* times that of the most frequent word, very few words account for the majority of natural language texts (Baeza-Yates / Ribeiro-Neto, 1999: p. 146).

As words that occur in the majority of all documents in a corpus are of little use for effectively discriminating documents that fit a given query and those that do not, the words occurring most frequently in a corpus may be removed before the actual document indexing in information retrieval systems takes place. Another argument for eliminating stopwords before building a document index is discussed by Baeza-Yates / Ribeiro-Neto (1999: p. 167):

> “Elimination of stopwords has an additional important benefit. It reduces the size of the indexing structure considerably.”

However, eliminating stopwords beforehand might have a serious impact on the recall of an information retrieval system, depending on the usage patterns of the users of that system. If, for example, a user enters a query like 'to be or not to be' stopword elimination might reduce this phrase query to zero length as it only contains words which usually can be considered stopwords both in terms of their usual relative frequency in document collections and the semantic content they convey. For this very reason some modern search engines do without stopword elimination right away (Baeza-Yates / Ribeiro-Neto, 1999: p. 188):

> “A good example of this trend is the fact that some Web search engines index all the words in the text regardless of their syntactic nature or their role in the text.”

As with stemming there are several ways to create stopword lists for subsequently carrying out stopword removal.

A simple approach is merely listing stopwords in a manually edited lexicon and removing all occurrences in the documents or corpora to be processed. Though this method, like table lookup for stemming, might seem rather brute-force at the first glimpse, too, for stopwords such an approach is actually quite sensible, because a language only has a limited number of possible stopwords as opposed to a potentially infinite number of morphemes.

The disadvantage of filtering stopwords by a fixed lexicon though is that it glosses over the variation of potential stopwords depending on the corpus in question. Certain terms, for instance, might serve as stopwords for one corpus while they might be inappropriate for another.

This is why another idea that lends itself for stopword list creation and makes use of the aforementioned Zipf distribution characteristics of natural language might prove useful depending on the setting. This idea is based upon a corpus-specific set of stopwords (Baeza-Yates / Ribeiro-Neto, 1999: p. 167):

> “In fact, a word which occurs in 80% of the documents in the collection is useless for retrieval. Such words are frequently referred to as stopwords and are normally filtered out as potential index terms.”

3.3 Building a Statistical Language Model

The next step in deriving knowledge from a corpus of text, after text normalisation has been done, consists of building a language model for that particular chunk of text.

Basically, as for statistical language models this first of all means counting the tokens and types which occur in the corpus at hand. Depending on the objective of the subsequent examination this can be extended to not only counting unigram tokens and types but n-gram ones, too.

After having counted the occurrences of the entities in the corpus, a first probabilistic model can be created from this data by calculating maximum likelihood estimates for each type.

Due to the nature of natural language data, this rather simple approach though is not without problems. Sparse data and a large number of rare events (LNRE) leads to specific problems that require treatment in order to generated useful information (Manning / Schütze, 1999: p. 198):

> “But the MLE is in general unsuitable for statistical inference in NLP. The problem is the sparseness of our data (even if we are using a large corpus). While a few words are common, the vast majority of words are very uncommon [...].”

One such treatment is called smoothing and essentially means finding appropriate predictions for outliers, artefacts and events that have not been observed in the data set so far but may still occur given a sufficiently large data set. This is done by decreasing the probabilities of seen events and assigning the 'free' probability mass to unseen events (Manning / Schütze, 1999: p. 199):

> “All such methods effectively work by somewhat decreasing the probability of previously seen events, so that there is a little bit of probability mass left over for previously unseen events. Thus these methods are frequently referred to as discounting methods. The process of discounting is often referred to as smoothing, presumably because a distribution without zeroes is smoother than one with zeroes.”

Another aspect when it comes to modelling language according to its statistical features is measuring the entropy of a model. Entropy is a concept from information theory.

To cut a long story short, entropy is a measure for the informativeness of particular model, that is to say, entropy tells us how many bits are needed on average to unequivocally select a specific entity from the model. Entropy is defined as

$$H(x) = -\Sigma_{x \in X} p(x) \log_2 p(x)$$

The *log*$_2$ in the equation results in entropy being measured in bits (Jurafsky / Martin, 2000: p. 224).

A high level of entropy means that the model at hand either uses the wrong features to characterise its events or that the events in the distribution of the events are rather balanced, which usually is not the case with natural language (Manning / Schütze, 1999: p. 76):

> “The essential point here is that if a model captures more of the structure of a language, then the entropy of the model should be lower. In other words, we can use entropy as a measure of the quality of our models.”

As for n-gram models, entropy is used as a metric to determine “how predictive a given N-gram grammar is about what the next word would be” (Jurafsky / Martin, 2000: p. 224).

Informally, entropy can be subsumed as the amount of surprise an observer feels when perceiving a random event. If the model entropy is low, an observer is more likely to predict the outcome of an event than if the model entropy, that is the amount of surprise contained within the model, is rather high (Manning / Schütze, 1999: p. 73):

> “Alternately, we can think of entropy as a matter of how surprised we will be.”

The following sections will attempt to consider the most important aspects of creating statistical language models and describe each of them in more detail.

3.3.1 Maximum Likelihood Estimates

Statistical estimators provide probabilistic estimates for events in a data set. That is, based upon the data collected they predict the probability of a particular outcome of a new event. The most common and easily accessible statistical estimator is defined by maximum likelihood estimates (MLE).

Maximum likelihood estimates are given by the relative frequencies of events, that is the frequency of each event divided by the total number of events.

The term *maximum likelihood estimates* is derived from the fact that this particular estimator assigns the maximum likelihood to the data set at hand. For a natural language corpus it estimates the probabilities such that the corpus is the most likely data set to have brought about these probabilities (Jurafsky / Martin, 2000: p. 200):

> “[...] the use of relative frequencies as a way to estimate probabilities is one example of the technique known as Maximum Likelihood Estimation (MLE), because the resulting parameter set is one in which the likelihood of the training set T given the model M (i.e., P(T|M)) is maximized.”

However simple and therefore likeable this approach may seem, it poses a severe problem when dealing with natural language data. This problem is commonly known as the sparse data problem or “data sparseness” (Manning / Schütze, 1999: p. 198) and takes into account that in natural language there are a few very common words while the vast majority of words occur very rarely. This problem and its consequences are described in more detail in the next section.

3.3.2 Zipf's Law And Sparse Data

Events in natural language are not distributed randomly, but there are a few common and many rare ones (Manning / Schütze, 1999: p. 24):

> “There are a few very common words, a middling number of medium frequency words, and many low frequency words.”

This implies that in contrast to normally-distributed samples, which reach the maximum value of available elements rather quickly, natural language typically requires much larger sample sizes – in fact the size converges to infinite – in order to cover every possible entity in the vocabulary, which is dubbed the LNRE (Large Number Of Rare Events) zone by Baayen (2001: p. 51):

“The LNRE zone can be described as the range of sample sizes for which it is clear from the shape of the spectral curves that we have only just begun to sample the types available in the population.”

This behaviour is subsumed by Zipf's Law, which can be stated as follows (Manning / Schütze, 1999: p. 24):

$$f \propto \frac{1}{r}$$

The meaning behind this is that the frequency of each word is proportional to the multiplicative inverse of its frequency-wise rank in the respective corpus multiplied with the frequency of the most common word. For example, the third most common word according to this law is supposed to occur $\frac{1}{3}$ times as frequent as the most common word.

Zipf's Law is an instance of the more general Zipf-Mandelbrot Law, which introduces further parameters (Manning / Schütze, 1999: p. 25):

$$f = P(r+\rho)^{-B}$$

P, *B* and ρ are properties of the respective corpus, which “collectively measure the richness of the text's use of words” (Manning / Schütze, 1999: p. 25). For values of $B = 1$ and $\rho = 0$ the results correspond to those posited by Zipf's Law. Mandelbrot's generalisation has advantages over Zipf's Law in that it more closely predicts the frequencies for very frequent and very rare events. The following diagrams, taken from Manning / Schütze (1999: pp. 26-27), illustrate the difference between a model as posited by Zipf's Law and a model given by the Mandelbrot's generalisation. Both graphs use logarithmic scales:

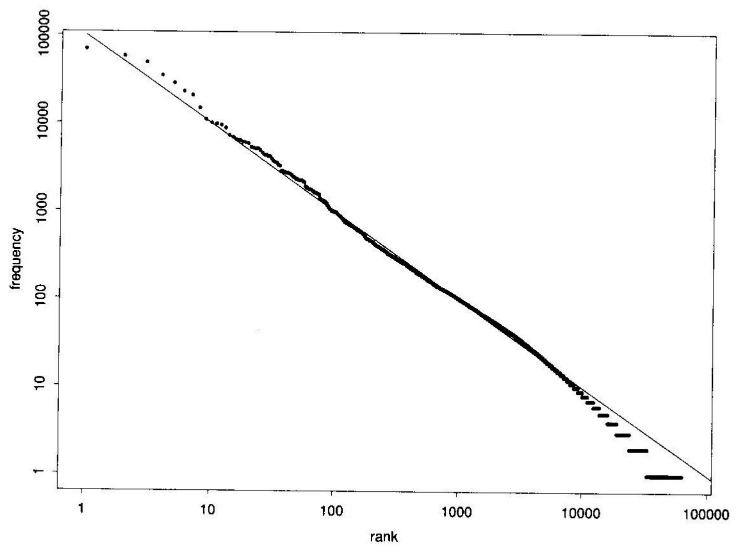


Figure 1: The dotted graph shows word frequencies for the Brown corpus. The line gives the frequencies for this corpus as predicted by Zipf's Law

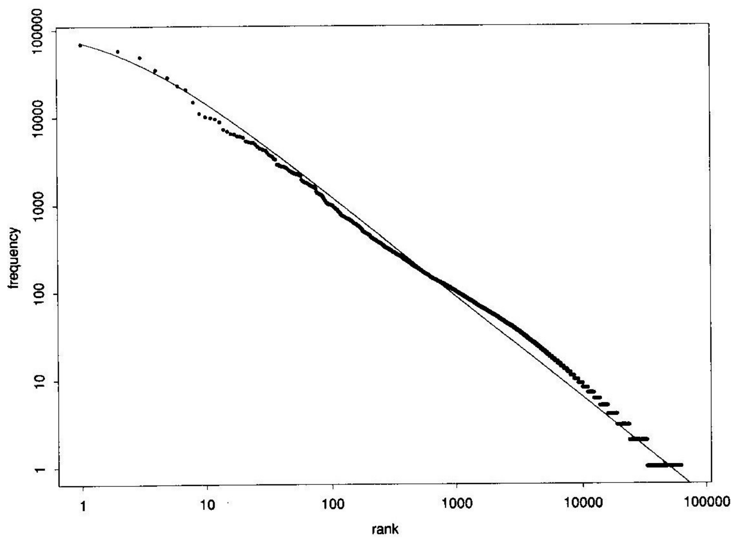


Figure 2: The dotted graph again shows word frequencies for the Brown corpus. The line in this case gives the frequencies for this corpus as predicted by Mandelbrot's formula.

Due to the "long tail" (Anderson, 2004) of Zipf distributions, which like the similar Pareto distributions belong to the family of power law distributions (Manning / Schütze, 1999: p. 28), Zipf's Law also makes predictions about the ratio of text and vocabulary size. As there is only a limited number of frequent words, the larger a text gets the less frequent are those words which have not been encountered till that particular point. This also entails that the larger the text the lower the chance of still finding new words to add to the vocabulary.

The diagrams below, taken from Baeza-Yates / Ribeiro-Neto (1999: p. 147), show the relation between word frequencies and text to vocabulary size ratio resulting according to Zipf's Law for a given textual sample.

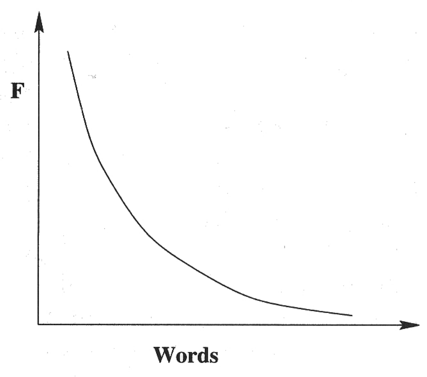


Figure 3: Zipf distribution of word frequencies in natural language corpora

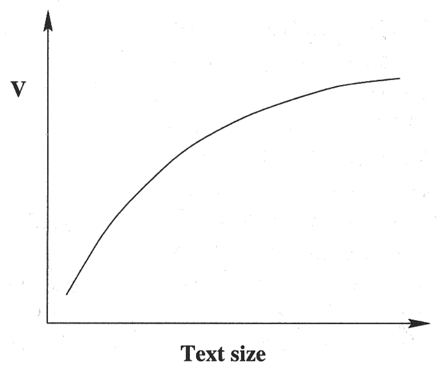


Figure 4: Text to vocabulary size ratio

Distributions displaying a LNRE are known for exhibiting the so-called sparse data problem. Given the long tail of such distributions we can never gobble up sufficient data to be sure that we have seen every possible event.

For example, given a context of a 'under' as the first word of collocation one cannot state for sure that all words that could potentially follow 'under' have been present in the data at hand. Such events that have not occurred in a data set, yet nonetheless are potential outcomes, are called unseen events.

Maximum likelihood estimates do not allow us to make appropriate predictions about this kind of events because they by definition only represent actually seen events.

Which is more, MLE also makes false statements about seen events, since underestimating the frequencies of unseen events at the same time brings about overestimated frequency values for the seen events. Apart from that zero probabilities are unfavourable when dealing with complex phenomena like n-grams whose probability in a given context is calculated by multiplying the probabilities of several singular events. If one of those events happens to be unseen in the corpus at hand, the overall probability of the n-gram will be zero when using MLE (Gale / Sampson, 1995: p. 218):

> “These [statistical calculations] often involve multiplying estimated probabilities for many simple phenomena to reach overall figures for the probability of interesting complex phenomena; zeros propagate through such calculations, so that the phenomena of interest are often assigned zero probability even when most of their elementary components are very common and the true probability of the complex phenomenon is reasonably high.”

There are however statistical estimators which can be used for taking unseen events into consideration. These estimators will be covered by the next section.

3.3.3 Smoothing

As we have seen in the previous sections, natural language exhibits properties which require re-considering the statistical estimator to use when creating statistical models for natural language corpora.

The methods used for properly estimating data sets that display a large number of rare events are referred to as discounting or smoothing methods. This is because these measures discount from the probability mass which has been assigned to the seen events and redistribute the amount of discounted probability mass among the unseen events.

On account of this process the distribution as well as its visual representation as a graph is smoothed meaning that there are no zero probabilities anymore.

The first estimator of this kind I would like to introduce is based upon using Laplace's Law. According to this law the probability of an n-gram is given as follows (Manning / Schütze, 1999: p. 202):

$$P_{Lap}(w_1 \ldots w_n) = \frac{C(w_1 \ldots w_n) + 1}{N + B}$$

N signifies the number of n-gram tokens whereas B means the number of n-gram types. This estimator has the effect of vastly reducing the probabilities for seen events while assigning a tiny amount of probability mass to the unseen events. Frequently, this method is also called *adding one* or *add one* as it adds a frequency count of 1 to each event (Jurafsky / Martin, 2000: p. 207).

While this estimator is an improvement over MLE it exhibits two drawbacks as well:

1. It assumes a uniform prior for the unseen events which results every event of this kind having the same probability (Manning / Schütze, 1999: p. 202):

   > “[...] this is actually the Bayesian estimator that one derives if one assumes a uniform prior on events (i.e., that every n-gram was equally likely)”

2. It is biased towards assigning a much larger probability mass to the unseen events than is appropriate (Manning / Schütze, 1999: p. 202):

   > “For sparse sets of data over large vocabularies, such as n-grams, Laplace's law actually gives far too much of the probability space to unseen events.”

Another statistical estimator used to improve the results given by MLE builds upon Lidstone's Law of succession (Manning / Schütze, 1999):

$$P_{Lap}(w_1 \ldots w_n) = \frac{C(w_1 \ldots w_n) + \lambda}{N + B\lambda}$$

It differs from adding one in that we do not simply add a value of 1, but some value $\lambda < 1$. Most commonly a value of $\frac{1}{2}$ is used for λ. This particular measure sometimes is also called expected likelihood estimation (ELE) (Manning / Schütze, 1999: p. 204).

The final smoothing method I am going to talk about is Good-Turing estimation. According to Jurafsky / Martin (2000: p. 214) the algorithm was first described by Good (1953), however Turing gets credited with the actual idea.

Though initially conceived for use with binomial distributions and hence admittedly being potentially “vitiated by breakdowns of the binomial assumption in natural-language data” (Gale / Sampson, 1995: p. 234), Manning / Schütze (1999: p. 212) state that the Good-Turing estimation is equally useful for distributions other than binomial ones such as the distributions typically shown by n-gram data.

Good-Turing estimation differs from the methods introduced above in that it does not operate on frequencies of events but on frequencies of frequencies (or count-counts (Manning / Schütze, 1999: p. 213)). Good-Turing estimation in contrast to adding-one or ELE does not interfere with probabilities directly, but operates on the frequencies of the events in the sample.

Thus, the first step to be done before applying Good-Turing estimation is assigning the events to equivalence classes according to their respective frequencies. This means that in our case words or n-grams are not treated as single entities anymore but merely based upon their frequency.

Apart from being a necessary preliminary step for Good-Turing estimation such treatment is useful for efficient calculation of probabilities as well, since given equivalence classes based upon frequency we can calculate the probability for all events with the same frequency in one step. For example, all words having the frequency 32 in a corpus would be assigned to frequency class 32 and only one computation would be necessary to calculate the probability for all of the words in that bin.

After having calculated frequencies of frequencies the actual smoothing process can be carried out. In the following lines a description of the Simple Good-Turing estimator by Gale / Sampson (1995) will be given.

The probability estimated by Simple-Good-Turing is given as follows

$$P_{goodturing} = \frac{r^*}{N}$$

where *r** can be thought of as an adjusted frequency or “the estimated number of cases of a species actually observed *r* times in that sample which *would* have been observed, if the sample were perfectly representative of the population” (Gale / Sampson, 1995: p. 219). *r** is given by

$$r^* = (r+1)\frac{E(N_{r+1})}{E(N_r)}$$

$E(N_{r+1})$ and $E(N_r)$ respectively are the estimated frequencies for the frequencies N_{r+1} and N_r. The probability of all unseen events is given by the following corollary:

$$P_0 = \frac{E(N_1)}{N}$$

For each frequency *r* there are *r'* and *r''* which represent the nearest lower and nearest higher sample frequencies.

Based upon this a new variable Z_r is defined as follows:

$$Z_r = 2\mathrm{n}_r / (r'' - r')$$

The resulting function is smoothed by approximating it with the log-linear function *S(r)* that fits the Z_r points best, as illustrated by the following figures taken from Gale / Sampson (1995: p. 222):

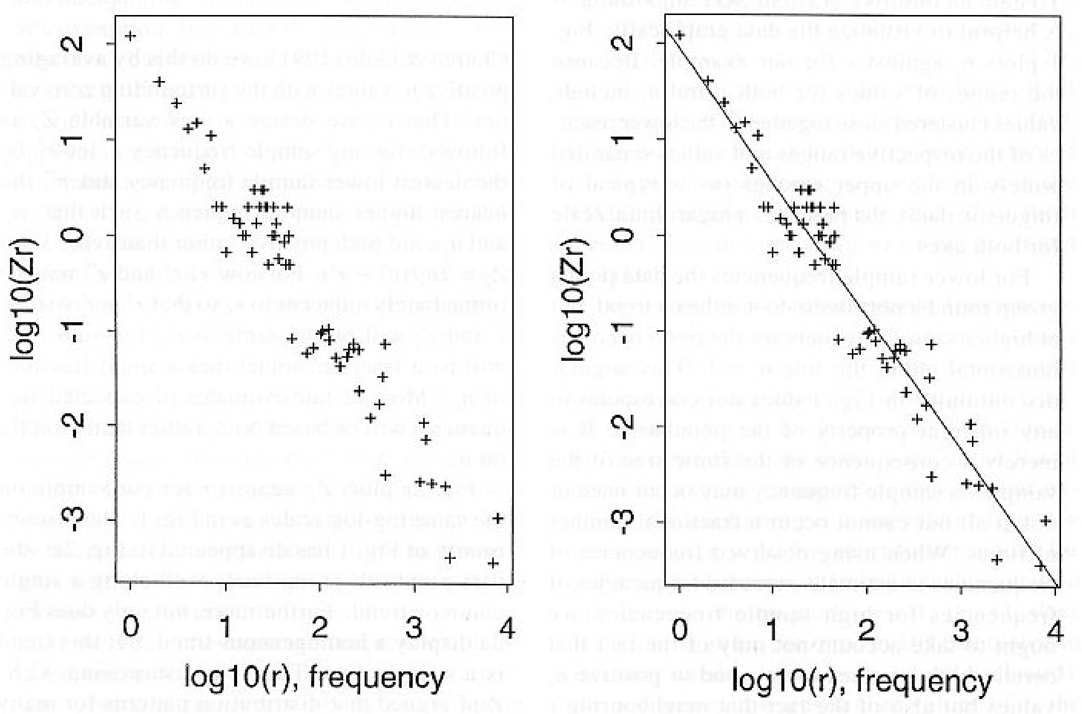


Figure 5: Z_r corresponding *S(r)*

The actual frequency n_r is used instead of *S(r)* as long as *S(r)* yields significantly different results. If the following inequality is true

$$|(x-y)|>1.96*\sqrt{(r+1)^2\frac{n_{r+1}}{n_r^2}(1+\frac{n_{r+1}}{n_r})}$$

*r** is calculated using n_r:

$$y=(r+1)\frac{n_{r+1}}{n_r}$$

Once the inequality does not hold anymore, all subsequent *r* are estimated using *S(r)*:

$$y=(r+1)\frac{S(r+1)}{S(r)}$$

After having properly estimated the probabilities for word and n-gram types in a corpus, this data can be used to derive information about the relatedness of word types, a subject that will be covered in the next section.

3.4 Statistical Hypothesis Tests

Apart from merely making statements about the frequency of particular words in a subset of a language or the overall probability of a single word to occur in a given textual excerpt written in that language, statistical language models can be used for deriving even more useful information.

Statistical models allow us to make statements about lexical relationships between words. There are several types of such relationships, two of which are positive association and difference (Pedersen / Kayaalp / Bruce, 1996: p. 455). Positive association means the probability of a value for one random variable to occur together with a value for another random variable while difference signifies the degree of overlap (or lack thereof) between the contexts for values of a given random variable.

In our case this means testing the association between the words that make up an n-gram and testing the difference between the contexts of words in terms of the n-grams they occur in. Knowing about the association between words allows us to make predictions about two or more words forming cohesive lexical entities or collocations.

If words co-occur more frequently than is to be expected by chance they possibly form collocations: Linguistic entities composed of several words which are quantifiably related (Jurafsky / Martin, 2000: p. 637):

> "In general the term collocation refers to a quantifiable position-specific relationship between two lexical items."

This includes binomials like *cease and desist* or *to have and hold*, compounds like *car park* and proper names like *General Electric*.

The difference between the contexts of words can prove to be useful if one wants to find out about words which are similar in that they occur in similar environments, which for example comes in handy when clustering words regarding their lexical and semantic similarity (see section 4.3 for more information about clustering algorithms).

Both properties, difference and association, are assessed using so called significance tests. A typical significance test is carried out as follows (Pedersen / Kayaalp / Bruce, 1996: p. 456):

1. Select the appropriate sampling model,

2. hypothesize a population model for the data sample,

3. select a goodness of fit statistic to use in testing the fit of the model to the data sample, and

4. assess the statistical significance of the model: determine the probability that the data sample came from a population described by the model.

Now, what does this mean for our particular problem of making predictions about the co-occurrence of words?

First, a sampling model for the words under scrutiny is built. Such a model basically consists of the counts of each of the words alone, the counts of the n-grams, the marginal and the overall counts. This information can be represented in a so-called contingency table. Take the following example from Pedersen (1996: p. 189):

	+industry	-industry	totals
+oil	$n_{11} = 17$	$n_{12} = 229$	$n_{1x} = 246$
-oil	$n_{21} = 935$	$n_{22} = 1381647$	$n_{2x} = 1382582$
totals	$n_{x1} = 952$	$n_{x2} = 1381876$	$n_{xx} = 1382828$

This matrix basically contains the following information: In the given sample, the bigram *oil industry* occurs 17 times. The unigram type *oil* occurs 229 times without *industry* as consecutive unigram token, amounting to a total number of 246 occurrences for *oil*. Likewise, the unigram type *industry* occurs 935 times without *oil* preceding it, which adds up to a total of 952 tokens of the type i*ndustry*. Furthermore, there are 1381647 bigrams in the given sample that neither contain *oil* nor *industry* as tokens resulting in 1381876 bigrams that do not contain *industry* as their respective second token. Finally, there are 1382582 bigrams which do not display *oil* as their first token and an overall number of 1382828 bigrams.

After having sampled that data, the next step is formulating a hypothesis for a population model to be tested for significance. By hypothesis we understand some assumption concerning our model that can be proved or disproved. In order to predict the association (or independence) of two random variables usually a null hypothesis is posited. As for co-occurrences, the null hypothesis is: Word A and word B occur independently, that is the occurrence of word A does not cause a subsequent occurrence of word B to be more likely.

In case this hypothesis is refuted and an alternative hypothesis, namely that the random variables indeed are found to be dependent on each other, is deemed to be more likely, we have evidence of two or more words being related to each other since they co-occur more frequently than would be expected by chance (Pedersen / Kayaalp / Bruce, 1996: p. 459):

> "A significance value of .000 implies that this data shows no evidence of independence; the likelihood of having randomly selected this data sample from a population where these words are independent is zero. A value of 1.00 indicates that the data sample is exactly what would be expected from a population where the words are independent and there is no reason to doubt that this sample was drawn from such a population."

A null hypothesis for two random variables is formulated as the joint probability made up of each of the random variables:

$$P(x,y)=P(x)P(y)$$

This results in two random variables to be considered independent if the probability of both events occurring together equals the product of the probabilities of the singular events (Pedersen, 1996: p. 200):

> "If the model for independence fits the data well as measured by its statistical significance, then one can infer from this data sample that these two words are independent in the larger population. The worse the fit, the more associated the words are judged to be."

The next step in testing the significance of relationships between random variables is selecting a goodness of fit statistic (or statistical hypothesis test).

There are several problems one is facing when studying the distribution of and the relations between words. As already mentioned in section 3.2.3, words, and consequently n-grams as well, tend to be distributed according to Zipf's Law.

This in turn results in skewed or sparse data sets. By skewed data we understand data that exhibits both large and small counts whereas sparse data means that there is a large number of rare events, called singletons or hapax legomena (Jurafsky / Martin, 2000: p. 314), if these events occur only once, present in the data under examination. The problem which arises from this is that most measures for testing significance require the following conditions to hold (Read / Cressie, 1988):

1. The sample size is large,

2. the number of cells in the contingency table is fixed and small relative to the sample size, and

3. the expected count under the hypothetical population model for each cell is large.

Sparse or skewed data, as frequently present with natural language data, detracts from the reliability of measures which utilise measuring the asymptotic approximation of a statistical distribution, like a X^2 (chi-square) distribution for instance, regarding the sample distribution (Pedersen / Kayaalp / Bruce, 1996: p. 458):

> “When the data is sparse or skewed, the assumption regarding the expected count for each cell makes the asymptotic approximation unreliable.”

Another aspect of language itself is that it is “never, ever, ever, random” (Kilgariff, 2005). In contrast to other data mining settings, relations in linguistic data are “essentially non-random” (Kilgariff, 2005: p. 263) since “we speak or write with purpose” (Kilgariff, 2005: p. 264).

Take for instance one of the most common tasks in data mining: Shopping basket analysis. While this kind of setting is, just as language, susceptible to the size of the sample data it is clearly less so than linguistic data. Consider for example, supermarket shopping events: Depending on the size of the data set it is possible to reject the null hypothesis for a relation between items like nappies and sixpacks of beer but it requires a data set that is substantially larger (i.e. ten times larger) to refute the null hypothesis for shoe polish and cat food (Kilgariff, 2005: p. 265-266).

The reason for this is that languages are inherently grammatical in nature, that is, well-formed language structures follow predefined rules which are essentially non-random (Kilgariff, 2005: p. 264):

> “We [...], without computational help are not capable of, producing words or sounds or sentences or documents randomly.”

In the following subsections we shall see some of the methods for testing the goodness of fit of a particular language model and their respective advantages and shortcomings in terms of language as a subject matter of statistical analysis.

3.4.1 Pearson's Chi-Square Test

Pearson's X^2 test is a commonly used test for statistical dependence between two variables when it comes to not normally distributed probabilities. This test compares observed with expected frequencies. If the difference between these is large the null hypothesis is rejected (Manning / Schütze, 1999: p. 169) and the entities in question therefore can be considered to be statistically related.

The X^2 test is given by:

$$X^2 = \Sigma_{i,j} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

The test takes a contingency table such as the one in section 3.4 and sums over the differences between observed and expected values in each square of the table, normalised by the respective expected values. Index i iterates over the rows of the table, while j ranges over the columns. The expected values E_{ij} are computed from the marginal probabilities of the variables whose independence is under scrutiny

(Manning / Schütze, 1999: pp. 169-170). For example, given the contingency table in section 3.4 the marginal probabilities for *oil* and *industry* would be $\frac{17+29}{1382828} \approx 0.000178$ and $\frac{17+935}{1382828} \approx 0.000687$.

The test derives its name from the property that its values show a chi-square distribution if the null hypothesis holds.

Pedersen / Kayaalp / Bruce (1996: p. 458) argue that "failure to meet any of three conditions described above [the ones described in section 3.4] can mean that the actual distribution of these goodness of fit statistics is far from the X^2 approximation". They continue to make their point by stating that "in that case, significance values assigned based on the X^2 approximation can be incorrect.".

3.4.2 G Square test

The G^2 test is a variation on X^2 test. Both of these measures belong to the family of power divergence tests as they both measure the divergence between observed and expected sample counts (Pedersen / Kayaalp / Bruce, 1996: p. 457).

The G^2 test differs from X^2 as follows:

$$G^2 = 2\,\Sigma_{i,j} O_{ij} \log \frac{O_{ij}}{E_{ij}}$$

Just as the X^2 test, the G^2 test approximates a chi-square distribution if the null hypothesis is true which also means that the same kind of criticism holds for this test as well.

3.4.3 t-test

The t-test is another commonly used test in terms of collocation discovery. It makes use of the mean and the variance of a sample, with the null hypothesis being that the sample in question was drawn from a distribution with the mean μ (Manning / Schütze, 1999: p. 163). The test is given by:

$$t = \frac{x - \mu}{\sqrt{(\frac{s^2}{N})}}$$

x signifies the mean of the sample which in case of n-grams means the actual frequency of an n-gram in a corpus. As mentioned above μ stands for the hypothetic mean that would be the case if the sample had been taken from a distribution for which the null hypothesis holds. In our case this would mean a distribution where the n-gram at hand is not more probable than given by the product of the single word probabilities.

s_2 is the variance of the sample while *N* stands for the sample size, that is the denominator takes care of scaling the calculated value by the sample variance and size.

If *t* is large enough, the null hypothesis is rejected. However, it is not clear a priori what 'large enough' does mean for a given sample. The exact value for *t* to reject a null hypothesis using the t-test either has to be determined empirically or taken from statistical reference books, thus using the empirical data gathered by previous statistical work (Manning / Schütze, 1999: p. 164).

According to Pedersen / Kayaalp / Bruce (1996: p. 458) a “two-sample t-test is identical to Pearson's X^2 test when applied to a 2x2 contingency table”, which makes the t-test useful only in those cases where the conditions given in section 3.4 are met. Another common criticism brought forth against the t-test in terms of collocation discovery is that “it assumes that probabilities are approximately normally distributed” (Manning / Schütze, 1999: p. 169), which clearly is not the case for n-gram models.

3.4.4 Log-Likelihood

Likelihood ratios are an alternative to the hypothesis testing methods described above. According to Manning / Schütze (1999: p. 172) “they are more appropriate for sparse data than the X^2 test”. Moreover, a likelihood ratio is a statistic that “is more interpretable than the X^2 statistic” (Manning / Schütze, 1999: p. 172).

Likelihood ratios relate the likelihood of two hypotheses, the one assuming independence and the one assuming dependence (Manning / Schütze, 1999: p. 172):

- Hypothesis 1. $P(w^2|w^1)=p=P(w^2|\neg w^1)$

- Hypothesis 2. $P(w^2|w^1)=p_1\neq p_2=P(w^2|\neg w^1)$

The values for p, p_1, p_2 are the maximum likelihood estimates for the number of occurrences of w^1, w^2 and w^1w^2 with the frequency counts c_1, c_2 and c_{12}. The actual likelihood of each hypothesis is calculated assuming a binomial distribution (Manning / Schütze, 1999: p. 173):

$$b(k;n,x)=\binom{n}{k}x^k(1-x)^{(n-k)}$$

The likelihood ratio then is given by logarithmising the ratio between the likelihood $L(H_1)=b(c_{12};c_{1,}p)b(c_2-c_{12};N-c_{1,}p)$ for hypothesis 1 and the likelihood $L(H_2)=b(c_{12};c_{1,}p_1)b(c_2-c_{12};N-c_{1,}p_2)$:

$$\log\lambda=\log\frac{L(H_1)}{L(H_2)}$$

The logarithm results in only sufficiently large discrepancies between the values for each hypothesis for a given event being recognised. This makes likelihood ratios especially suitable in terms of natural language, as just to once again quote Kilgariff (2005):

“Language is never, ever, ever, random.”

In other words, scaling the results by using a logarithmic representation marginalises effects brought forth by natural language grammar.

3.4.5 Mutual Information

The final measure for making predictions about possible collocations that I would like to present here is pointwise mutual information.

Mutual information is a concept from information theory, which is related to the concept of entropy introduced above. This is why mutual information can be formulated in terms of entropy (Manning / Schütze, 1999: pp. 66-67):

$$I(X,Y)=H(X)-H(X|Y)=\Sigma_{x,y}\, p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$$

In other words: Mutual information "is the reduction in uncertainty of one random variable due to knowing about another" (Manning / Schütze, 1999: p. 66).

Pointwise mutual information now is given as follows (Manning / Schütze, 1999: p. 178):

$$I(x',y')=\log\frac{P(x'y')}{P(x')P(y')}=\log\frac{P(x'|y')}{P(x')}=\log\frac{P(y'|x')}{P(y')}$$

The attribute *pointwise* is derived from the fact that in contrast to mutual information we do not measure the reduction in uncertainty of a random variable given another random variable but the reduction of uncertainty of one particular event given the presence of another event. Manning / Schütze (1999: p. 178, following the definition of Fano, 1961: pp. 27-28) define pointwise mutual information as follows:

> "The amount of information provided by the occurrence of the event represented by [*y'*] about the occurrence of the event represented by [*x'*] [...]"

Manning / Schütze (1999: p. 182) continue to argue "that mutual information is a good measure of independence. Values close to 9 indicate independence (independent of frequency). But it is a bad measure of dependence because for dependence the score depends on the frequency of the individual words.". This simply means that n-grams composed of low-frequency words will have a higher mutual information than those composed of words with a high frequency in the corpus.

In specific settings, namely if one wants to find out about the dependence between two words in a document from a document collection, this problem might be amended by using TFIDF (see Chakrabarti, 2003: pp. 56-57 and section 4.2 below) for weighting the frequencies appropriately.

4. Document Indexing And Clustering

The information provided by statistical natural language models can be used for more high-level applications as well. One such application is document indexing and, building upon that, document clustering.

Document indexing, which essentially is a pre-processing sub-task of information retrieval, means indexing documents from a collection, for example a subset of the web pages available on the Internet, by the terms they contain.

This is done by building a so-called inverted index (or inverted file) with the terms as keys and a list of document identifiers for each of the terms as values. The purpose of this task is speeding up later search operations (Baeza-Yates / Ribeiro-Neto, 1999: p. 192).

This kind of index is called *inverted* because it does not index the terms by the documents they are contained in but the other way round. Hence, the indexing process basically takes a set of terms assigned to each document and transposes the resulting matrix so that each term is assigned to the documents it occurs in (Chakrabarti, 2003):

> “In effect, indexing takes a document-term matrix and turns it into a term-document matrix, and is therefore called *inverted* indexing, although *transposing* might be a more accurate description.”

Once created, such an inverted index can be used to narrow down the set of documents which is appropriate for a given query, for instance in terms of an Internet search engine.

A key issue when creating inverted indexes is choosing the right index terms, which is where properly recognised collocations might come in handy. On a more abstract level it might be useful to use methods such as Latent Semantic Indexing (LSI) to derive abstract key terms from the actual terms present in a document collection (Konchady, 2006: p. 101):

> “While the previous methods identified these relationships by counting the number of documents in which a word occurs or associating a signal/discrimination value for words, LSI builds relationships based on co-occurring words in multiple documents. These hidden underlying relationships are called the *latent semantic structure* in the collection.”

These abstract terms are considered to be the underlying concepts or topics a document refers to (Konchady, 2006: p. 101):

> “The main advantages of LSI over other methods is that it does not depend on individual words to locate documents, but rather uses a *concept* or *topic* to find relevant documents.”

After having retrieved the potentially useful documents for a user's query an information retrieval will usually seek to rank these documents with the most relevant document coming first. The first step in establishing an appropriate ranking is determining the similarity of each of the documents from the result set and the query supplied by the user. This is done by applying a vector space model to both the user query and the documents in the result set with terms as vector indices and term frequencies as values for the respective vector and index.

Subsequently, standard operations like calculating the angle between two vectors can be applied in order to derive a measure for the similarity between the entities these vectors represent.

Building upon document indexing and the ability to calculate the similarity between documents, we are capable of clustering the documents in a collection in terms of their similarity regarding the terms they contain.

The following sections will deal with each of these topics and give an outline of some of the most widely used clustering algorithms.

4.1 Inverted Index

An inverted index is a standard model in information retrieval for relating search terms to the documents they occur in. According to Baeza-Yates / Ribeiro-Neto

(1999: p. 192) an inverted index “is a word-oriented mechanism for indexing a text collection in order to speed up the searching task”.

There are several flavours of inverted indexes that differ in their granularity of addressing words in a document collection (Baeza-Yates / Ribeiro-Neto, 1999: pp. 193-195): An inverted index can, depending on the requirements of and the memory available to the system, contain anything from mere references from terms to the documents they occur in to the exact positions in the document text, the latter sometimes being called “full inverted indices” (Baeza-Yates / Ribeiro-Neto, 1999: p. 193). In-between lies a technique called block-addressing which divides the text into blocks of a fixed length and refers terms to the blocks they occur in. The information available in such a model is much more detailed than as for a mere indexing of document, yet at the same time it avoids the pitfall of excessive memory consumption a full inverted index might entail (Baeza-Yates / Ribeiro-Neto, 1999: p. 192):

> “To reduce space requirements, a technique called *block addressing* is used. The text is divided in blocks, and the occurrences point to the blocks where the word appears (instead of the exact positions)”

The examples below, taken from Baeza-Yates / Ribeiro-Neto (1999: p. 193-194) show the difference between full inverted indexes and block addressing:

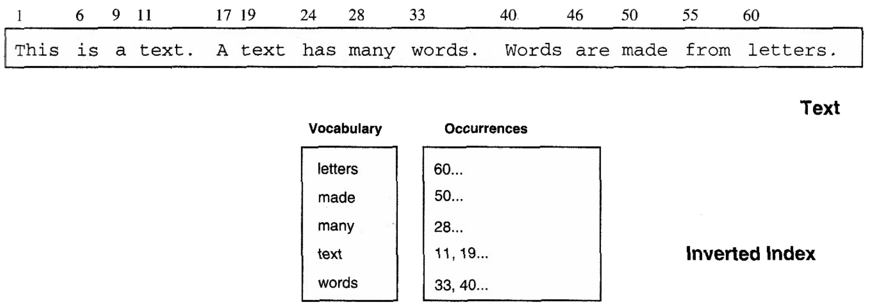


Figure 6: Inverted index

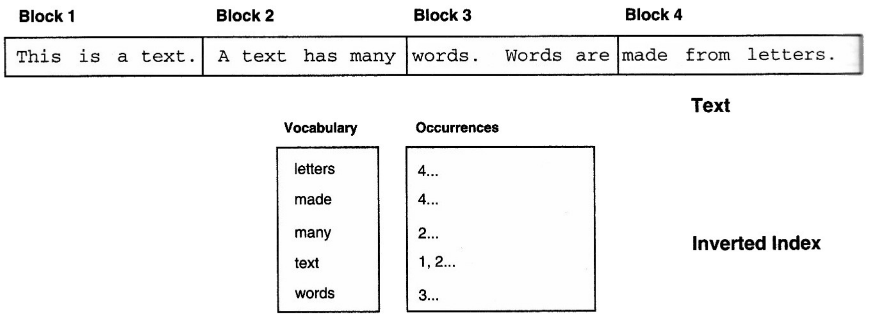


Figure 7: Block addressing

Searching on an inverted index is done using the following steps (Baeza-Yates / Ribeiro-Neto, 1999: p. 195):

- **Vocabulary search** The words and patterns present in the query are isolated and searched in the vocabulary. Notice that phrases and proximity queries are split into single words.

- **Retrieval of occurrences** The list of the occurrences of all the words found are retrieved.

- **Manipulation of occurrences** The occurrences are processed to solve phrases, proximity, or Boolean operations. If block addressing is used it may be necessary to directly search the text to find the information missing from the occurrences (e.g., exact word positions to form phrases).

An inverted index for a text of *n* characters can be constructed in *O(n)* time by using a trie structure (see example taken from Baeza-Yates / Ribeiro-Neto (1999: p. 197) below).

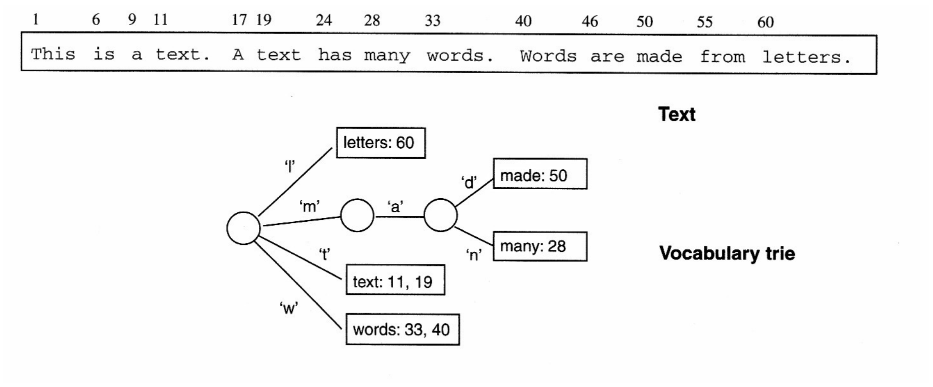


Figure 8: Using a trie for building an inverted index

4.2 Document Similarity

Given a vector space model for a document collection, standard vector operations like determining the angle between two vectors can be used for finding out about the similarity of documents.

This is useful both in terms of searching and ranking relevant documents as well as clustering documents according to their similarity (see the next section for a more detailed description of selected clustering algorithms).

A vector space model of textual data represents each document in a collection as a vector of the terms it contains (Chakrabarti, 2003: p. 56):

> "In the vector-space model, documents are represented as vectors in a multidimensional Euclidean space. Each axis in this space corresponds to a term (token)."

The value for the coordinate of a document on such an axis is determined by the TFIDF measure. This measure is calculated by multiplying the term frequency (TF) and the inverse document frequency (IDF).

The term frequency, as the name suggests, in the simplest case is given as the number of times a term occurred in a document. This simple measure may be modified in order to accommodate document length. Chakrabarti (2003: p. 56) provides the TF measure used by the Cornell SMART system as an example:

$$TF(d,t)=\begin{Bmatrix} 0\,, if\ n(d,t)=0 \\ 1+\log(1+\log(n(d,t)))\,, otherwise \end{Bmatrix}$$

n(d, t) stands for the number of times a term *t* occurs in a document *d*.

Baeza-Yates / Ribeiro-Neto (1999: p. 29) in contrast to that provide the following method for computing TF:

$$f_{i,j}=\frac{freq_{i,j}}{max_l\, freq_{l,j}}$$

In this equation $freq_{i,j}$ corresponds to the *n(d, t)* above. The "maximum is computed over all terms which are mentioned in the text of the document d_j" (Baeza-Yates / Ribeiro-Neto, 1999: p. 29).

The inverse document frequency IDF of a term is used to weight its term frequency according to its assumed relevance in the document collection at hand. This is based upon the rationale that a term that occurs in every document in the collection, no matter how frequent it is in a given document, bears little significance in terms of distinguishing between relevant and irrelevant documents (Baeza-Yates / Ribeiro-Neto, 1999: p. 29):

> “The motivation for usage of an idf factor is that terms which appear in many documents are not very useful for distinguishing a relevant document from a non-relevant one.”

The IDF of a term *t* according to Chakrabarti (2003) is given as

$$IDF(t) = \log \frac{1+|D|}{|D_t|}$$

where $|D|$ is the cardinality of the document set and $|D_t|$ represents the cardinality of the set of documents containing the term *t*.

Baeza-Yates / Ribeiro-Neto (1999: p. 29) define the IDF of the term with index *i* as

$$idf_i = \log \frac{N}{n_i}$$

where N corresponds to $|D|$ and n_i is equivalent to $|D_t|$ in the equation above.

TFIDF provides weighting for term frequencies. Without TFIDF, terms that occur in every document (such as stopwords) could be considered as highly significant, while with TFIDF due to the logarithm in the IDF equation they do not matter at all. As they occur in every document their value for the ratio in IDF equals 1 which translates into a logarithmised IDF, and hence TFIDF, value of close to or depending on the implementation exactly 0.

An example of a simple document vector is given below. The figures have been arbitrarily chosen for illustration purpose, they do not represent actual TFIDF values:

$$d = \begin{pmatrix} this:5 \\ is:4 \\ a:3 \\ text:6 \end{pmatrix}$$

A frequently used measure for computing the similarity between two document vectors is the so-called cosine similarity which yields the cosine of the angle between two document vectors. If the cosine equals 0 the corresponding angle is 0º, which means that documents in question are an exact match. Cosine similarity is defined as

$$\cos\alpha = \frac{A \cdot B}{\|A\| \|B\|}$$

where both A and B are document vectors. In information retrieval applications the notion of document is extended to the query provided by the user. This means that the query is merely understood as another document which can be matched against the documents in the collection in order to retrieve the relevant documents for the user's query.

Another similarity measure is the Jaccard coefficient (Chakrabarti, 2003: p. 68):

$$r'(d_{1,} d_2) = \frac{|T(d_1) \cap T(d_2)|}{|T(d_1) \cup T(d_2)|}$$

T(d_1) and *T(d_2)* represent the sets of terms occuring in d_1, d_2 respectively.

4.3 Document Clustering

In the following subsections I shall describe K-means, one of the most well-known clustering algorithms, in more detail, as well as outline some alternatives to this particular algorithm.

4.3.1 K-means

A frequently used heuristic clustering algorithm is K-means. This algorithm has a complexity of *O(n log n)* which makes it suitable even for larger document collections (Konchady 2006: p. 276).

The name is derived from the main concept the algorithm is built upon: K-means starts off with an initial set k representatives – one for each cluster – taken from the document collection (or more generally, data points) at hand. After the algorithm is finished, each of the k representatives is intended to be the mean (also called centre or centroid respectively) of a cluster.

The geometrical concept of centroid is related to the physical concept of centre of gravity. Informally, a centroid can be understood as the average of all data points of a cluster. Hence, each of those centroids is considered to be the entity from the cluster that represents it best.

What exactly is meant here by 'represents best' depends on the feature space the entities from the collection are characterised by. The most readily accessible features are term frequencies with each term representing a vector index and each frequency being the value for this index and the respective document vector. Another possible set of features could be provided by LSI, as described above.

There are some preliminary aspects to consider regarding K-means (Konchady, 2006: pp. 276-277):

First, there are several ways to determine the initial set of *k* cluster representatives, the first being randomly picking them. The problem with this approach is, that despite being attractive due to simplicity, it is bound to give different results for repeated runs of the algorithm and hence entails varying precision for varying initial sets of representatives.

A second method is taking a small sample from the document collection and using this to determine likely initial representatives. This can be achieved by having a preliminary clustering run on that sample, which should result in a likely set of initial centroids without wasting too much computing time, given the sample is reasonably small.

The third way described here is to pick cluster centres by conducting a density test. A density test in this case is similar to voting methods used in certain classification algorithms such as k-nearest-neighbour (see Konchady (2006: pp. 323-325)). Given a sample set an entity from that set is chosen as a cluster representative if a certain number of entities has a similarity greater than some threshold.

The second aspect to consider is the actual value for *k*, that is how many clusters we want the algorithm to result in (Konchady, 2006: pp. 276). This value has to be selected carefully as it greatly affects recall and precision. Consider Manning / Schütze (1999: p. 192) on forming equivalence classes and the ambivalence between reliability and discrimination:

> "[...] dividing the data into many *bins* gives us greater discriminiation. Going against this is the problem that if we use a lot of bins then a particular bin may contain no or a very small number of training instances [...]"

Given a measure of goodness for a set of clusters, it is possible to determine the best value for *k* for a document collection. Konchady (2006: p. 276-277) describes some measures for achieving this.

The K-means algorithm itself can actually be summarised in a few simple steps (Konchady, 2006: p. 277-278):

1. Select *k* documents from the collection to form *k* initial singleton clusters.

2. Repeat until termination conditions are satisfied.

    2.1 For every document *d*, find the cluster *i* whose centroid is most similar, assign *d* to cluster *i*.

    2.2 For every cluster *i,* recompute the centroid based on the current member documents.

    2.3 Check for termination – minimal or no changes in the assignment of documents to clusters.

3. Return the list of clusters.

Regarding step 2.1, the similarity between a document vector and a centroid can for example be defined as the cosine similarity between those two vectors, or alternatively, as the length of the distance vector between the document vector and the centroid.

For further illustration of step 2.2, consider the following figure taken from Konchady (2006: p. 277):

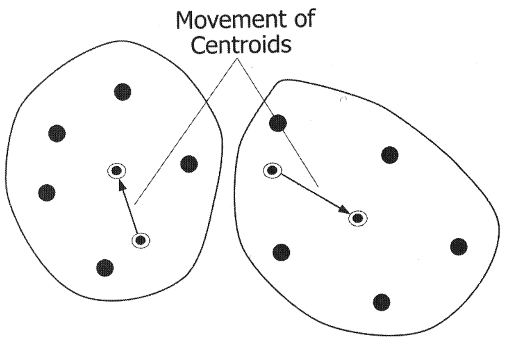


Figure 9: Movement of Centroids

Another, more formal account of K-means is given by Bishop (2006: p. 424). The goal of K-means is to minimise a distortion measure given by:

$$J = \Sigma_{n=1}^{N} \Sigma_{k=1}^{K} r_{nk} \| x_n - \mu_k \|^2$$

Here, x_n stands for the vector representing data point n while μ_k is the centroid (and hence cluster) such a data point is assigned to.

r_{nk} is given as follows:

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \| x_n - \mu_j \|^2 \\ 0 & \text{otherwise} \end{cases}$$

Hence, J represents “the sum of the squares of the distances of each data point to its assigned vector μ_k” (Bishop, 2006: pp. 424-425).

4.3.2 Simulated Annealing

According to Konchady (2006: p. 279) K-means “gets stuck in local minima and is sensitive to the set of initial clusters”.

A method that avoids these pitfalls is simulated annealing (Konchady, 2006: pp. 279-281). As suggested by the name simulated annealing draws an analogy to metallurgic heat treatment and hence uses temperature as a metaphor for building clusters.

A termination of the algorithm at local minima is avoided by alongside with a general downhill movement (i.e. lowering the temperature) allowing occasional uphill movements (i.e. increasing the temperature again).

The steps of the algorithm are described by Konchady (2006: p. 279):

```
1. Get the initial set of clusters and set the initial temperature parameter
   $T$.
2. Repeat until the temperature is reduced to the minimum.
    2.1 Run a loop $x$ times
        2.1.1 Find a new set of clusters by altering the
              membership of some documents.
        2.1.2 Compare the difference between the values of
              the new and the old set of clusters. If there is an
              improvement, accept the new set of clusters,
              otherwise accept the new set of clusters with a
              probability $p$.
    2.2 Reduce the temperature based on the cooling schedule.
3. Return the final set of clusters.
```

4.3.3 Genetic Algorithms

Genetic algorithms are a class of algorithms that can be used for clustering as well in order to avoid getting stuck in local minima. These algorithms like simulated annealing use specific metaphors, which in this case are taken from genetics.

Genetic algorithms instead of "working with one solution at a time [...] create a population of solutions" (Konchady, 2006: p. 282).

The clustering problem is solved by arranging the documents in a circle with documents which are similar close to each other and subsequently finding the key documents in that circle in order to build clusters from their neighbourhood (Konchady, 2006: p. 282).

Each of these arrangements is considered to be a *chromosome* (or solution) of *x* bits. Each chromosome has a fitness score defined as (Konchady, 2006: p. 282)

$$fitness(S) = \Sigma_{i=1}^{n} \Sigma_{j=1}^{r} \mathrm{sim}(i, i-j) + \mathrm{sim}(i, i+j)$$

where *r* means the size of the neighbourhood of similar documents for each document.

Each iteration of the algorithm then can be divided into three steps (Konchady, 2006: p. 282):

1. Pick two parent solutions *x* and *y* from the set of all solutions with a preference for solutions with higher fitness scores.

2. Use a crossover operator to combine *x* and *y* to generate a new solution *z*.

3. Periodically, *mutate* a solution by randomly exchanging two documents in a solution.

4.3.4 Scatter-Gather

Scatter-Gather is a clustering algorithm originally conceived for assisting the user in iteratively selecting the relevant documents for a query (Konchady, 2006: p. 285). This algorithm goes from a broad categorisation of documents to more detailed distinctions for each iteration.

It basically consists of two operations *scatter* and *gather*, which alternate during the execution of the algorithm. *Scatter* clusters the currently focussed documents whereas *gather* selects a few relevant clusters for the next iteration (Konchady, 2006: p. 286).

The Scatter-Gather process can be summarised as follows (Konchady, 2006: p. 286):

1. Initially cluster all documents in the collection.

2. Use a keyword-based query to fetch a list of clusters or specify a list of initial clusters.

3. Repeat in a loop.

    3.1 **Scatter:** Cluster the documents from the selected clusters.

    3.2 **Gather:** Select a few clusters from the results.

    3.3 **Terminate:** Terminate if there are too few documents or too few clusters.

5. TextSieve: Creating and Using Language Models

Alongside with the methods discussed in this paper, I developed a web application called TextSieve in order to exemplify some of the aspects of statistical language modelling and to provide an easy-to-use and easy-to-extend framework for work involving statistical language models.

The ideas and concepts regarding the software described in the sections to follow are based upon previous work regarding the Perl module Text-NSP (Ngram Statistics Package) by Pedersen et al.[3]

The software has been implemented as a web application using the Groovy scripting language[4] for the Java Virtual Machine and, building on this, the web application framework Grails[5].

The decision to implement this software as a web application has been made since it caters for a balance between ease of implementation and ease of use.

3 See Banerjee, S. / Pedersen (2003) or http://search.cpan.org/dist/Text-NSP/ for more information on this software package.
4 See http://groovy.codehaus.org/ .
5 See http://www.grails.org/ .

Grails and Groovy have been chosen because while relying on the performance of the Java system architecture and the copious supply of libraries available for the Java programming language, they still provide the ease of use of scripting languages like Perl, Python or Ruby and the productivity of web application frameworks like Ruby on Rails[6]. Keeping the objective of extensibility in mind, opting for an environment that supports a lot of potentially useful third-party software packages, was a major concern when developing this application.

5.1 Feature Overview

The software basically comprises two major features: Language model creation and measuring the statistical dependency between unigram types.

5.1.1 Language Model Creation

TextSieve takes a corpus and extracts all n-grams of user-defined length *n* in a window of size *m*. This means for instance, that given $n = 2$ and $m = 3$ the software will find not only all the n-grams which are formed by directly adjacent words, but those with one word in-between as well. For example, given the string 'under thorough investigation', TextSieve will recognise 'under thorough', 'under investigation' and 'thorough investigation'.

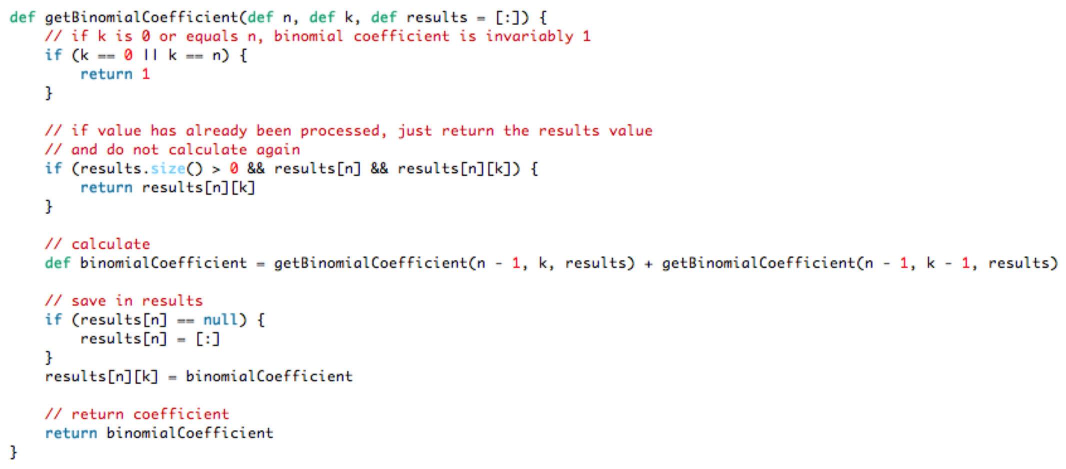
In order to accomplish this task the binomial coefficient defined as

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

has to be calculated beforehand to find the *n* possible combinations that can be taken from a set of size *m*. Combination in this case has to be understood in terms of the strict mathematical sense: An unordered collection of *n* items, as opposed to an ordered collection or permutation. This means that, given the example above, we want to consider 'under investigation', but not 'investigation under'. This at first might seem counterintuitive as word order matters in natural language. However, the set ['under', 'investigation'] abstracts from element order while using permutations we would have to consider both ('under', 'investigation') and ('investigation', 'under').

6 See http://www.rubyonrails.org/ .

The following code excerpt displays the recursive algorithm used for calculating the binomial coefficient:



```
def getBinomialCoefficient(def n, def k, def results = [:]) {
    // if k is 0 or equals n, binomial coefficient is invariably 1
    if (k == 0 || k == n) {
        return 1
    }

    // if value has already been processed, just return the results value
    // and do not calculate again
    if (results.size() > 0 && results[n] && results[n][k]) {
        return results[n][k]
    }

    // calculate
    def binomialCoefficient = getBinomialCoefficient(n - 1, k, results) + getBinomialCoefficient(n - 1, k - 1, results)

    // save in results
    if (results[n] == null) {
        results[n] = [:]
    }
    results[n][k] = binomialCoefficient

    // return coefficient
    return binomialCoefficient
}
```

Figure 10: Calculating the binomial coefficient

The second problem that has to be solved after we have calculated the number of combinations to extract from a window is getting the actual combinations from a window. The algorithm shown below draws upon the Combination Generator by Michael Gilland of Merriam Park Software[7], who in turn refers to Rosen (1991: pp. 284-286) for a description of the algorithm.

7 See http://merriampark.com/comb.htm .

The following piece of Groovy code gets all token combinations from a window:

```
/*
 * method for getting token combinations
 * algorithm based on Combination Generator by Michael Gilleland, Merriam Park Software,
 * see http://www.merriampark.com/comb.htm for further information
 */
def getTokenCombinations(def windowSize, def ngramSize) {
    def tokenCombinations = []

    // get binomial coefficient in order to determine number of steps to do;
    // window and ngram size are decremented by 1 for this purpose, as some
    // possible ngram combinations will be covered by the following windows
    // anyway and hence would be counted twice
    def todo = getBinomialCoefficient(windowSize - 1, ngramSize - 1)

    // build first combination
    def combination = []
    for (i in 0 .. ngramSize - 1) {
        combination.add(i)
    }

    // initialise counter
    def counter = 1

    // add first stack to token combinations
    def newCombination = []
    newCombination.addAll(combination)
    tokenCombinations.add(newCombination)

    // first step done, therefore increment counter
    counter++

    // process counter + 1 stacks
    while (counter <= todo) {
        // set index i = ngram - 1, as indices begins at 0
        def i = ngramSize - 1

        // while current combination value equals window size - ngram size + current index i, decrement index i
        // e.g.: window = 5, ngram = 3, current combination = 1 3 4, i = 1
        // => value at combination position = window - ngram + i
        while (combination[i] == windowSize - ngramSize + i) {
            i--
        }

        // increment value at current combination position
        combination[i]++

        // re-initialise values of following combination positions (j) according to incremented value
        // set to value at current position (i) + delta(i, j)
        for (j in i + 1 .. ngramSize - 1) {
            if (j < ngramSize) {
                combination[j] = combination[i] + j - i
            }
        }

        // add combination to token combinations
        newCombination = []
        newCombination.addAll(combination)
        tokenCombinations.add(newCombination)

        // increment counter
        counter++
    }

    // return token combinations
    return tokenCombinations
}
```

Figure 11: N-gram combinations

5.1.2 Association Between Words

The second major feature of TextSieve makes use of some of the methods described in section 3.4 to test the statistical association between unigram types in a corpus.

The software currently provides two such measures:

- chi-square (see section 3.4.1)
- pointwise mutual information (see section 3.4.5)

A major difference from the way for example Text-NSP handles this task, is that TextSieve does not simply calculate the associations for all n-grams in a corpus, but allows for limiting the analysis to those n-grams which are actually interesting to the user (see section 6.1 for a use case). Depending on the user selection this can vastly reduce the space to operate upon, which in turn leads to less processing time.

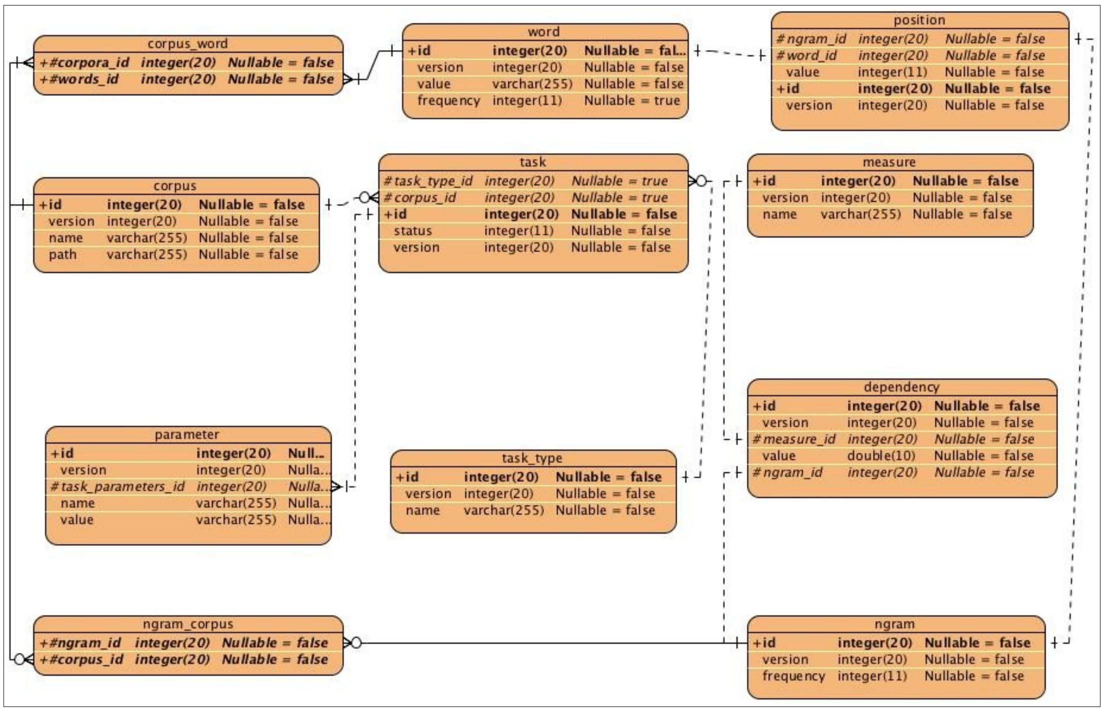
For selecting the n-grams to be analysed during this step queries like the following can be supplied by the user:

'this':1, 'test':4

This example stands for “select all n-grams having 'this' as their first word and 'test' as their fourth”. Apart from this rather succinct notation, TextSieve tries to be more explicit about what happens behind the scenes by allowing a more verbose syntax like “select n-grams having 'this' in position:1 and 'test' in position:4” as well.

5.2 Design

In this section I shall give an overview of the software design behind TextSieve. First of all, the following diagram shows the entity relationship model for TextSieve. I opted for storing the data in a relational database in favour of flat text files because I wanted the data to be easily accessible, especially by other software packages.



corpus_word
+#corpora_id integer(20) Nullable = false
+#words_id integer(20) Nullable = false
word
+id integer(20) Nullable = fal...
version integer(20) Nullable = false
value varchar(255) Nullable = false
frequency integer(11) Nullable = true
position
#ngram_id integer(20) Nullable = false
#word_id integer(20) Nullable = false
value integer(11) Nullable = false
+id integer(20) Nullable = false
version integer(20) Nullable = false
corpus
+id integer(20) Nullable = false
version integer(20) Nullable = false
name varchar(255) Nullable = false
path varchar(255) Nullable = false
task
#task_type_id integer(20) Nullable = true
#corpus_id integer(20) Nullable = true
+id integer(20) Nullable = false
status integer(11) Nullable = false
version integer(20) Nullable = false
measure
+id integer(20) Nullable = false
version integer(20) Nullable = false
name varchar(255) Nullable = false
dependency
+id integer(20) Nullable = false
version integer(20) Nullable = false
#measure_id integer(20) Nullable = false
value double(10) Nullable = false
#ngram_id integer(20) Nullable = false
parameter
+id integer(20) Null...
version integer(20) Nulla...
#task_parameters_id integer(20) Nulla...
name varchar(255) Nulla...
value varchar(255) Nulla...
task_type
+id integer(20) Nullable = false
version integer(20) Nullable = false
name varchar(255) Nullable = false
ngram_corpus
+#ngram_id integer(20) Nullable = false
+#corpus_id integer(20) Nullable = false
ngram
+id integer(20) Nullable = false
version integer(20) Nullable = false
frequency integer(11) Nullable = false

Figure 12: ER model

The following component model gives a rough overview of the components TextSieve consists of. Much of this structure is already provided by the Grails framework. Grails follows the Model-View-Controller Pattern by separating presentation, business logic and domain model layers. Additionally Grails provides a service layer for encapsulating functionality that will be used by more than one controller or domain object.

Finally, TextSieve makes use of the Quartz library for Java[8] and the corresponding Grails plugin[9] for scheduling counting and hypothesis testing tasks.

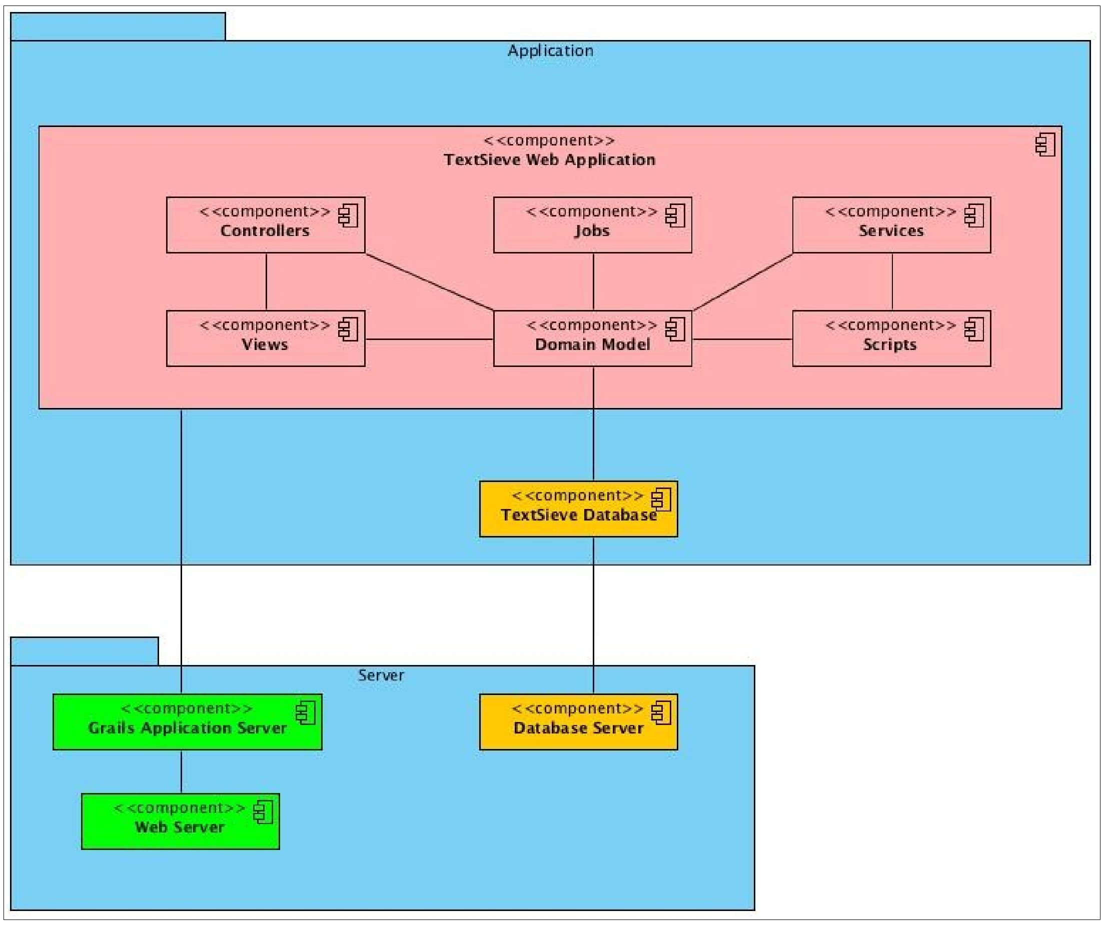


Figure 13: Component model

8 See http://www.opensymphony.com/quartz/ .
9 See http://grails.codehaus.org/Quartz+plugin .

The following class models render a more detailed view of the domain and service layers:

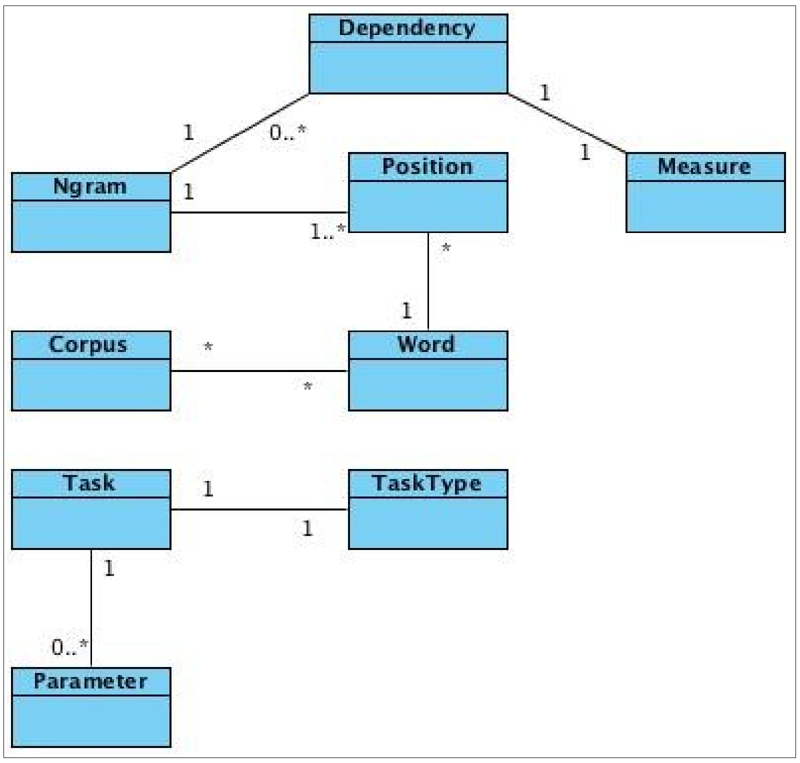


Figure 14: Domain classes

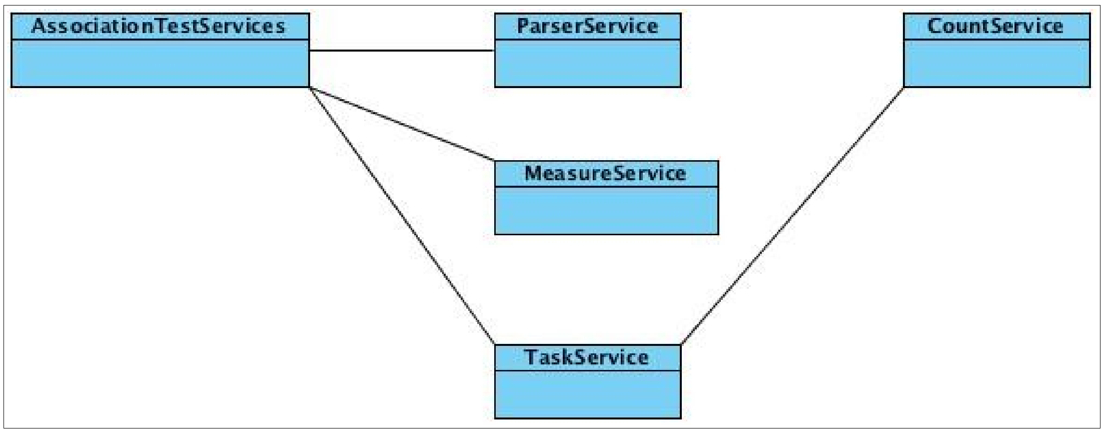


Figure 15: Service classes

The following state machine diagrams display the processes described in 5.1.1 and 5.1.2:

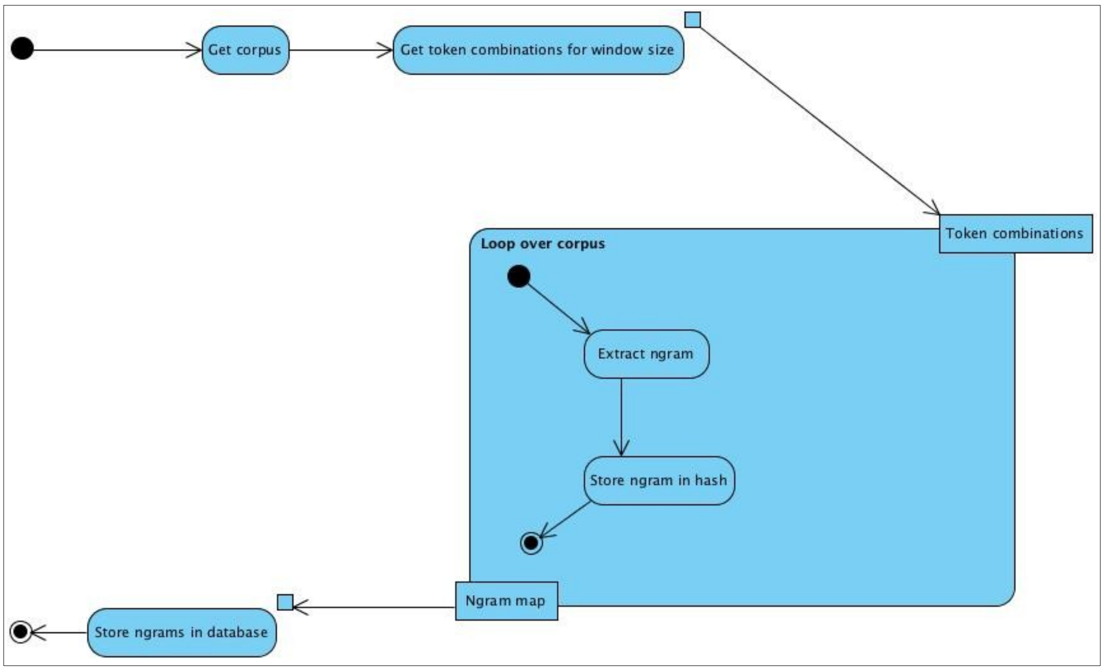


Figure 16: Counting n-grams

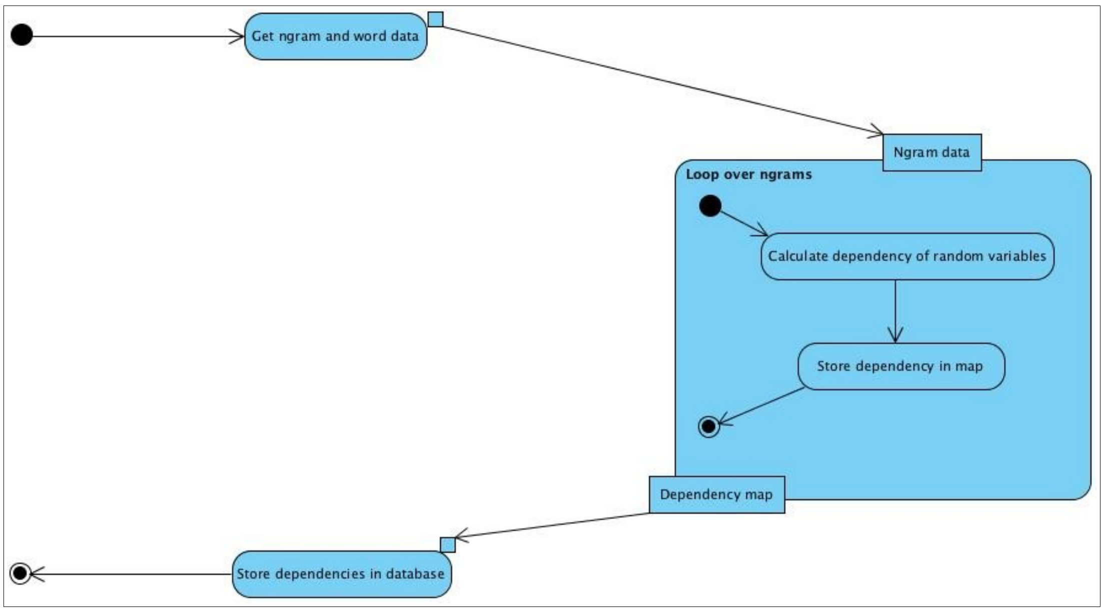


Figure 17: Calculating dependencies

The sequence diagrams below describe the same processes, however, from the perspective of participating objects and the messages exchanged between these objects to accomplish the task.

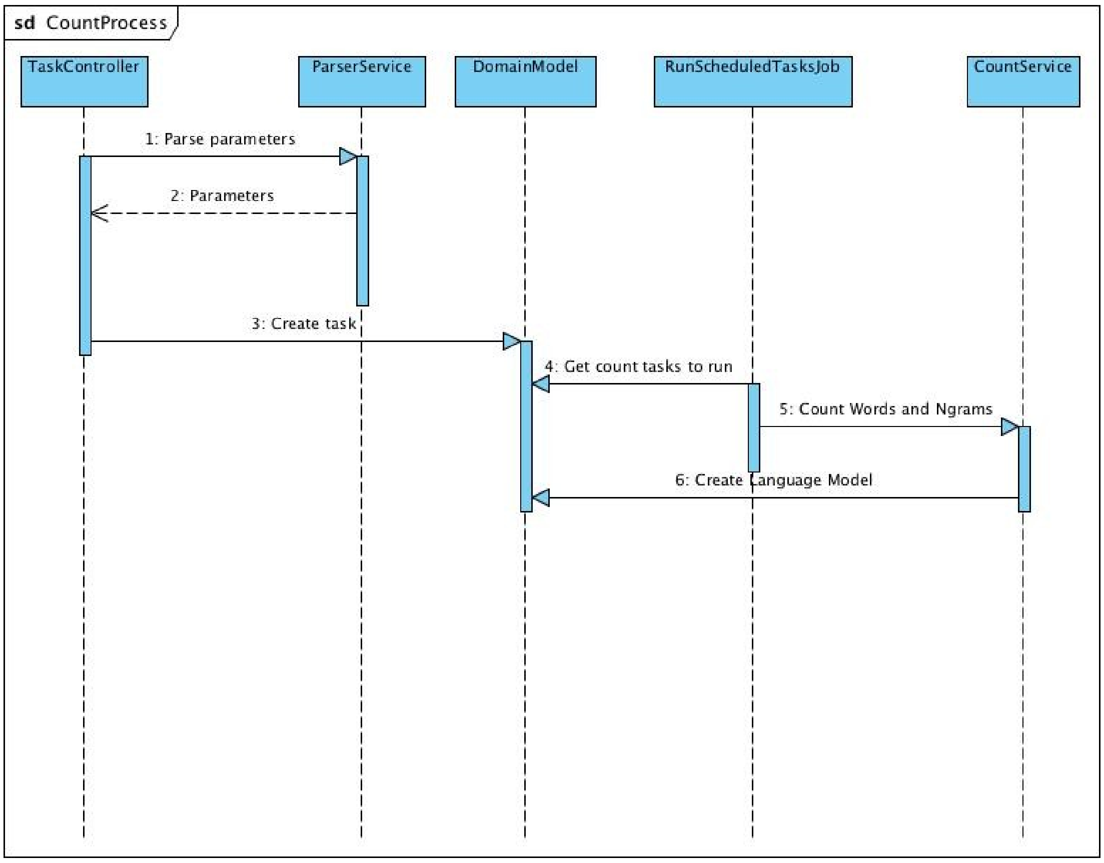
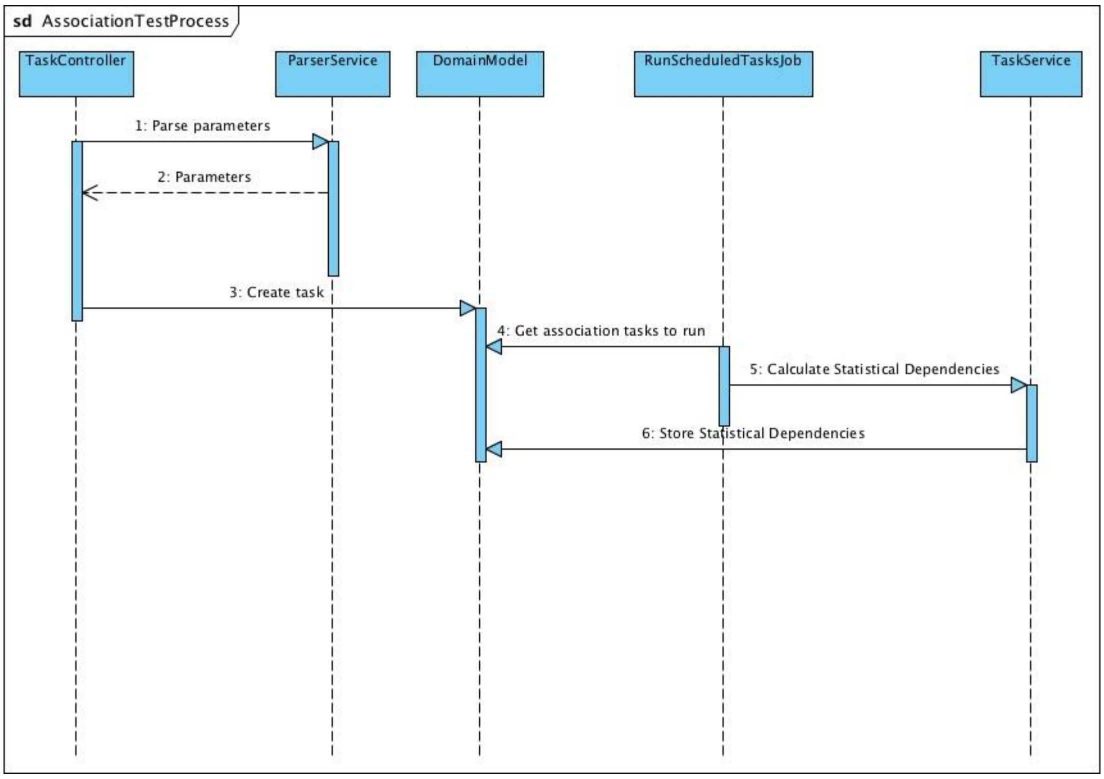


Figure 18: Count process


sd AssociationTestProcess
TaskController
ParserService
DomainModel
RunScheduledTasksJob
TaskService
1: Parse parameters
2: Parameters
3: Create task
4: Get association tasks to run
5: Calculate Statistical Dependencies
6: Store Statistical Dependencies
Figure 19: Association test

5.3 Testing

TextSieve was developed using a test-driven development (TDD) approach. This means that before an actual method was written, a unit test was implemented. Unit tests ensure that each component of a software for itself works correctly. In TDD a test is used to define the results a method is supposed to yield. A method consequently is considered to be correct only if the corresponding test succeeds.

It is important to note that unit testing does not increase the quality of existing code but its purpose is to retain the already achieved level of quality. This means that unit tests prevent errors that crept in due to code changes from going unnoticed.

In each test, assertions state the expected outcome of the method to be tested. The getBinomialCoefficient and getTokenCombinations methods shown in section 5.1.1, for example are tested and defined by the following tests:

```
void testBinomialCoefficientGetter() {
    def countService = new CountService()

    assert countService.getBinomialCoefficient(2, 2) == 1
    assert countService.getBinomialCoefficient(3, 2) == 3
    assert countService.getBinomialCoefficient(4, 2) == 6
    assert countService.getBinomialCoefficient(5, 3) == 10
}
```

Figure 20: Unit test for binomial coefficient getter

```
void testTokenCombinationsGetter() {
    def countService = new CountService()

    def tokenCombinations = countService.getTokenCombinations(3, 2)
    assert tokenCombinations.size() == 2
    assert tokenCombinations[0] == [0, 1]
    assert tokenCombinations[1] == [0, 2]

    tokenCombinations = countService.getTokenCombinations(4, 2)
    assert tokenCombinations.size() == 3
    assert tokenCombinations[0] == [0, 1]
    assert tokenCombinations[1] == [0, 2]
    assert tokenCombinations[2] == [0, 3]

    tokenCombinations = countService.getTokenCombinations(5, 3)
    assert tokenCombinations.size() == 6
    assert tokenCombinations[0] == [0, 1, 2]
    assert tokenCombinations[1] == [0, 1, 3]
    assert tokenCombinations[2] == [0, 1, 4]
    assert tokenCombinations[3] == [0, 2, 3]
    assert tokenCombinations[4] == [0, 2, 4]
    assert tokenCombinations[5] == [0, 3, 4]
}
```

Figure 21: Unit test for token combination getter

6. Linguistic Research and Applications

In this section I will outline two areas of linguistic research in which the text mining methods described above are applied.

The first of these is the productivity of structures like determinerless prepositional phrases (determinerless PPs or, according to Baldwin et al. (2006) PP-Ds), which sometimes are considered to be idiomatic, and hence essentially non-productive.

The second area I would like to introduce is concerned with lexicographic issues like automatically finding semantically related words as well as collocations and fixed expressions.

6.1 Lexicography

The term lexicography as used in this section means both the process of compiling dictionaries as well as the discipline dealing with theoretical issues like how the vocabulary of language is organised in terms of syntactic and semantic relations.

Discovering fixed expressions such as 'save from sth.' for inclusion in a dictionary used to require a lot of intuition from the lexicographer (Church / Hanks, 1991: p. 29):

> “The triangulation [i.e. the use of context for defining the meaning of a word] requires “art”. How does the lexicographer decide which potential cut points are “interesting” and which are merely due to chance?”

Drawing upon the psycholinguistic notion of word association norm Church / Hanks (1991) extend this notion to an information theoretic view and propose the use of pointwise mutual information for gaining objective evidence of associated words instead of merely relying on the intuition of the lexicographer (Church / Hanks, 1991: p. 29):

> “The proposed association ratio score provides a practical and objective measure that is often a fairly good approximation to the “art”.”

Church / Hanks (1991: p. 22) put forward several further applications the data gained by their approach might be used for:

> “[...] (a) constraining the language model both for speech recognition and optical character recognition (OCR), (b) providing disambiguation cues for highly ambiguous syntactic structures such as noun compounds, conjunctions, and prepositional phrases, (c) retrieving texts from large databases (e.g. newspapers, patents) [...]”

Halama (2002) uses a modified t-test and likelihood ratios for determining the semantic similarity of words.

This is done by comparing all words which share a certain context, that is for instance the initial words in collocations like 'mindful consideration' or 'deliberate consideration'.

The t-test or alternatively a log-likelihood measure is used to compare the frequency of one collocation in a corpus with the frequency of another. Words which roughly show the same statistical distribution given a certain context are considered to be so similar in terms of semantics that they can be exchanged for each other.

The modified t-test is defined as follows (Halama, 2002: p. 46):

$$t=\frac{x_1-x_2}{\sqrt{(\frac{s_1^2}{n_1}+\frac{s_2^2}{n_2})}}$$

In this particular case this boils down to the following (Halama, 2002: p. 47):

$$t\approx\frac{C(v^1 w)-C(v^2 w)}{\sqrt{(C(v^1 w)+C(v^2 w))}}$$

C signifies the frequency of a bigram consisting of the words v^1 w, v^2 w respectively. This means that this particular flavour of the t-test does not make predictions about the statistical dependence (or independence, for that matter) between two words, but by substituting one word in a collocation for another compares the distribution of those two words in the given context.

Both the t-test and the log-likelihood measure yield smaller results for words which are considered to be semantically similar given the criteria outline above.

Hence, by the methods described in this paper, it is not only possible to infer collocations and fixed expressions, but they can be used for deriving information about the semantic proximity (and to a certain degree exchangeability) of words as well.

6.2 Productivity of Determinerless PPs

The treatment of determinerless PPs and their appropriate incorporation into a theory of grammar is a recent area of linguistic research.

Baldwin et al. (2006: p. 163) define determinerless PPs as “made up of a preposition (*P*) and a singular noun (N_{sing}) without a determiner”.

Determinerless PPs occur in a variety of languages, including English, Tagalog and German (Himmelmann, 1998). The following phrases are but a few examples from German combining the preposition 'unter' (*under*) with a singular noun:

- unter Vorbehalt (*with reservations*, e.g as in *recommended with reservations*)
- unter Voraussetzung (*provided that*)
- unter Verweis (*with reference to*)

The following examples show English determinerless PPs, which display a combination of the prepositions 'in' and 'at' with a subsequent singular noun:

- in church
- in gaol
- at school
- at lunch

Although a rather frequent structure, for example both in written and spoken German, determinerless PPs pose quite a few problems to grammar theory.

At the first glimpse, one could assume these structures to be nothing less but usual PPs. However, this contradicts conventional grammars for German in that prepositions are usually said to combine only with plural nouns if the determiner is omitted (Kiss, 2006: p. 64).

A seemingly simply solution to this problem obviously is to extend this rule to singular nouns as well.

However, there are two problems coming with this approach: First of all, speakers of German do not seem to be able to creatively coin new expressions of this kind, which they should be if this process was rule-governed and hence part of the speaker's linguistic competence (Kiss, 2006: p. 65):

> “Speakers of German are not instantly able to coin new combinations of *P+Noun*, and also find it highly problematic to judge the grammaticality of *P+Noun* combinations, if these are offered without a context.”

Furthermore, such an extension of the rule is not in line with how prepositions and singular nouns generally combine in German. The following determinerless PP is perfectly grammatical in German (Kiss, 2007: p. 319):

(1) Auch Philippe Egli besteht auf einer eigenen Handschrift – **unter Voraussetzung** des Einverständnisses des Ensembles. [Philippe Egli insists on a proper handwriting as well – provided the cast agree]

If the rule really was that prepositions combined with singular nouns to form regular PPs, the following combination should be possible as well. In this combination the singular noun of the determinerless PP 'unter Voraussetzung' has been replaced by the noun 'Prämisse', which usually is a synomym for 'Voraussetzung' (Kiss, 2007: p. 319):

(2) * Auch Philippe Egli besteht auf einer eigenen Handschrift – **unter Prämisse** des Einverständnisses des Ensembles.

However, while the first sentence is acceptable to speakers of German, the second one clearly is ungrammatical. The fact that 'Prämisse' is a synonym for 'Voraussetzung' precludes the possibility that determinerless PPs are only possible for nouns which share certain semantic properties right away.

Another approach of accommodating determinerless PPs is simply listing all of them, implying that they are so-called listemes (Kiss, 2006: p. 66)

Despite these issues, empirical evidence of determinerless PPs being productive and therefore rule-governed actually can be found.

Kiss (2006: pp. 66-68) extends Baayen's (2001: pp. 203-210) method for measuring the productivity of morphological structures to syntactic structures. The method measures the likelihood *R(N)* of seeing new instances of $P{+}N_{sing}$ combinations, given a corpus of *N* words, the vocabulary *V(N)* and the vocabulary of hapax legomena *V(1, N)*.

R(N) is defined as (Kiss (2006: pp. 66):

$$\frac{E[V(1,N)]}{N}$$

where *E[V(1, N)]* stands for the expected value of the number of hapax legomena given a text consisting of *N* words. Therefore the formula above yields the probability of a new type occurring after *N* words.

If the process forming determinerless PPs is indeed productive, both *V(N)* and *V(1, N)* will increase steadily. Consequently, *R(N)* will be low but will not converge to zero, since the class of nouns is infinite which results in infinite possibilities for creating determinerless PPs as well (Kiss, 2006: p. 67)

If, however, the process is unproductive, *V(1, N)* at some point will stop to grow and hence *R(N)* will be asymptotic to zero.

Using this method, Kiss (2006: p. 67-70) is capable of showing that for a corpus comprising 106 million words, extracted from the 1995-1998 editions of the *Neue Zürcher Zeitung*, *V(1, N)* shows a steady increase for an increasing number of words. Hence, *R(N)* does not approach zero for an increasing value for N. This leads to the conclusion "that the construction is not finite in its extension" (Kiss, 2006: p. 67).

Furthermore, Kiss (2006: p. 68, 70) compares the distribution of $P+N_{pl}$ constructions, which are licensed by a grammar rule and therefore clearly are created by a productive process. The results clearly show that, although less frequent, $P+N_{sing}$ constructions show a similar distribution over time, which lends itself to interpreting these constructions as being rule-governed as well.

These results combined with the obvious discrepancy between empirical and cognitive productivity, that is the inability of speakers of German to coin new constructions of this kind (Kiss, 2007: p. 342), question the assumption that "speakers of a language have tacit knowledge, i.e. access to all rules making up language" (Kiss, 2006: p. 68).

These findings are corroborated by Dömges et al. (2007). They examine two LNRE models, a finite Zipf-Mandelbrot model (*fZM*) and a general Zipf Mandelbrot model (*ZM*), as to which fits the observed distribution of new types of $P+N_{sing}$ construction for an increasing word count best. The corpus, containing 213 million words, was taken from the 1995-1998 editions of the *Neue Zürcher Zeitung* and 1997-1999 editions of the *Frankfurter Rundschau*. As already suggested by their name, the two models basically differ in that the *fZM* assumes a minimal probability brought about by a finite set of types for a sufficiently large number of words *N*. The general *ZM* however, neither does assume a minimal probability, nor a finite set of types, but essentially predicts that the number of types is infinite (Dömges et al., 2007: p. 33):

> "The fZM assumes a minimal probability A [...]. This amounts to the assumption that the vocabulary size itself is finite. Hence, it can be expected according to the fZM that the set of observed types does not increase once $N \approx \frac{1}{A}$ is reached. In the general ZM model there is no such minimal probability."

If the number of P+N_{sing} types is finite implying that these constructions are merely listemes and not created by a productive rule of grammar, the *fZM* should fit the distribution of new types better than the general *ZM*. If, however, P+N_{sing} combinations come about by a productive process the general *ZM* should fit their distribution better than the finite one.

As shown by Dömges et al. (2007: pp. 35-36) the general *ZM* indeed fits the distribution of P+N_{sing} combinations better than the *fZM*, which further underlines the conclusion that P+N_{sing} combinations are rule-governed and deserve being considered in grammar models.

Additionally, Kiss (2007: p. 330-334) provides further evidence of the productivity of the structures at hand by comparing the log-likelihood distributions of P+N_{sing} and P+N_{pl} combinations. The rationale behind this approach is using log-likelihood ratios to determine how strongly a particular preposition and the accompanying singular noun are connected.

Kiss (2007: p. 333) shows that both classes exhibit an approximately equal number of very strongly connected combinations, as well as a generally similar behaviour in terms of their respective log-likelihood distribution. This once again implies that both classes are to be treated equally.

Finally, considering further research, clustering methods like K-means could be used to cluster determinerless PPs according to the properties of the nouns and / or the prepositions they are made up of. This information could be used to find regularities among this vast class of linguistic structures, possibly finding the features which are responsible for the way some singular nouns combine with prepositions without a determiner and some do not. Furthermore, Latent Semantic Indexing (LSI) might be used for finding the most appropriate abstract dimensions along which to optimally distinguish between different classes.

7. Conclusion

Today, natural language data is available in abundance. Be it (X)HTML documents on the Web or natural language corpora specifically designed for tasks in linguistic research: This data can be used for a variety of purposes ranging from industry applications of text mining to fundamental linguistic research.

Natural language data comes in a variety of flavours, from the most basic plain text corpora to corpora that provide additional information about linguistic features like part-of-speech tags, morphological and syntactic structure and semantics. Moreover, hyperlinked documents, which usually means (X)HTML documents, provide additional semantics by special tags signalling a document title or different paragraphs and, even more importantly, links between single documents.

Depending on the application, each of these pieces of information can provide useful information, from the mere frequency counts and co-occurrences of a plain text corpus to deep linguistic structures that for example can help in disambiguating concepts and tracking down relevant entities from a text. Hyperlinked documents additionally can supply useful information about document and text structure, as well as information about the social network linked to a particular document. The latter feature is exploited extensively by current search engines in order to sort search results by the assumed relevance attributed to a page by linking to it.

Building statistical language models from plain text is a particularly interesting approach to gain information from natural language corpora because it does not require human interaction, whereas corpora with rich feature structures rely on human labour to enrich raw textual data with more intricate linguistic features.

A preliminary step to be taken before creating such models is normalising the textual data provided by the corpus. Several methods like stemming, phonetic indexing and stopword removal can be used to map word forms to their underlying lexemes or to reduce noise.

Statistical language models represent the n-gram types occurring in a corpus and their respective frequency in that corpus.

The process of creating a statistical language model has to take into consideration features of natural language data. The arguably most important feature in terms of statistics is the distribution of word frequencies according to Zipf's Law, which results in the sparse data problem. This problem has to be addressed by favouring smoothed probabilities over maximum likelihood estimates. A method that has proven to be very useful regarding the creation of accurate statistical language models is Good-Turing smoothing.

Once we have accurately set up our language model, we can use the n-gram data to conduct statistical hypothesis tests. Such tests, including the chi-square test, the t-test and log-likelihood measures, can be used to make predictions about the relatedness of words in terms of their probability to co-occur in the corpus at hand.

Building upon this data, document indexing and clustering algorithms can be used to make predictions about similarity on a more abstract level, i.e. regarding documents. Term frequencies, weighted by the inverse document frequencies of terms, lend themselves to the creation of vector-space representations of documents. These in turn can be used to determine the similarity between documents using standard operations like the calculation of the angle between two vectors.

Given a measure of similarity, documents can be clustered according to their content, which ideally leads to documents about similar topics occupying the same cluster. Clustering algorithms, for example are used to find similar documents given an example document about a specific subject.

In order to display some of the measures and algorithms described in this paper in more detail, I have developed a web application called TextSieve that allows the user to build statistical language models from plain text corpora. Furthermore, the software provides the means of hypothesis testing the relatedness of words using some of the measures described in section 3.4.

This software has been developed with previous work on the Text-NSP package in mind. Text-NSP provides a similar functionality, although being much more complete in terms of the different hypothesis testing measures, as well as a variety of special purpose options. However, Text-NSP stores its data in a text file format especially designed for Text-NSP, which has two drawbacks:

- Processing the data with external applications requires a data converter.
- All or nothing. Text-NSP deals with each and every n-gram in a language model when carrying out statistical hypothesis tests, even if the user is only interested in a subset of the n-grams

TextSieve tries to avoid this problem by using a (largely) SQL-compliant database management system. This allows external applications to access information gained by TextSieve more easily using standard SQL commands. Secondly, the user can select a subset of n-grams for statistical hypothesis testing, which, depending on the size of this subset, can reduce the processing time needed for a particular task.

Finally, two more recent issues in linguistic research have been brought up in order to underline the actual relevance of statistical hypothesis tests.

The first of these is the use of the t-test and log-likelihood measures in lexicography. Church / Hanks (1991) and Halama (2002) show how such methods may be used to find semantically related words and fixed expressions for inclusion in a lexicon.

Secondly, drawing upon the works of Baldwin et al. (2006), Himmelmann (1998) and Baayen (2001), Kiss (2006, 2007) and Dömges et al. (2007) address the problem of determinerless PPs. These structures are generally considered to be irregular and hence not determined by grammar rules. However, determinerless PPs ($P+N_{sing}$) not only seem to be productive (a feature that normally is only associated with rule-governed phenomena), but they also display statistical properties which are similar to those of clearly rule-governed structures like $P+N_{pl}$.

This brings about several questions not only about the regularity of determinerless PPs, but about the actual nature of grammar itself, since speakers of a language displaying this phenomenon usually cannot create novel expressions of this kind. If, given its statistical properties, which clearly resemble those of a productive process, this phenomenon is considered to be productive, we should assume that the cognitive productivity, that is the speaker's competence, tallies with the empirical productivity. If this is not given anymore, assumptions about the structure of grammar might need to be revised.

The authors admit that further research in this direction has to be done. Clustering algorithms might prove to be useful to aggregate words annotated with their morphological, syntactic and semantic features in order to find out which kind of nouns is especially prone to forming such structures.

Natural language corpora, statistical language models and indexing / clustering methods play a pivotal role in further examining phenomena like the ones described above, since given the enormous amount of natural language data available today on one hand lends itself to supporting or disproving theories of grammar, yet on the other hand requires the use of large-scale, efficient methods of processing and extracting useful information.

8. References

– Anderson, C. (2004): *The Long Tail*. Wired. http://www.wired.com/wired/archive/12.10/tail.html
– Baayen, R. H. (2001): *Word Frequency Distributions*. Dordrecht, NL: Kluwer Academic Publishers.
– Baeza-Yates, R. / Ribeiro-Neto, B. (1999): *Modern Information Retrieval*. Boston, MA: Addison-Wesley.
– Baldwin, T. / Beavers, J. / van der Beek, L. / Bond, F. / Flickinger, D. / Sag, I. A. (2006): *In Search of a Systematic Treatment of Determinerless PPs*. In: Patrick Saint-Dizier (ed.), *Syntax and Semantics of Prepositions*, pp. 163-179. Dordrecht: Springer.
– Banerjee, S. / Pedersen, T. (2003): *The Design, Implementation, and Use of the Ngram Statistics Package*. In: *Proceedings of the Fourth International Conference on Intelligent Text Processing and Computational Linguistics*, pp. 370-381, Mexico City.
– Bishop, C. M. (2006): *Pattern Recognition and Machine Learning*. Singapore: Springer.
– Brants, T. (2000): *TnT - A Statistical Part-of-Speech Tagger*. In: *Proceedings of the Sixth Applied Natural Language Processing Conference ANLP-2000*, pp. 224 – 231. Seattle, WA.
– Brants, T. / Franz, A., Google Machine Translation Team (2006): *All Our N-gram are Belong to You*. http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html
– Brin, S. / Page, L. (1998): *The anatomy of a large-scale hypertextual Web search engine*. In: *Proceedings of the Seventh International Conference on World Wide Web*: pp. 107-117. Amsterdam, NL: Elsevier.
– Chakrabarti, S. (2003): *Mining the Web - Discovering Knowledge from Hypertext Data*. San Francisco, CA: Morgan Kaufmann.
– Choueka, Y. (1988): *Looking for needles in a haystack or locating interesting collocational expressions in large textual databases*. In: *Proceedings of the International Conference on User-Oriented Content-Based Text and Image Handling*: pp. 609-623, Cambridge, MA.
– Church, K. W. / Hanks, P. (1991): *Word association norms, mutual information and lexicography.* In: *Computational Linguistics*, 16-1, pp. 22-29. Cambridge, MA: MIT Press.
– Dey, L. / Abulaish, M. / Sharma, J. / Sharma, G. (2007): *Text Mining through Entity-Relationship Based Information Extraction*. In: *Proceedings of 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, pp. 177-180. Fremont, CA.
– Dömges, F. / Kiss, T. / Müller, A. / Roch, C. (2007): *Measuring the Productivity of Determinerless PPs*. In: Costello, Fintan, John Kelleher and Martin Volk (Eds.): *Proceedings of the 4th ACL-SIGSEM Workshop on Prepositions*, pp. 31-37. Prague, CZ.
– Fano, R. M. (1961): *Transmission of information; a statistical theory of communications*. Ney York: MIT Press.
– Gale, W. / Sampson, G. (1995): *Good-Turing frequency estimation without tears* In: *Journal of Quantitative Linguistics 2-1995*: pp. 217-237. London: Routledge Taylor & Francis Group / International Quantitative Linguistics Association.
– Good, I. J. (1953): *The population frequencies of species and the estimation of population parameters*. In: *Biometrika 66*, pp. 393-396.
– Grefenstette, G. / Kilgarriff, A. (2003): *Introduction to the Special Issue on Web as Corpus*. In: *Computational Linguistics, 29-3.* Cambridge, MA: MIT Press.
– Halama, A. (2002): *Statistische Verfahren zur Ermittlung semantischer Nähe*. In: Busemann, S. (ed.), *Konvens 2002 Tagungsband*, pp. 45-49, Saarbrücken, Germany: DFKI.
– Himmelmann, N. (1998): *Regularity in irregularity: Article use in adpositional phrases*. In: *Linguistic Typology 2*, pp- 315-353. New York: De Gruyter.
– Jurafsky, D. / Martin, J. H. (2000): *Speech and Language Processing*. Upper Saddle River, NJ: Prentice Hall.
– Kilgarriff, A. (2005): *Language is never, ever, ever, random*. In: *Corpus Linguistics and Linguistic Theory 1-2*, pp. 263-276.
– Kiss, T. (2006): *Do we need a grammar of irregular sequences?* In: Miriam Butt (ed.), *Proceedings of KONVENS*, pp. 64-70. Konstanz.
– Kiss, T. (2007): *Produktivität und Idiomatizität von Präposition-Substantiv-Sequenzen*. In: *Zeitschrift für Sprachwissenschaft*, 26-2, pp. 317-345. Konstanz.
– Konchady, M. (2006): *Text Mining Application Programming*. Florence, KY: Thomson Learning.

- Liu, V. / Curran, J. R. (2006): *Web Text Corpus for Natural Language Processing*. In: *Proceedings of the 11th Meeting of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 233-240.
- Manning, C. D. / Schütze, H. (1999): *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.
- Pedersen, T. (1996): *Fishing for Exactness*. In: *Proceedings of the South-Central SAS Users Group Conference (SCSUG-96)*, pp. 188-200. Austin, Texas.
- Pedersen, T. / Kayaalp, M. / Bruce, R. (1996): *Significant Lexical Relationships*. In: *Proceedings of the 13th Nation Conference on Artifical Intelligence*, pp. 455-460. Menlo Park, CA.
- Porter, M. (2006): *The Porter Stemming Algorithm*. http://tartarus.org/~martin/PorterStemmer/
- Porter, Martin (2001): *Snowball: A language for stemming algorithms*. http://snowball.tartarus.org/texts/introduction.html
- Rosen, K. H. (1991): *Discrete Mathematics and Its Applications, 2nd edition*. New York: McGraw-Hill.