Morphology Mining:

Comparison Between Selecting Inflectional Classes Using Unsupervised Corpus-Compression Methods vs Googling the Solution

Bachelor's Thesis
by
Thorben Krüger
thkruege@uos.de

Cognitive Science
University of Osnabrück

November 2009

First Supervisor: Prof. Dr. Stefan Evert
Second Supervisor: Dr. Helmar Gust

Für Katha

Abstract

Unknown-word categorization remains an important topic in the field of computational linguistics, because a language constantly evolves new words (*e.g.*) to accommodate novel concepts. It is hard to cover all results of this process using just static word lists or lexicons. In the past, some work has been done to fill this gap for the German language. For this purpose, a lexicon-less finite-state morphology has been built, that can offer hypotheses about possible morphological features for a given word form.

This thesis concerns itself with the evaluation of two different approaches for the determination of the correct explanation of a word form from such a set of hypotheses.

The first approach explores the possibility of using methods and theoretical considerations derived from the minimal-description-length principle (MDL) to find a mapping between corpus word-forms and hypotheses that would allow for a maximal compression of the corpus. By Ockham's razor, this minimal mapping is expected to mostly incorporate the correct choices. However, although this effect is certainly visible in the results, it does not appear to be strong enough to suggest a practical applicability of this approach.

The second approach attempts to use the "number-of-hits feature" as returned as meta data by modern web search-engines in response to any query to gain a confidence measure on each particular hypothesis for the explanation of a word form. Even using a basic heuristic on the returned meta data to determine the most likely correct hypothesis already achieves a satisfactory level of accuracy, suggesting potential for further improvement.

This thesis concludes with remarking that while the first approach makes use of a compelling theoretical background, further investigation should concentrate on the more promising, data-mining-oriented second approach.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my superviser Stefan Evert for suggesting such interesting and hands-on topics for me to write about and Peter Adolphs for laying the foundation for this work by providing me with the necessary documentation and scripts from his diploma thesis. Without the significant support and advice I received from Katharina Wilmes, Thomas Göbel, Andreas and Detlef Krüger, this work in its current form would not have been possible.

Chapter 1

Introduction

The vocabulary of a natural language is usually not finite. Most languages readily accommodate references to novel objects or concepts, which humans happen to conceive on a very regular basis. A language usually readily accommodates the verbalization of such a novel idea by means of *open class* (or *lexical*) *words*, which in many languages are counterpart to *closed class* (or *function*) *words*.

Closed class words mainly add grammatical information, which means that they are predominantly defined by their syntactic behavior. Compared to the open class vocabulary of a language, their usage also varies very little over time. As such, closed class words are considered the core of a language. The fact that there are finitely many such words for any given language is implied by the name *closed class words*. Their inherently enumeratible nature facilitates their automated recognition and categorization even in large texts of obscure content. (Content-carrying words are usually *not* closed class words) ([van Gelderen, 2009] ,[O'Grady et al., 1997])

On the other hand, the *open class* words of many languages are usually subject to morphology-altering processes such as affixation/derivation, inflection and compounding. While for the English language these processes are still comparatively restricted to a minimum of complexity[1], matters seem to be increasingly non-trivial for many other languages.

As an inflecting language with three grammatical genders and four cases, German has a particularly complex morphology, occasionally (in the case of some adjectives) resulting in more than a dozen different possible forms for a single word belonging to an open class. As a result (*e.g.*) the automated

[1] Perhaps a reason for the wide adoption of English as the new *lingua franca*.

lemmatization[2] of German texts can be a difficult and error-prone process, especially when auxiliary information as, *e.g.*, part-of-speech (POS) tags is not present.

When viewed from a purely morphological perspective, German (open class) word forms are ambiguous: A word form could have been formed from different underlying word stems and/or a number of different morphological affixation processes.

There exist implementations of *morphological grammars* for the decomposition of word forms into stems and affixes, although they commonly rely on an existing and extensive *lexicon* for stems. In his diploma thesis, [Adolphs, 2008] adapted parts of the SMOR ([Schmid et al., 2004]) computational morphology for German to produce a system capable of hypothesizing on possible stems of a word form without having to rely on a stem lexicon. This then formed the basis for a system intended for the creation of an inflectional lexicon for German, *i.e.*, a lexicon listing stems and associating each stem with a *continuation class* denoting how this stem is to be inflected to form all its derivable word forms permitted by the (German) language.

Adolphs' hypothesizer offers a range of morphological explanations (hypotheses) for any given word form. (See Example 1.1 below for a sample output). However, in most cases, only one of these hypotheses is correct. In his work Adolphs applied several simple heuristics based on statistical data of word observations. This technique was successful only to some degree. This thesis concerns itself with exploring two additional and very different approaches for tackling this issue.

The first and more information-theoretical approach builds upon ideas from the *Minimum Description Length Principle* in order to determine the correct mappings between corpus word forms and corresponding hypotheses by creating and incrementally minimizing a model for a compressed representation of the corpus that makes use of stems and continuation classes to encode word forms using as little space as possible. As will be discussed below, this approach relies on the assumption that the smallest compression of the corpus can be achieved with a model encorporating the highest number of correct mappings between word forms and hypotheses.

[2] *lemma*: the basic form of a word, for example the singular form of a noun or the infinitive form of a verb, as it is shown at the beginning of a dictionary entry. ([Oxford University Press, 2009])

The second approach tries to address the problem using internet data-mining. It is attempted to find the correct interpretation of a word form by taking each interpretation hypothesis and generating all potential word forms describable by the hypothesized stem and continuation class before constructing and executing internet search-engine queries from these word forms. The number of returned results for each query is compared. The interpretation with more hits than another is considered to be more likely to be the correct one.

1.1 Basic Concepts

Some basic concepts and terms used in this work are outlined in this section.

1.1.1 Morphology

Inflectional/Continuation Class

A set of morphological rules to generate inflected word forms from a set of stems can be grouped under an *inflectional class* that represents the morphological features of this inflection. By being inflected, a stem usually gains a suffix, *i.e.* "is *continued*" justifying the alternative expression of *continuation class*. With knowledge about the inflectional class of the stem, all its possible inflections (*i.e. word forms*) can be inferred and generated. For the purposes of this thesis, an inflectional class is represented as a string enclosed by angular brackets, *e.g.* `<VVReg>`. This notation for inflectional classes is shared by all morphological systems used in this thesis.

Morphological Hypothesizer

A *morphological hypothesizer* as used in this work accepts a (fully inflected) word form as input and generates a set of *hypotheses* (*i.e.* possible morphological interpretations), each consisting of a stem and associated inflectional class. (See Example 1.1 below.) The input word form could have been constructed using any of the resulting hypotheses.

(1.1) Set of possible morphological hypotheses for the word "fangen":
`<ge>fang<VVReg>`, `<ge>fangen<VVReg>`, `fang<VVReg>`, `fangen<VVReg>`, `fangen<Adj+>`, `fangen<Adj+e>`, `fangen<Adj+(e)>`, `fange<Adj+>`, `fange<Adj+e>`, `fange<Adj+(e)>`, `fang<Adj+>`, `fang<Adj+e>`, `fang<Adj+(e)>`

1.1.2 Compression

Minimum Description Length

In general, the minimum description length (MDL) principle can be used as a method for the selection of an explanatory model from a set of competing models that has the best chances for describing a set of data.

One of the core assumptions of this approach is that by conducting statistical inference, one primarily attempts to find regularities in the given data. Once such regularities have been identified, they enable us to compress the data by means of refering to inferred rules that govern these regularities. In this vein, learning, *i.e.*, recognizing patterns, can be viewed as data compression: the more one is able to compress information (generalize), the more one has learned.

MDL realizes this insight by choosing the hypothesis or a selection of hypotheses from the set of all hypotheses H as best describing data set D, if their application compresses D most.

Chapter 2

Unsupervised Maximization of Corpus Compression

This chapter discusses theory and practice of automatically finding the highest number of correct mappings of word forms to their respective stem and inflectional continuation class by means of finding a maximal compression of the corpus containing the word forms. Since the compressed corpus data is strongly associated with the model that facilitaed the compression, this is sometimes called *two-level* coding.

2.1 Previous Work

There are numerous previous works regarding the automated analysis of word forms, some of which take two-level approaches similar to the one that is presented here. In his dissertation, [Koskenniemi, 1983] is apparently the first to give an account of this approach for the morphological analysis of highly inflecting languages such as Finnish or Russian. My work is particularly influenced by [Goldsmith, 2004] and [Goldsmith, 2001] on applying MDL principles to unsupervised learning of natural language morphology. In case of the model design, I heavily borrow from [Goldsmith, 2004].

[Grünwald, 2007] is a general and very readable book about the Minimum Description Length principle, while [Grünwald, 2004] offers a shorter introductory tutorial on the matter. Grünwald's work in general covers the essential theoretical background information needed for this approach.

Finally, [Vitanyi and Li, 1999] point out that data compression is "almost always the best strategy, both in hypothesis identification and prediction".

2.2 Theory of Corpus Compression

I am interested in finding the correct morphological continuation classes for a set of stems. In order to do so, I leave (most) linguistic considerations aside in order to simply exploit the effects predicted by Ockham's Razor: The best (correct) morphology, when used for the compression of the corpus, will also lead to the best (smallest) results. If this premise is correct, then by searching for a the highest compression I should find models of increasing accuracy.

2.2.1 Data Compression

In general terms, data compression minimizes a data set by identifying and replacing every repeated instance of a reoccurring segment with a (potentially shorter) reference (or *pointer*) to a single copy of this segment.

This raises the question of how to best chose short pointers as replacements for every occurrence of a redundancy within a text. The answer lies in variable-length coding as employed in *Huffman trees* for instance. Huffman coding is a mechanism that determines a unique bit sequence for an element, based (for example) on the element's frequency within a data set. This "entropy encoding" technique assigns shorter bit sequences to more frequent elements and longer sequences for the rarer ones. An additional property of the code sequences is that they are "prefix-free", that is, the bit sequence representing some particular element is never a prefix of the bit sequence representing any other element.[1] This is a consequence that stems from the underlying concept of a *binary-branching tree* that has both internal and leaf nodes: The two successor nodes of any internal node are "addressed" using bits 0 and 1 respectively. A leaf node "stores" a particular element and traversing the tree from the root node to this leaf node traces a unique path over the "addresses" of the intermediate nodes. This path corresponds to the resulting code for the stored element. Because there are no successor nodes to a leaf node, it becomes clear why any code can never be the prefix to a longer one. It follows that such codes can be concatenated without losing information. To compress a data set, one replaces each element with its corresponding (nearly) entropy-optimal code, which is obtained when entering the element into the binary tree. (For details on the algorithm that generates these codes, see [Huffman, 1952].) Any redundancies in the set of elements

[1] Perhaps confusingly, the term "prefix code" often refers to this very property as well.

from the data set thereby get deflated to the length of its corresponding code, which can result in an overall shorter representation.

(The important question of how to best segment the input data in order to find the best compression ratio is not explored here, because the data I am going to compress will be segmented in a very well-defined way. More on that later.)

2.2.2 Representation of Word Forms

C	the corpus, a (finite) stream of text
W	the ordered set of different words in C
w_n	the nth word in W
f_n	the frequency of w_n in C
H_n	the ordered set of morphological hypotheses for w_n
h_{nk}	the kth hypothesis in H_n
s_{nk}	the stem hypothesis of h_{nk}
c_{nk}	the class hypothesis of h_{nk}
P_{nk}	the ordered set of word forms predicted by h_{nk}
p_i	the ith predicted word form in P_{nk}
$HYP(w_n)$	the function that generates H_n from w_n
$GEN(h_{nk})$	the function that generates P_{nk} from h_{nk}

Table 2.1: Overview of Notation

Let W be an ordered[2] list of all *word forms* from a corpus C. Then then the nth word form in W is denoted by w_n. The frequency of w_n in C is f_n. Let the system that produces stem and class hypotheses for a word form be represented by the function HYP. When using w_n as an argument to the function, it returns the corresponding set of morphological hypotheses denoted by H_n. Each hypothesis $h_{nk} \in H_n$ consists of a stem s_{nk} and a morphological continuation class c_{nk}. A hypothesis h_{nk} can be used with the function $GEN(h_{nk})$ (*i.e.*, the inverse of HYP) to generate the ordered set P_{nk} containing all potential word forms that h_{nk} describes.

Example 2.1 shows a sequence of words that represents a toy corpus C.

(2.1) "Der dicke Navigator repariert den Kompressor des dünnen Navigators"

The corresponding ordered set W of open class words in C is shown in Example 2.2.

[2] *i.e.*, deterministically sorted according to a pattern

(2.2) “dicke”, “dünnen”, “Kompressor”, “Navigator”, “Navigators”, “repariert”

Example 2.3 shows part of each set of possible hypotheses H_n as generated by the function $HYP(w_n)$ for each $w_n \in W$.

(2.3)

Word Form	Hypotheses
“dicke”	`dicke<Adj+>, dick<Adj+>, dick<Adj+e>, ...`
“dünnen”	`dünnen<Adj+e>, dünne<Adj+e>, dünn<Adj+e>, ...`
“Kompressor”	`Kompressor<NNeut_s_s>, Kompressor<NMasc_s_en>,...`
“Navigator”	`Navigator<NFem_0_s>, Navigator<NMasc_s_en>,...`
“Navigators”	`Navigator<NFem_0_s>, Navigator<NMasc_s_en>,...`
“repariert”	`reparier<VVReg>, repariert<Adj+>, ...`

Selecting one hypothesis (h_{nk}) for every $w_n \in W$ might result in choices as displayed in Example 2.4. For demonstration purposes, two different hypotheses for “Kompressor” and “Kompressors” were selected.

(2.4) `dick<Adj+e>, dünn<Adj+e>, Kompressor<NMasc_s_en>,`
`Navigator<NMasc_s_en>, Navigator<NFem_0_s>, reparier<VVReg>`

Example 2.5 lists the ordered set of word forms P_{nk} predicted by the GEN function for each h_{nk}. The observed word form w_n among each P_{nk} is emphasized for convenience.

(2.5)

Hypothesis	Set of Predicted Word Forms
`dick<Adj+e>`	"dick", *"dicke"*, "dicken", "dicker", "dickere", "dickeren", "dickerer", "dickerem", "dickeres", "dickem", "dickes", "dickest", "dickeste", "dickesten", "dickester", "dickestem", "dickestes"
`dünn<Adj+e>`	"dünn", "dünne", *"dünnen"*, "dünner", "dünnere", "dünneren", "dünnerer", "dünnerem", "dünneres", "dünnem", "dünnes", "dünnest", "dünneste", "dünnesten", "dünnester", "dünnestem", "dünnestes"
`Kompressor<NMasc_s_en>`	*"Kompressor"*, "Kompressoren", "Kompressors"
`Navigator<NMasc_s_en>`	*"Navigator"*, "Navigatoren", "Navigators"
`Navigator<NFem_0_s>`	"Navigator", *"Navigators"*
`reparier<VVReg>`	"reparier", "repariere", "reparieren", "reparierend", "reparieret", "reparierest", *"repariert"*, "reparierte", "reparierten", "repariertet", "repariertest", "reparierst"

Because GEN is the exact reverse of HYP and the function HYP and GEN are deterministic (*i.e.*, given the same input they will always return the same results in the same order), the following condition holds for all cases: for all $w_n \in W$ and for all $h_{nk} \in H_n$ there exists a $p_i \in P_{nk}$ such that $p_i = w_n$. This index i of the corpus word form w_n in P_{nk} is a crucial part in the compression of the corpus: To encode w_n using my model, I first chose a hypothesis h_{nk} (from the set of all hypotheses H_n that can describe w_n). Applying h_{nk} to the GEN function produces the ordered set P_{nk} of all word forms that are predicted by h_{nk}. The original word form w_n is guaranteed to be among the members of P_{nk}. Since P_{nk} is an ordered set, I can uniquely refer to any of its members using indices. As a result and using the GEN function, I can encode w_n in terms of h_{nk} and the unique index i for the predicted word form $p_i \in P_{nk}$ where $p_i = w_n$. In short: $w_n = GEN(h_{nk})_i$.

Since the sets of predicted word forms are always ordered in the same way, I can simply refer to their members via indices. The second member p_2 of the set P returned by GEN(`dick<Adj+e>`) is always "dicke". This justifies

referring to "dicke" simply by stating $GEN(\texttt{dick<Adj+e>})_2$. Example 2.6 shows the results of applying this technique to every $w_n \in W$ from Example 2.2.

(2.6) $GEN(\texttt{dick<Adj+e>})_2$, $GEN(\texttt{dünn<Adj+e>})_3$,
$GEN(\texttt{Kompressor<NMasc\_s\_en>})_1$,
$GEN(\texttt{Navigator<NMasc\_s\_en>})_1$, $GEN(\texttt{Navigator<NFem\_0\_s>})_2$,
$GEN(\texttt{reparier<VVReg>})_7$

This way of encoding word forms exposes redundancies on several well defined levels which can be reduced. As already mentioned, a hypothesis h has a stem part (s) and a class part (c). So for instance, although there are 6 different word forms in W (Example 2.2), only 5 different stems are needed to encode them ("Navigator" occurring twice). In Example 2.6, only 4 different classes are observed (<Nmasc_s_en> and <Adj+e> occuring twice each).

If I now replace every occurrence of any ubiquitous element (of whatever kind) with a shorter pointer to a single instance of this element, I have successfully achieved some compression of the data. To generate suitable pointers from a set of elements, I can make use of the Huffman coding algorithm that to each element assigns a pointer of a length close[3] to the negative base 2 logarithm of the elements relative frequency (in the set of all elements). The latter is called the *weight* of the element.

I am going to encode each w_n in terms of c_{nk}, s_{nk} and i so each is going to be compressible. The model for encoding word forms will be structured like this:

- globally, stems (s) are stored in a Huffman tree according to/weighted by the relative frequency with which each is used for explaining a corpus word form.
- class structures are stored in a Huffman tree according to/weighted by the relative frequency with which each is used for explaining a corpus word form.
- a class structure c consists of an internal Huffman tree, storing the (global) code of each stem (plus a Huffman tree for indices, see below) that appears in conjunction with the class, weighted by the relative frequency the stem *and* the class are used for explaining a corpus word

[3]For reasons of simplicity, we will use this approximation for all applicable future purposes.

form (which can obviously be different from the weighting of the global tree).

What remains to be encoded to compress the corpus is the index i selecting the one word form $p_i \in P_{xk}$ that is equal to the w_x that is intended to be described by the current hypothesis h_{xk}. This is entered into the Huffman tree simply by using f_x as weight. Such a tree is created and stored with each stem s_x inside a class structure.

Huffman codes can simply be concatenated due to the fact that a complete code for one entry can not simultaneously be an incomplete code for another entry. Hence all potential boundaries are always clear-cut and a word form w_n can be encoded by the Huffman code for the class c_{nk} followed by the class-internal code for s_{nk} in turn followed by the stem-specific code for index i.

In reverse, to determine the encoded word, a code sequence can be processed bit by bit until an entry c is found in the Huffman tree for class structures. From the next bit on, the Huffman tree for stems inside the found class is crawled until a stem s is found. The final bits are used with the tree stored with the stem to look up the index i of the intended word form in the results generated by the GEN function, which is called with the concatenation h of class c and stem t: $w = GEN(h)_i = GEN(c \circ s)_i$.

2.2.3 Storing the Corpus

Every single word in the corpus can be replaced with a code sequence following this pattern, shrinking the compression size at the cost of introducing complexity into the generating model.

2.2.4 Storing the Model

Having compressed the corpus by introducing a lot of complexity into the model I now have to deal with storing the model in a minimal manner. In particular, I need to discuss storing

1. strings of characters (words)[4]
2. Huffman trees / lookup tables
3. a finite state morphology

[4]See Section 2.3

efficiently. The last point from this list is not only beyond the scope of this work but can also be safely disregarded due to the fact that the HYP and GEN functions representing the morphology remain unchanged between different models. The same is obviously true for the algorithms used for constructing (and working with) the computational equivalents of strings, tables or trees from their respective minimal representations which I am going to discuss next.

For reasons of simplicity the space complexity of Huffman trees is considered to be proportional to that of an equivalent lookup table. Due to the mentioned deterministic nature of a Huffman code, a simple list structure is sufficient. (*i.e.* it is not necessary to distinguish between key and value field.) To simplify further, every entry of such a list is prefixed by its bit length. The length of the binary coding of the longest entry bit length is stored at the beginning of the list in a constant number of bits. The bit length prefix for every list entry is of this predefined size. The first part after the prefix is the Huffman code. In case of a mismatch, the lookup algorithm would skip ahead in the bit stream by the amount defined by the prefix (minus the bits parsed beyond the prefix) to get to the next entry. The bit length of such a lookup table would be the prefix bit length times the number of entries (Huffman code plus node content) plus the sum of the bit lengths of all entries plus the constant at the beginning of the list denoting the prefix size.[5] Given similarly sized entries the list size grows proportional to its length.

My model consists of

1. a Huffman coded list of stems,
2. a Huffman coded list of morphological continuation classes,
3. a (constant) finite state morphology,

where I ignore the last part as already mentioned. The Huffman coding of stems is done the way it was described above. The coding of morphological continuation classes is a bit more involved as each node contains a Huffman coded list of data structures consisting of stem plus another Huffman coded list of indices for the desired GEN interpretations.

2.2.5 Summary

In this section, I have tried to provide a sound and solid theoretical foundation for an *à posteriori* corpus compression by means of exploiting an existing

[5] See Section 2.3 below for an example

morphological model (*i.e.* set of morphological facts). Although I have now identified the required steps to compress the corpus in some detail, I will not bother with actually implementing them. For the purposes of this work, it is merely sufficient to know the *would be* compression size of the corpus for any given model. Hence, I will later on be able to make do with reasonable abstractions for space complexities of the algorithms I am incorporating in my theoretical considerations.

2.3 Practical Considerations

As already mentioned, I do not strive to for an actual maximal corpus compression, but for the particular set of morphological hypotheses that would make such a compression maximal. The whole purpose of calculating the code length of a corpus compression is the *comparison* with the code length of a compression that incorporates a slightly different morphological mapping.

This allows me to apply a set of abstractions, alterations and approximations to the above described methods in order to reduce their computational (and implementational) complexity.

2.3.1 Approximations

Although the code length for strings in the model could be based on the frequency-weighted Huffman codes of their characters, a reasonable approximation (according to [Goldsmith, 2004]) can be made by assuming equal probability for all characters and simply use the product of the resulting character code length and the length of the string. In this case, a string's bit length is roughly equivalent to $(n+1) \cdot \log_2(|\Sigma|+1)$ where $|\Sigma|$ is the number of characters in the alphabet and n the number of characters in the string. Adding 1 to n and $|\Sigma|$ respectively pays duty to the special string stop symbol which is outside of the alphabet. For the German alphabet with 3 additional umlauts plus the ligature "ß" and the stop symbol, $\log_2(31)$ approaches 5,[6] which is therefore used forthwith as the approximate code length for any alphabetic character.[7]

[6] Captalization is left out of consideration in this approach.

[7] My implementation can optionally use accurate string code lengths based on Huffman code lengths of characters instead of this approximation.

2.3.2 Simplifying Assumptions and Abstractions

A major potential for simplification is the above stated fact that I only compare code lengths between compressions. Additionally, only the fact that one compression is either shorter than, equal to or longer than another is going to be considered. This enables me to introduce simplifications that keep the simplified code length merely *proportional* to the original one. Also, any invariant parts such as static data structure overhead of the actual compression can be left out. I already briefly argued the case for leaving the code length of the morphological hypothesizer out of even the detailed, theoretical compression, and the same applies here. In the following an overview over most of the simplifications is given.

Disregarding Closed Class and Irregular Words

Irregular words by definition form exceptions from the common morphological rules[8] and as such form a *finite* set of words together with the closed class function words. These words can be assumed to be simply enumerated in a static part, either of the morphological hypothesizer or the compression itself. As such, they can be saveloy disregarded from the simplified code length calculation.

Huffman Tree and Simple List Code Lengths

The theoretical code length of a crude simple list data structure capable of representing Huffman trees was previously examined. It was observed that a code length of a simple list is largely proportional to the number n of its entries $[E_1, E_2, ..., E_n]$ plus the code lengths of the entries themselves ($\sum_{i=1}^{n} |E_i|$) plus some constant overhead c (for *e.g.* data structure containment). For each entry, a constant overhead of length l is required to encode its length. These constants need to be in $\mathbb{N}\backslash\{0\}$ to influence the outcome of comparisons between list lengths in the desired manner.

Word Interpretation

A given word form (although possibly occurring at different places all over the corpus) is assumed to only *one* correct morphological interpretation, *i.e.* to always be of the same inflectional class. Although there are certainly examples for conflicting classes for the same word form,[9] their infrequent

[8] Although their morphology might well adhere to a more complex set of rules, this possibility is not explored here.

[9] There is for instance the German noun "Leiter" which in feminine form translates to "ladder" and in masculine form means "leader". Other examples include "See" and "Tau".

occurrence does not justify the amount of additional work necessary to incorporate this kind of exception into the compression model.

Subset of Corpus

As will become clear in the next section, I dealing with subsets of the corpus significantly reduces training runtime. We are only dealing with *continuation classes* that should not have any influence on the first letters of any predicted word forms. This justifies the assumption that updates on the model (*i.e.*, identifying a better hypothesis for a given word form) can only significantly influence the corpus or model-compression properties of *related* word forms. Restricting the corpus to a subset of (*e.g.*) word forms that all begin with the same letter can be expected to leave these kinds of interactions largely intact while at the same time prevents combinatorial explosions that lead to intolerable training runtimes.

2.4 Methodology

The modified SMOR morphology is applied to every *word form* in the (filtered) corpus in order to obtain a hypothesis for the *stem* and associated *continuation class* producing the respective word form. Of the usually multiple hypotheses which the morphology (*i.e.* the implementation of the HYP function) produces for a single word form, (only) one is selected, while the remainder is kept in memory as possible future alternatives to the currently chosen hypothesis. If the continuation class of the selected hypothesis has not been entered into the model before now, it is then used to create a new *class structure* (or just *class*), wherein (a pointer to) its associated stem is entered. If a class structure for the given continuation class had already existed, the whole process would have been reduced to simply adding the stem to the class structure. The word form causing this creation/modification of a class structure is entered along with its corpus frequency into a temporary list which is associated with its stem entry in the class structure.

After the first pass, when all class structures have been created and entered into the model, the GEN function (*i.e.* the SMOR morphology in reverse) is applied to every stem and continuation class of each class structure, generating all word forms their combination potentially describes. This resulting list of potential word forms is searched for the actual observed word forms from the corpus to determine their *indices* therein. This is done in order to be able to replace every word form in the corpus with a matching compressed coding for its stem and continuation class plus index.

2.4.1 Procedure

The raw text from the German TIGER corpus is sieved for open-class words using the morphisto "gold standard" reference morphology. Only verbs, common nouns and adjectives were kept, although a less restrictive selection procedure could be easily implemented. In the following, this subset of tokens from the TIGER corpus will just be referred to as the *corpus*.

A simple inflectional morphology is then used to generate a set of *hypotheses* for the correct morphological analysis for each token in the corpus. A hypothesis is here treated as consisting of two parts:

1. the stem of the word (will be referred to as *stem* further on)
2. one or more tags with morphological information (will be referred to as *tag string* further on)

I want to determine which hypothesis is the right one for a given word.

It should be noted that the approach used here is substantially different than the one used by [Goldsmith, 2001]. Goldsmith bootstrapped a morphology concentrating on suffixation with categorization as a welcome side effect. In contrast, this work concentrates on the correct selection of the best hypothesis for a given word, namely that which successfully describes the word while only minimally extending the combined length of compressed model and corpus. While the internal structure of the model is heavily influenced by Goldsmith's work, the mathematical considerations had to be made largely independent from his.

Algorithmic Work Flow

1. Decide which word classes the model should be able to describe *(Example: Nouns, Verbs, Adjectives)*
2. Take a corpus and apply the evaluation morphology to each token. Only keep those tokens which have an analysis that the HYP function is also able to produce. (*e.g.* throw away all closed-class stop words.) 100% accuracy will be achieved, if the model produces the same analysis as the gold-standard morphology for each word.
3. For each of these (remaining) tokens, generate hypotheses using the HYP function. Randomly choose one of these hypotheses and enter it into the model. The model is now randomly initialized.

4. Calculate the compression length of this random model and the length of the corpus encoding.

5. For each token in the corpus (taken from a randomized list of all tokens) (*alternatively:* For each random token from the corpus)

   (a) remove its currently associated hypothesis from the model

   (b) select a different hypothesis, enter it into the model and recalculate the size of the compression

   (c) if the description length improved, the corresponding hypothesis replaces the previous one

   (d) repeat until all possible hypotheses for a given token have been analysed for improvement of the description length

   (e) enter the so determined "best" hypothesis into the model

6. Repeat this process until no further improvement of the model is achieved (over a certain number of iterations).

2.4.2 System Features

The system I implemented has a range of parameters that can be tuned.

2.4.3 Dynamic Programming

So far, the process of compression minimization has not been discussed, although this approach heavily relies on the effectiveness and speed of alternative hypothesis testing. It is inconceivable to completely recalculate the compression size for every change that is done to the model. Consequently, some dynamic-programming techniques have been identified that reduce the computational complexity somewhat.

Huffman Trees

Since I use the concept of a Huffman tree only to have a plausible measure of code lengths, I can replace its implementation by a simple algorithm that approximates its theoretical bit length based on the elements that would be stored in it. The size of the *prefix codes* in a Huffman tree is generally close to *optimal entropy encoding* and, although there are exceptions, I use the *information content* (in bits) given by $-\log_2(P(x))$ (as defined by [Shannon, 1948]) as a measure to approximate the code length of each element stored in the tree.

Calculating (or approximating) the bit length of the entire Huffman tree is, unfortunately, hard without addressing the task numerically. One has to calculate the information content for each element in the tree (and add the bit lengths of the stored elements themselves) in order to create a combined bit length which approximates the size of the tree. Whenever an element is added to or removed from the tree, the information content usually changes for most of the stored elements, essentially forcing a complete recalculation of the bit lengths.

Perhaps conveniently, the *worst case* code length for any element in a Huffman tree with n entries is given by $\log_2(n)$, which allows for the computation of the combined bit length of all the prefix codes in the tree without the need to iterate over every node. Here, adding a new entry to a tree is a simple matter of adding 1 to the overall number of elements n and adding the bit length of the entry to the accumulated sum of all bit lengths. Correspondingly, removing of elements from the tree does the reverse. While this approach has a number of issues with the applicability of the general theory, it was nevertheless evaluated along with the previous, computationally more expensive but also theoretically more justified approach above.

For both approaches, I have to allow adding elements to the tree that are already present. This is needed for cases wherever the probability mass of an entry gets increased in the model. This does not change the overall number of elements but decreases the Shannon entropy (and thereby the code length) of the entry in question. Conversely, the reduction of probability mass for an entry is also accounted for.

While these techniques in themselves are not considered to be cases of "dynamic programming", they are essential in facilitating the quick computation of the overall bit length of a tree, including that of the elements stored in it. Both approaches rely on accumulatively storing the sum of the individual bit lengths of elements as they are entered into the tree. When computing the bit length of the tree, this figure is simply added to the combined code lengths and to the previously mentioned overhead of the list structure that stores the tree.

Updating the Model

Updating the model is mainly done by adding or removing hypotheses, which translates to adding or removing elements from the underlying Huffman trees, which can be computationally cheap in case the above mentioned "worst-case approximation" is used.

Updating the Compression

There is no apparent way to approximate the size of the resulting corpus compression by any means that are faster than summing over the model-based bit lengths of all word forms in the corpus.

2.5 Results

As has been argued in Section 2.3.2, it is permissible to reduce the corpus to a subset that is expected to largely capture interrelated word forms. For all of the following results, training was done on a subset of the corpus capturing all word forms beginning with the letter *"S"* (both upper and lower case). When the TIGER corpus is sieved for those word forms that appear in the gold-standard lexicon, 1176 unique word forms are left to train on.

The system choses a pseudo-random "update candidate" from all alternatives for every single iteration. Each iteration takes about 0.18 seconds to compute, resulting in a processing time of 30-40 minutes[10] for each of the following plots.

To determine the theoretical upper bound of accuracy for the system, some training runs were performed on models that had been initialized with the correct mappings according to the gold-standard lexicon. Results can be seen in Figures 2.2 and 2.4.

Two different approaches for Huffman-tree length-calculation were tested. The more accurate but computationally more complex version was used for Figures 2.1 and 2.2. Figures 2.3 and 2.4 show the simpler solution based on the trivial worst-case approximation. Contrary to expectations, no significant differences in overall run times were observed. The pseudo-random number generator used is deterministic over invocations allowing comparisons such as Figure 2.5 or 2.6.

Figures 2.7 and 2.8 each explore a naïve "one-level" approach, just minimizing either the corpus compression or the model compression, while still tracking the "would be" compression size of the "other part" respectively. The final Figure 2.9 compares the compression size relations between the minimized and the complementary level for both runs.

Note that these results will only be discussed in Chapter 4

[10] Measured on a 2.4 GHz `x86_64` CPU with native 64 bit operating system.

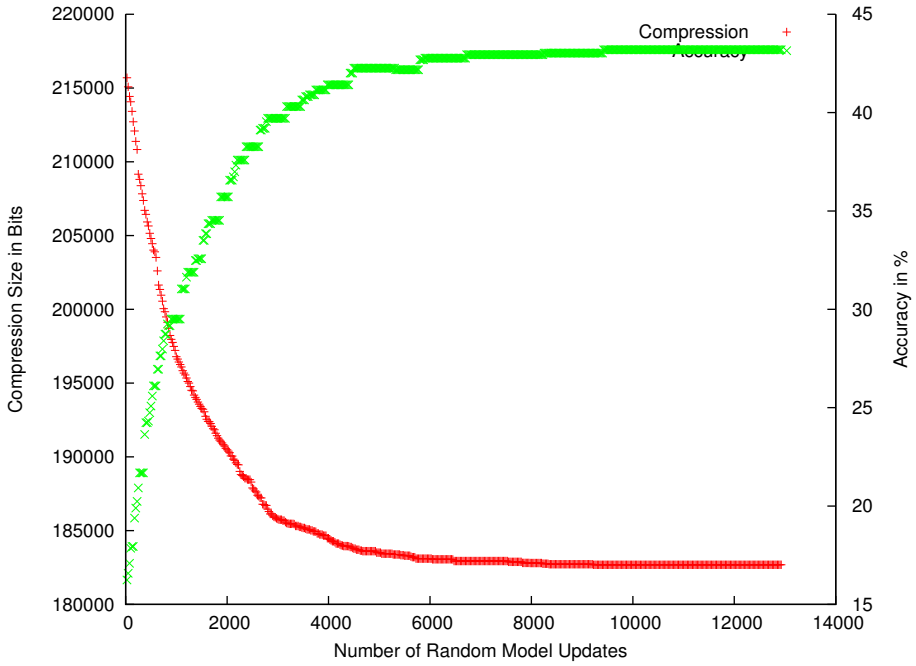

Figure 2.1: Model training run (until termination) using (slow) accurate Huffman-tree length-calculation. Plot shows *compression bit-length* versus *gold-standard accuracy* for randomly initialized model

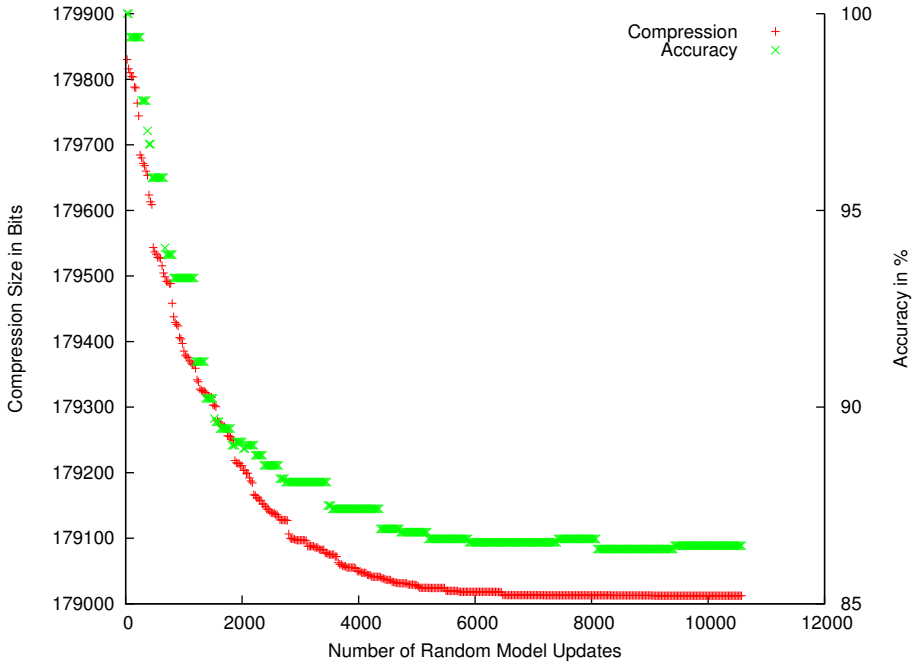

Figure 2.2: Model training run (until termination) using (slow) accurate Huffman-tree length-calculation. Plot shows *compression bit-length* versus *gold-standard accuracy* for gold standard initialized model

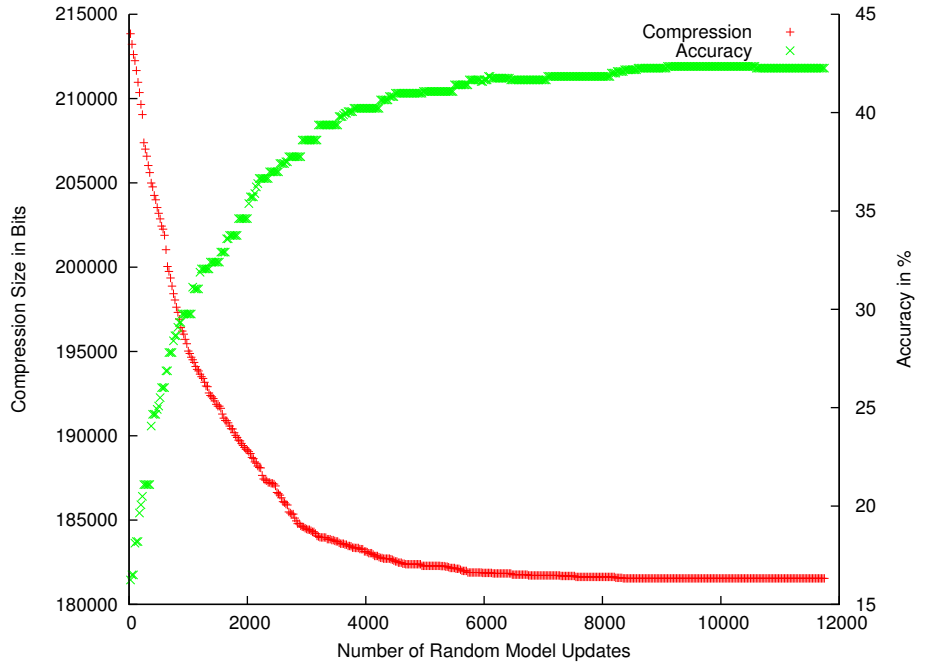

Figure 2.3: Model training run (until termination) using (fast) worst-case Huffman-tree length-calculation. Plot shows *compression bit-length* versus *gold-standard accuracy* for randomly initialized model

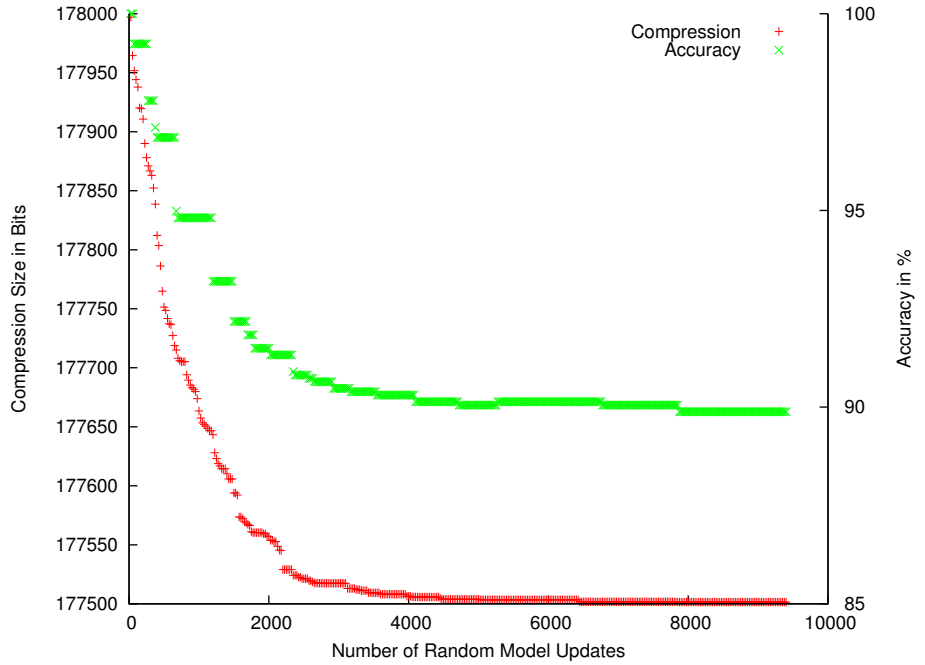

Figure 2.4: Model training run (until termination) using (fast) worst-case Huffman-tree length-calculation. Plot shows *compression bit-length* versus *gold-standard accuracy* for gold standard initialized model

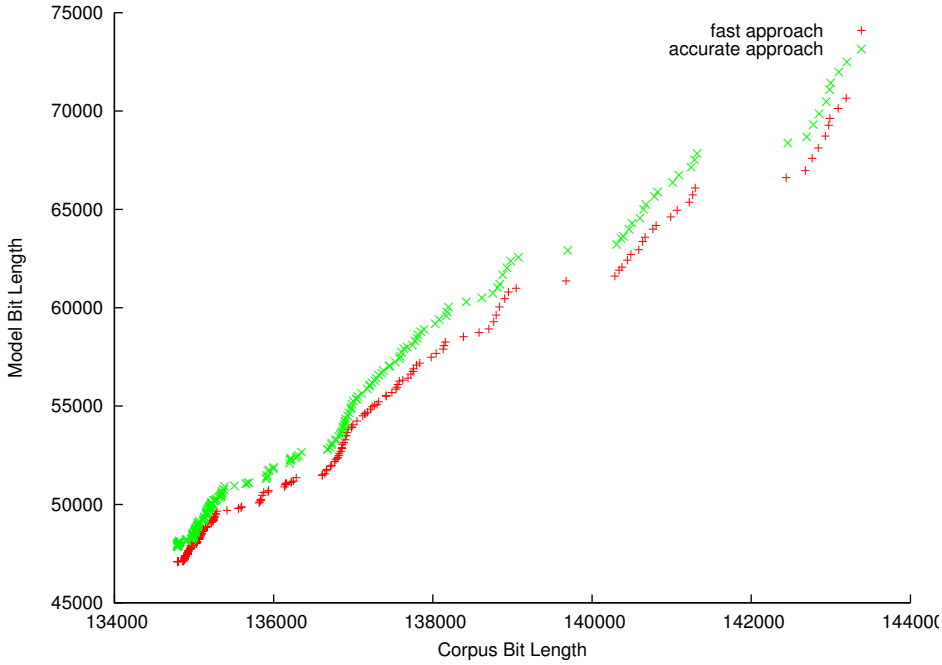

Figure 2.5: *Corpus-compression size* versus *model-compression size* for both accurate and fast approach to Huffman-tree size-approximation for randomly initialized models

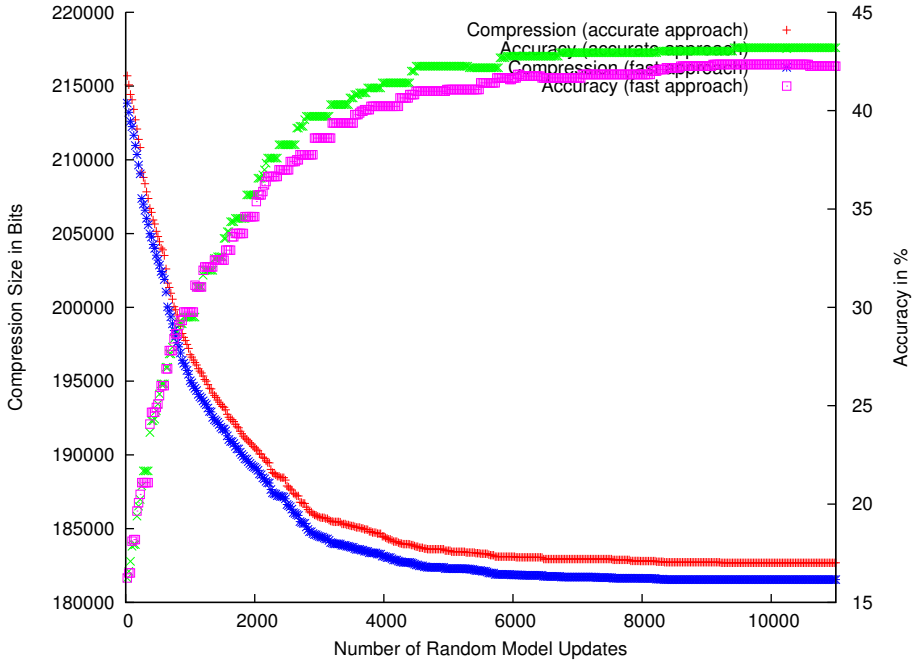

Figure 2.6: Comparison of model training runs using both approaches to Huffman-tree length-approximation. Plot shows *compression bit-length* versus *gold-standard accuracy* for randomly initialized model

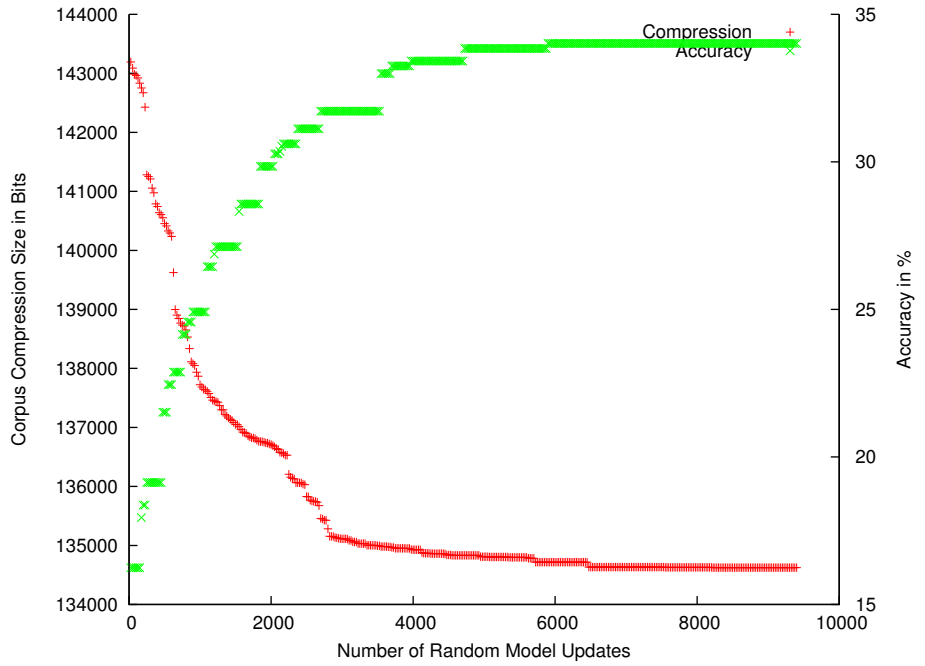

Figure 2.7: Naïve approach that minimizes only corpus compression for accurate Huffman-tree length-calculation and gold standard initialized model.

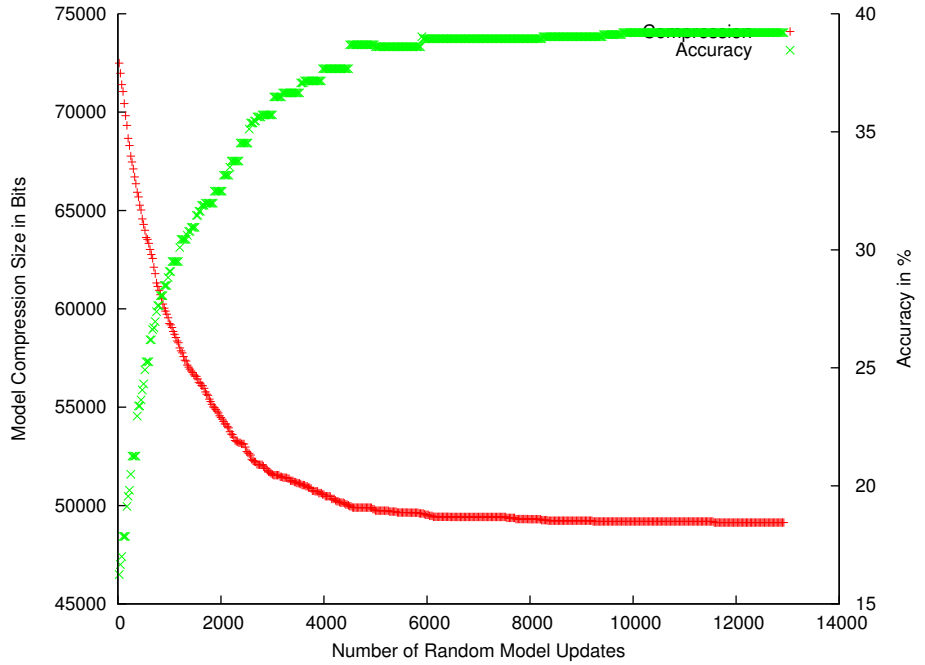

Figure 2.8: Naïve approach that minimizes only model compression for accurate Huffman-tree length-calculation and gold standard initialized model.

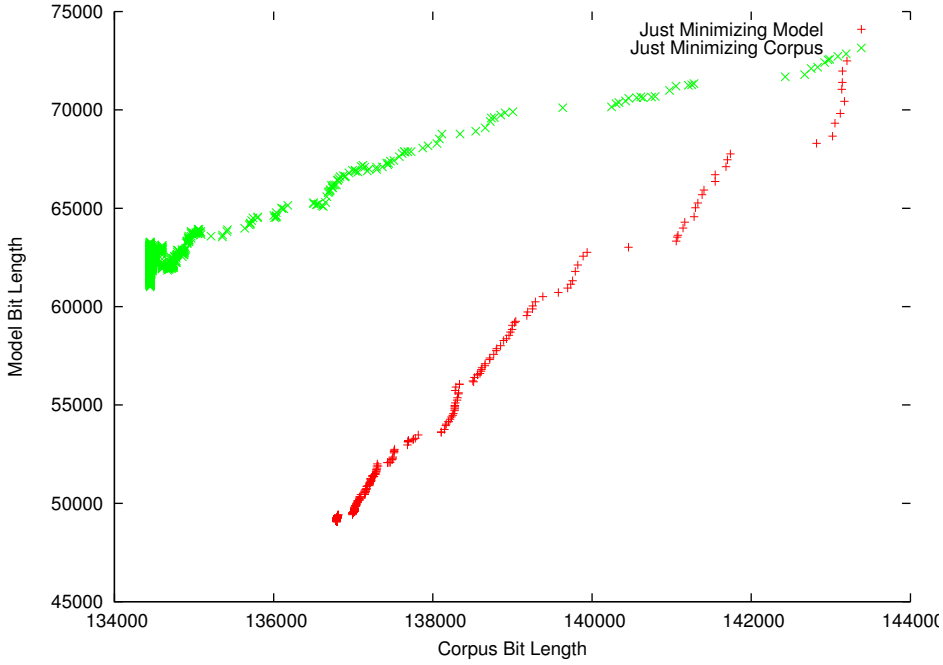

Figure 2.9: Comparison compression-size levels between both naïve approaches.

Chapter 3

The Web-Search-Hits Approach

One of the uses of the Minimum Description Length (MDL) approach as detailed above is to help with the categorization of any unknown word from a text into its correct morphological class by means of entering each potential hypothesis into the model and selecting that which makes the model grow least.
This very same task can also be approached from an entirely different perspective.

3.1 Introduction

Web search-engines such as *Google* or *Yahoo! Search* create and maintain indices of a major part of the natural language content on the World Wide Web ([Gulli and Signorini, 2005]). Performing a web search on a word (or term) by dispatching a query (containing the word) to one of those search engines typically takes a fraction of a second. As a result, part of the (ranked) list of URLs to web sites (and matching excerpts of their content) that match the query in some way is provided, usually along with some meta data. Among this meta data, search engines usually also provide a measure called "number of hits" (NoH), the (estimated) number of all (indexed) web sites that match the query. Faced with two competing concepts, dispatching a query on each to compare the NoH to support or discredit one has reportedly been used even in scientific disputes ([Kilgarriff, 2007]). In this chapter, I examine the applicability of NoH measures (as returned by popular search engines) in response to a set of automatically generated queries to determine the correct morphological interpretation of an unknown word form.

It must be mentioned that the methodology presented here is by no means

uncontrovercial. [Kilgarriff, 2007] objects to the usage of the NoH measure as a scientific resource in general, although part of his argument is rendered moot by the introduction of new APIs by the major search engines that do not impose the criticized restrictions on the number of queries per day. His criticism of the closed and commercial nature of the popular search engines however is also highly relevant to this work.[1]

I compare the accuracy of this approach for the analysing unknown word forms encountered in a raw[2] German corpus using *Microsoft Bing* (called "Bing" in the following), *Yahoo! Search* (called "Yahoo" in the following) and *Google* (as of 2009 the three most popular search engines). I also discuss some legal aspects of this usage of search engines in Appendix B.

3.2 Approach

For an observed word, a morphological hypothesizer generates hypotheses about the underlying word stem and inflectional class, for which the observed word is one of the possible forms derivable from the hypothesized stem and inflectional paradigm. The hypothesis is a correct morphological explanation of the observed word if and only if every other word form that the hypothesis allows is also valid in the language. A low NoH when querying search engines for any of these "predicted" word forms discredits the corresponding hypotheses.

The same morphological analyzer (called "morphology" in the following) as described previously is used on an unknown word to generate morphological hypotheses that are applied to the reverse of the analyzer to generate the set of word forms "predicted" by each hypothesis. These predicted word forms are then processed into search engine queries. By comparing the returned NoHs associated with each hypothesis, one expects to find the correct one with reasonable accuracy, since non-correct hypotheses are expected to predict (or "describe") non-existing word forms for which the NoH is expected to be generally low. As a crude heuristic, a low or zero NoH for just one word form would strongly discredit the hypothesis, while a constantly high NoH among all its predicted word forms would support it.

Traditionally, search engines perform character/string matching between query and website content to determine its relevance ([Brin and Page, 1998]). As a logical consequence of this, typographical errors in a query lead to a

[1] See Appendix B

[2] While annotations exist for the corpus, no use is made of them.

much smaller result set, mainly consisting of pages where the same error was made. While the result set can contain results based on informed automated guesses, it is possible to enforce the exact matching of the query to the content of the returned websites by placing quotation marks around the query ([Long et al., 2004]). This behaviour is preferred, because many queries can be expected to be "near misses" (*i.e.* by missing a single character) to the correct form and I do not want any search engine heuristics to influence the NoH.

3.2.1 Dealing with Word Form Ambiguity

Although incorrect hypotheses often predict word forms that are illegal in the language and thus have a very low (or zero) NoH, two hypotheses could also predict the same set of word forms (see Example 3.1 below) while different grammatical features for the word forms are implied by their respective morphological classes.

(3.1) `Schere<NMasc_n_n>` describes: "Schere", "Scheren", implies *male gender*
`Schere<NFem_0_n>` describes: "Schere", "Scheren", implies *female gender*

It is therefore desirable to somehow disambiguate between such predicted word forms. This can be most easily addressed in case of nouns, where applicable definite articles differ widely with the grammatical features (gender, case and number) of a noun. Definite articles could resolve the ambiguity in Example 3.1 by making the gender explicit:

(3.2) `Schere<NMasc_n_n>` describes: "der Schere", "des Scheren", "den Scheren", "der Scheren", "die Scheren", "dem Scheren"
`Schere<NFem_0_n>` describes: "die Schere", "der Schere", "der Scheren", "die Scheren", "den Scheren"

Although there still is some overlap in Example 3.2, "des Scheren" and "dem Scheren" are unique to `Schere<NMasc_n_n>` and "die Schere" is unique to `Schere<NFem_0_n>`. The NoH is 1170 for "des Scheren" and 18900 for "die Schere".[3]

[3]The NoH for "dem Scheren" is admittedly highest with 20300, but this is due to the fact that "Scheren" is ambiguous in the German language and can either mean "scissors" or "the act of shearing" and the discussion of the latter on the Internet seems to be popular with pet owners.

Nouns are easy to disambiguate this way, which is one of the reasons why I decided to restrict myself to just evaluating those in this approach. The disambiguation of verbs in a similar manner would be a more complex task, since word forms can differ with respect to features such as tense or grammatical mode, requiring the development of a range of more complex query constructs in order to disambiguate the individual forms properly.

3.3 Procedure and Implementational Details

3.3.1 Corpus and Gold Standard

The TIGER treebank corpus is used as a source for raw text tokens. Although originally chosen for its rich annotation data, no use is made of these. In effect, any other source of tokenized raw text should work with the system.

The "morphisto" morphological analyzer[4] for the most common German words provides a lexicon file with close to 20000 entries that lists the stem and associated continuation class for each of these words (see [Zielinski and Simon, 2008]). Although the line format differs from that of the system-generated hypotheses, class tags follow convention in both systems and can be matched. Stems can be extracted from the lexicon by means of regular expressions and can also be matched.

For evaluation purposes, the set of corpus word forms to be analyzed by the system can optionally be restricted to those that are in the gold-standard lexicon. This still requires processing of and hypothesizing over the whole set of word forms to determine if the hypothesis set for each form contains a member that is matchable to an entry in the lexicon. Only if this is not the case, the word form is removed from consideration.

3.3.2 Morphology Modification

The morphology that I base my work upon was never intended for the here described application and therefore does not return the grammatical features that are essential to perform article attachment as seen in Example 3.2. The raw word forms are occasionally also returned in inconsistent order which prevents the post processing of the output by automatically assigning features to different places of the returned list. I therefore opted for attempting an extension of the morphology to return the desired grammatical features along with the word forms.

[4] built on the same finite state technology as the hypothesizer used here

Although the morphology uses grammatical features internally to generate the word forms, I was not able to modify it into passing them through the output. I however found a slightly non-elegant, special solution that only required the removal of 5 characters in the definition of one transducer that upon *analysis* of a word form would return the matching stems and inflectional classes *along with* the grammatical features of the *analysed word form* (see Example 3.6 below). So in order to obtain the features which a certain hypothesis assigns to its potential word forms, I would analyse this word form using the hacked version of the morphology and select the features from the result set that are attached to the original hypothesis.

As an example, consider the last word ("Schafen") from Example 3.3, the (unmodified) morphology produces a certain set of hypotheses.

(3.3) "Es gibt Scheren zum Scheren von Schafen"

Example 3.4 shows an excerpt from the set of all hypotheses applicable to the word form "Schafen", among which is also the correct one (`Schaf<NNeut_es_e>`).

(3.4) ...
`Schafen<NMasc_s_x>`
`Schafen<NNeut_es_en>`
`Schafe<NFem_0_n>`
`Schafe<NNeut_s_0>`
`Schafe<NNeut_s_n>`
`Schafe<NMasc_s_0>`
`Schafe<NMasc_n_n>`
`Schafe<NMasc_s_n>`
`Schaf<NMasc_es_e>`
`Schaf<NNeut_es_e>`
`Schaf<NMasc_s_en>`
`Schaf<NMasc_es_en>`
`Schaf<NFem_0_en>`
`Schaf<NMasc_en_en>`
`Schaf<NNeut_es_en>`
...

Each of these hypotheses can describe a number of word forms. Using the (unmodified) morphology in reverse, one can generate all the word forms a certain hypothesis "predicts" or "describes":

(3.5) `Schaf<NNeut_es_e>` describes: "Schaf", "Schafe", "Schafen", "Schafes", "Schafs"

It is now desirable to determine the matching article for each of these word forms. Due to the fact that the morphology is not guaranteed to return an ordered set of results, I chose the option of obtaining the morphological features for each word form instead. I apply the *modified* morphology to each of these predicted word forms. (Part of) the results of applying it to "Schafe" can be observed in Example 3.6.

(3.6)
```
...
Schafe<NMasc_n_n><+NN><Masc><Nom><Sg>
Schafe<NMasc_s_n><+NN><Masc><Dat><Sg>
Schafe<NMasc_s_n><+NN><Masc><Nom><Sg>
Schafe<NMasc_s_n><+NN><Masc><Acc><Sg>
Schaf<NMasc_es_e><+NN><Masc><Nom><Pl>
Schaf<NMasc_es_e><+NN><Masc><Acc><Pl>
Schaf<NMasc_es_e><+NN><Masc><Gen><Pl>
Schaf<NNeut_es_e><+NN><Neut><Nom><Pl>
Schaf<NNeut_es_e><+NN><Neut><Acc><Pl>
Schaf<NNeut_es_e><+NN><Neut><Gen><Pl>
...
```

Any line of the result set that does not match the hypothesis we are currently investigating (`Schaf<NNeut_es_e>`) is discarded, leaving Example 3.7.

(3.7)
```
Schaf<NNeut_es_e><+NN><Neut><Nom><Pl>
Schaf<NNeut_es_e><+NN><Neut><Acc><Pl>
Schaf<NNeut_es_e><+NN><Neut><Gen><Pl>
```

The hypothesis part of these results is simply discarded leaving Example 3.8

(3.8)
```
<+NN><Neut><Nom><Pl>
<+NN><Neut><Acc><Pl>
<+NN><Neut><Gen><Pl>
```

This tells us that for `Schaf<NNeut_es_e>`, "Schafe" is the single plural inflection of "Schaf" used in case of[5] *nominative, genitive* and *accusative*. It is trivial to construct a query from this using a mapping of features to articles as is hard coded in Example 3.1.

```
-- defines a mapping between grammatical
-- features and (definite) articles
local mapping = {
```

[5] *Pun intended.*

```
    ["<+NN><Neut><Nom><Sg>"]="das",
    ["<+NN><Neut><Gen><Sg>"]="des",
    ["<+NN><Neut><Dat><Sg>"]="dem",
    ["<+NN><Neut><Acc><Sg>"]="das",
    ["<+NN><Neut><Nom><Pl>"]="die",
    ["<+NN><Neut><Gen><Pl>"]="der",
    ["<+NN><Neut><Dat><Pl>"]="den",
    ["<+NN><Neut><Acc><Pl>"]="die",
    ["<+NN><Masc><Nom><Sg>"]="der",
    ["<+NN><Masc><Gen><Sg>"]="des",
    ["<+NN><Masc><Dat><Sg>"]="dem",
    ["<+NN><Masc><Acc><Sg>"]="den",
    ["<+NN><Masc><Nom><Pl>"]="die",
    ["<+NN><Masc><Gen><Pl>"]="der",
    ["<+NN><Masc><Dat><Pl>"]="den",
    ["<+NN><Masc><Acc><Pl>"]="die",
    ["<+NN><Fem><Nom><Sg>"]="die",
    ["<+NN><Fem><Gen><Sg>"]="der",
    ["<+NN><Fem><Dat><Sg>"]="der",
    ["<+NN><Fem><Acc><Sg>"]="die",
    ["<+NN><Fem><Nom><Pl>"]="die",
    ["<+NN><Fem><Gen><Pl>"]="der",
    ["<+NN><Fem><Dat><Pl>"]="den",
    ["<+NN><Fem><Acc><Pl>"]="die",
};
```

Listing 3.1: Code excerpt from `diligentweb.lua`, mapping grammatical features to articles

The result of applying this mapping to the word form “Schafe” can be seen in Example 3.9 below.

(3.9) “die Schafe”, “der Schafe”[6]

This procedure is repeated for the other word forms in Example 3.3 to eventually build a query for each.

3.3.3 Accessing Search Engines

Yahoo, Google and Bing each provide RESTful APIs to not only access their search services from resources other than a web browser but optionally also obtain the results as a JSON data structure (as opposed to an HTML formatted document). JSON is simple to parse and its usage reduces network overhead over HTML or XML due to dense encoding of content in a compact tree structure.

[6] The system removes duplicate elements.

All search engine providers offer unlimited access to their search engine, but require (or in case of Google "encourage") that the "application" identifies itself using a registered ID. Such an (application) ID (or "key") is obtained through an registering process that (while requiring a textual description of the service one intends to implement) is most likely completely automated on the side of the providers.[7]

Yahoo offers access to their search engine without hard limits for users of its soon-to-be-commercial[8] *BOSS*[9] service ([Yahoo Inc., 2009b], [Tran, 2008]). As of the time of this writing, usage of the API is still free of charge for developers and no information has been available about the time of termination of this circumstance.

Upon re-branding their "Live Search" product as "Bing" and releasing a new API, Microsoft removed all previous restrictions on query quota ([Manoochehri, 2009]).

Google does not enforce access limits on its "AJAX Search API". In previous, more experimental APIs, Google enforced a quota of 1000 queries per day and application ID (or "key"), a measure that was necessary due to limited the resources ([Google Inc., 2009b]). Google is the only provider to not require authentication via an application ID, but also stands out by requiring an HTTP referrer URL of an existing site for all API accesses. For this purpose, I created a simple web page where I briefly explaining this project and used this as a referrer. This requirement forced me to use the Unix command `curl` (a case which is even described in the API documentation) in place of the more primitive native Lua HTTP client library.

3.3.4 Result Collection

The previously identified overlap of predicted word forms between different hyptheses combined with the possibility that two ostensibly different observed words could be different inflections of the same root was initially expected to lead to a certain amount of redundancy between queries. It was therefore decided to cache the NoH for each query in order to save network bandwidth and (due to the greater latency of querying compared to cache

[7] Confirmation emails of successful registration were typically received mere seconds after completing the web form.

[8] Yahoo! Inc. is currently planning a fee structure for the BOSS API based on the number of search results requested. In the use case presented here, no actual results are needed (and it is indeed possible to request a result set of size 0), which is however not accounted for in the planned billing scheme ([Yahoo Inc., 2009a]). The question if this poses any issues for the accounting system should be subject of future investigation.

[9] ***B****uild your* ***O****wn* ***S****earch* ***S****ystem*

access) time. The cache maps query strings to the returned NoH. For added reliability, the cache primarily resides in a data structure that is held in persistent memory, thereby protecting it from program failures. This facilitates modifications and improvements to the system without the need for rebuilding the cache each time.

3.3.5 Result Analysis via Simple Heuristic

For a given word form the set of corresponding hypotheses needs to be boiled down until only one hypothesis remains. In order to achieve this, each hypothesis is ranked according to how many of the word forms it predicts achieve the highest NoH in their category. (*e.g.* no other hypothesis has a higher NoH count for the (*e.g.*) nominative singular word form.) In case this ranking results in a tie for two or more hypotheses, this is resolved by summing the NoHs for each hypothesis and comparing these. A special rule exists in case a hypothesis has a NoH of zero for one or more of its predicted word forms. In this case, the offending hypothesis is also ranked with zero, usually putting it at the bottom of the heap.[10]
After ranking them in this way, either an ordered list of hypotheses or the single most likely one is returned.

3.4 Results

The nouns hypothesized upon divide into the genders as listed in Table 3.1.

masculine nouns	3336
feminine nouns	2697
neuter nouns	1309
total	7342

Table 3.1: Numbers of Nouns, Ordered by Gender

Four different accuracy measures were applied to the results produced by each search engine.

3.4.1 Remarks on Search Engine Performance

Query response time was slowest for Yahoo, with Bing being marginally faster and Google between two and three times as fast as Yahoo.

[10]For reasons of system robustness, the hypothesis is *not* simply removed.

Percentage of Results	Bing	Yahoo	Google
agreeing with gold standard on masculine gender	57%	65%	73%
agreeing with gold standard on feminine gender	63%	71%	82%
agreeing with gold standard on neuter gender	53%	72%	78%
average agreement with gold standard so far	59%	69%	78%
agreeing with the gold standard on everything	42%	55%	61%

Table 3.2: Accuracy Percentages for each Search Engine

A rough estimate of response times credits Google with five queries per second, Bing with three and Yahoo with two.

Quality of Service (QoS) was worst for Bing, with connections dropped on such a regular basis that failsafe mechanisms needed to be introduced to automatically resume querying in these cases. While only a single similar failure occurred for Yahoo, Google never exhibited this kind of problem.

It should probably be noted that Google has the lowest average of hits for all queries. In very general terms, the average number of hits differs by several orders of magnitude between search engines, with Bing usually claiming hit numbers two to three orders of magnitude higher than Google.

Finally, the caching approach proved viable, with the results for query being accessed an average of 8 times.

Chapter 4

Discussion

In this chapter, I discuss the results obtained from the approaches detailed in Chapters 2 and 3 above.

4.1 Two-Level Compression

Within the two-level corpus-compression approach, two different paradigms for approximating the bit length of the compressed model were evaluated. One supposedly enabled a quick analytic approximation of the worst-case bit length of the model, while the others provided a more accurate numeric solution.

When comparing these two paradigms for approximating the Huffman-tree length in the model, an unexpectedly high degree of apparent functional equivalence must be acknowledged. This degree of similarity between results of the two paradigms (Figures 2.5 and 2.6) begs for an explanation.

One could argue that a largely balanced probability distribution for the training data results in theoretical Huffman-tree lengths that are indeed close to the worst case. This would make the usage of the worst case length as an approximation viable only for the given data. Unfortunately no additional training runs were conducted on different data, which would have been useful to support or discredit this hypothesis. However, barring the possibility of errors, no other explanations come to mind.

The results for either part of the two-level compression describing the data (Figures 2.7 and 2.8) are worth of notice. By allowing either corpus or model length to run unchecked, one would expect the system to put a strong bias on the shortening of the bit length of the respective complement. However, Figure 2.9 suggests otherwise: While emphasis is on the "controlled"

part, *both* compression sizes usually decrease with advancement of training.

4.2 Search-Engine Hits

Although only a very basic heuristic was employed for the determination of accuracy for the search-engine hits approach, the results are promising. Startling are the apparent differences in accuracy for the different genders of nouns. While the feminine gender is generally recognized with a higher probability than the masculine, comparison with the neuter gender reveals startling differences. With a probability below that of any other category, Bing performs worst in the correct classification of the neuter gender. In contrast, for Yahoo, the neuter gender is correctly classified with the highest probability.

It should be noted that even at its worst, Google out-performs its competitors at their best.

4.3 Comparing the Approaches

While an exploration of the relation of information content and the measure of search-engine hits might lead to interesting results, both approaches have little methodology in common. I therefore compare them in terms of their common task and output.

While the two-level compression approach as detailed in Chapter 2 builds upon a very compelling theory, its (practical) applicability to the problem at hand is at least doubtful. Above all, in its current state, the system is not capable of even matching the performance of the heuristics-based system (as presented by [Adolphs, 2008]) that I initially intended to improve upon.

Since the problem of data sparseness can only be addressed by entering more and more word forms into the compression, (each time requiring a number of complete recalculations), I can see little practical use for this approach.

In contrast, (my implementation of) the search-engine system has a higher potential for putting it to use as an actual tool to find morphological features for any unknown word. As enforced by the terms-of-use agreements[1] imposed by search-engine providers on all users of their APIs, I programmed a tool that works as a standalone application. It can be used to analyze a

[1] See Appendix B for a somewhat amateurish analysis

specific word form and dispatch all relevant queries to any of the three search engines. This may form the basis for a future, support-vector machine (SVM) aided system for the automatic determination of morphological features of unknown word forms.

4.4 Outlook

For the two-level compression approach, several points for improvement can be considered.

- A proper pre-categorization and splitting of the corpus into well-defined subsets with common features could begin to tackle the issue of the combinatorial explosion.
- The code could be profiled properly. By exposing and removing even small redundancies in routines that get executed thousands of times, runtime could be reduced significantly.
- Routines that suffer from significantly reduced performance due to not being implemented in a native compiled language could be ported from Lua to (*e.g.*) C.

I consider all of the above suggested improvements applicable to the code of the software I implemented. However, none of these modifications can be expected to result in improved accuracy of the resulting compressions. While any successful improvement could make training on a larger data set feasible, the methodology remains essentially unaltered.

For the search-engine approach, the most promise for future improvement probably lies in the crudeness of the currently used heuristic. Training an SVM on the data can be expected to expose many regularities that are unaccounted for by the current heuristic. Indeed, preliminary tests conducted with a popular SVM implementation and default parameters[2] suggest an accuracy in excess of 90% for this approach. This certainly merits further investigation but was beyond the scope of this work to conduct properly.

[2] `libsvm` in conjunction with the shipped `easy` script

Chapter 5

Conclusion

In this thesis, I evaluated two different approaches for the task of finding the correct morphological explanation for an unknown word form, chosing from a set of candidate hypotheses. A previous take on this issue has been attempted by Peter Adolphs, who developed the required finite-state morphological analyzer suitable for generating said candidate hypotheses from a given word form that formed a crucial part for both approaches.
The first approach used insights from information theory and the minimal-description-length principle to develop a corpus representation suitable for minimizing theoretical compression size. The side effect of such a maximal compression, namely the expected association of many word forms with their correct morphological explanation, proved to be of a low magnitude.
The second approach used data-mining techniques on internet search-engines to obtain frequency estimates on the predicted word-forms for all morphological hypotheses that potentially explain a given unknown word form. The application of a basic heuristic to the data already resulted in decent accuracy, suggesting much potential for future improvement.
While the first approach appeals through its particular theoretical framework, it fails to deliver any significant improvement to the basic heuristics first suggested by Adolphs. The search-engine-hits approach on the other hand performs well even in a very practical scenario, while additionally leaving room for further improvement. The preliminary basic heuristics could make way for a more advanced classification system in the vein of support vector machines or similar.

Bibliography

[Adolphs, 2008] Adolphs, P. (2008). Acquiring a poor man's inflectional lexicon for german. In (ELRA), E. L. R. A., editor, *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*. ELRA. URL: http://www.lrec-conf.org/proceedings/lrec2008/pdf/867$_p$*aper.pdf*.

[Brin and Page, 1998] Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117.

[Goldsmith, 2001] Goldsmith, J. (2001). Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, 27:153–198.

[Goldsmith, 2004] Goldsmith, J. (2004). An algorithm for the unsupervised learning of morphology. *Natural Language Engineering (to apperar)*.

[Google Inc., 2009a] Google Inc. (2009a). Google ajax search api terms of use. [Online; accessed 02-November-2009].

[Google Inc., 2009b] Google Inc. (2009b). Google soap search api. [Online; accessed 11-November-2009].

[Grünwald, 2004] Grünwald, P. (2004). A tutorial introduction to the minimum description length principle. *math/0406077*.

[Grünwald, 2007] Grünwald, P. D. (2007). *The Minimum Description Length Principle (Adaptive Computation and Machine Learning)*. The MIT Press.

[Gulli and Signorini, 2005] Gulli, A. and Signorini, A. (2005). The indexable web is more than 11.5 billion pages. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, page 903. ACM.

[Huffman, 1952] Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. In *PROCEEDINGS OF THE I.R.E.*, volume 1, pages 1098–1102.

[Kilgarriff, 2007] Kilgarriff, A. (2007). Googleology is bad science. *Computational Linguistics*, 33(1):147–151.

[Koskenniemi, 1983] Koskenniemi, K. (1983). *Two-level Morphology: A General Computational Model for Word-Form Recognition and Production.* PhD thesis, University of Helsinki.

[Long et al., 2004] Long, J., Skoudis, E., and van Eijkelenborg, A. (2004). *Google hacking for penetration testers.* Syngress Publishing.

[Manoochehri, 2009] Manoochehri, M. (2009). Microsoft releases bing api - with no usage quotas. [Online; accessed 11-November-2009].

[Microsoft, 2009] Microsoft (2009). Bing web service api terms of use. [Online; accessed 02-November-2009].

[O'Grady et al., 1997] O'Grady, W., Dobrovolsky, M., and Katamba, F. (1997). *Contemporary Linguistics — An Introduction.* Longman.

[Oxford University Press, 2009] Oxford University Press (2009). Oxford advanced learners dictionary, possible entries for lemma. [Online; accessed 21-November-2009].

[Schmid et al., 2004] Schmid, H., Fitschen, A., and Heid, U. (2004). Smor: A german computational morphology covering derivation, composition, and inflection. In *Proceedings of the 4th International Conference of Language Resources and Evaluation*, page 1263Âŋ1266.

[Shannon, 1948] Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423,623–656.

[Tran, 2008] Tran, S. (2008). Yahoo! opens up search technology infrastructure for innovative, new search experiences, providing third parties with unprecedented access, re-ranking and presentation control of web search results. [Online; accessed 02-November-2009].

[van Gelderen, 2009] van Gelderen, E. (2009). Function words. [Online; accessed 17-August-2009].

[Vitanyi and Li, 1999] Vitanyi, P. and Li, M. (1999). Minimum description length induction, bayesianism, and kolmogorov complexity. *cs/9901014.* IEEE Transactions on Information Theory, 46:2(2000), 446-464.

[Yahoo Inc., 2009a] Yahoo Inc. (2009a). Yahoo! search boss. [Online; accessed 02-November-2009].

[Yahoo Inc., 2009b] Yahoo Inc. (2009b). Yahoo! search boss. [Online; accessed 17-August-2009].

[Yahoo Inc., 2009c] Yahoo Inc. (2009c). Yahoo! search boss services terms of use. [Online; accessed 02-November-2009].

[Zielinski and Simon, 2008] Zielinski, A. and Simon, C. (2008). Morphisto: An open-source morphological analyzer for german. In *Proceedings of the Conference on Finite State Methods in Natural Language Processing.*

Appendix A

Notation by Example

In order to provide a slightly more intuitive approach to the notation, it is in the following reiterated by example. For reasons of simplicity, let the word form "Mannes" be at the 23rd place of a list of word forms (W) from the TIGER corpus (C). The word form w_{23} ("Mannes") occurs f_{23} (1 8) times in C. The set (H_{23}) of all potential morphological explanations (hypotheses) for w_{23} is generated. (See below for a subset of H_{23}.)

```
...
Manne<NNeut_s_s>
Manne<NMasc_s_0>
Mann<NMasc_es_e>
Mann<NMasc_es_$e>
Mann<NNeut_es_e>
Mann<NMasc_es_$er>
...
```

Each entry in H_{23} is a concatenation of a stem and a morphological continuation class. *e.g.* for the above subset, $h_{23,8}$ (Mann<NNeut_es_e>) is the concatenation of $t_{23,8}$ (Mann) and $c_{23,8}$ (<NNeut_es_e>). All the potential word forms ($P_{23,8}$) generated with $h_{23,8}$ and GEN are these:

```
Mann
Manne
Mannen
Mannes
Manns
```

As can be easily observed, $p_4 \in P_{23,8}$ (Mannes) is equal to w_{23} (Mannes), my starting point. The index (4) of p_4 has also be stored when encoding w_{23} using $h_{23,8}$ and GEN.
In short: Mannes $= GEN($Mann<NNeut_es_e>$)_4$.

Appendix B

Legal Gray Areas

Some care has been taken to stay within the legally permissible usage of the search APIs. The official use case for all of the used APIs is mainly their employment for providing application-/web site-embedded dynamic search functions. As such, none of the search engine providers cover the here used paradigm in their respective *Terms of Use* (ToU) agreements in satisfactory detail. Permissive usage is generally defined in terms of what the user must or must not do with the "obtained (search) results", which in my case are either simply discarded or not present, the search meta data (containing the number of hits) being the only information of interest.

B.1 The Yahoo! BOSS API ToU agreement

The following paragraph from the BOSS API ToU document *only apparently* prevents researchers from legally making any practical use of the API in the manner described here.

> "You are permitted to use the Services only for the purpose of incorporating and displaying Web Search Results from such Services as part of a Search Product deployed on your Web site ("Your Offering"). A "Search Product" means a service which provides a response to a request in the form of a search query, keyword, term or phrase (each request, a "Query") served from an index or indexes of data related to Web pages generated, in whole or in part, by the application of an algorithmic search engine."([Yahoo Inc., 2009c])

Another part of the same ToU requires the presence of a well defined search mask on the web site incorporating the BOSS service.

> "You will include in the user interface of Your Offering at least one of the following implementations of the Service (each,

> a "Link"): (i) a field or graphical area that accepts typed-in text (*i.e.*, a "Search Box") enabling a user to enter a Query (ii) words that are displayed in the form of hyper links, that generate a Query comprising when clicked on or used by a user (a "Hyper link"); or (iii) a method that sufficiently elicits a user's specific intent to initiate or refine a certain Query. You will use the Links to initiate a Query to the Services, in response to which Web Search Results are served from Yahoo!. You will not request Web Search Results by any means except such Links, and will not place Links or Web Search Results on any location except for the Your Offering."([Yahoo Inc., 2009c])

At no point is an automated query dispatch mechanism explicitly forbidden. Nor is the user required to prohibit automated queries from being dispatched *through* his web site. This would allow the usage of the BOSS API through such a custom proxy web site, rendering the above restriction of the service usage *to* web sites useless. Although Yahoo reserves the right to monitor the web site for compliance with their ToU, there is no explicit requirement for the web site to be (publicly) accessible. It should therefore be noted that all data acquired through the BOSS API has potentially been acquired through means permitted by the ToU.

B.2 The Microsoft Bing Web Service API ToU agreement

Similar to Yahoo, Microsoft does not explicitly require public access to the web site or application using their API. Unlike Yahoo, the Bing API ToU contents itself with merely defining a comparatively liberal "code of conduct" that reserves the right to apply the service to a number of uses. Among those, a user must not

> "(n) copy, store, or cache any Bing results, except for the intermediate purpose allowed in §2(b);"([Microsoft, 2009])

Although there is no dedicated §2(b) anywhere in the ToU, the *first* part of §2 says the following.

> "Solely to the extent that you are in compliance with all terms of this Agreement, we grant you a non-exclusive, non-transferable, non-sub-licensable license to use the services to: enable your Website or application to obtain Bing results; make limited intermediate copies of the Bing results, solely as necessary

> to display them on your Website or application; and host and display Bing results on your Website or application."([Microsoft, 2009])

One interpretation of the sole purpose of (part of) my "application" can indeed be the extraction and " of *meta data* from the result set.[1] No explicit prohibition of automated query dispatch has been specified, the closest (but still inapplicable) mention of a potentially related usage pattern is given in the following quote.

> "(q) directly or indirectly generate impressions or clicks on Bing results, or authorize or encourage others to do so, though[*sic!*] any automated, deceptive, fraudulent, or other invalid means."([Microsoft, 2009])

I come to the conclusion that the Bing ToU accommodates my use case.

B.3 The Google AJAX Search API ToU agreement

The Google AJAX Search API ToU agreement suffers from the same lack of definition of what constitutes "Search Results" as the other agreements. This is particularly unfortunate because of the following statements from the introductory sentences.

> "The API consists of [...] service protocols that enable You to display results from Google searches [...] ("Google Search Results") on your website, in your application, or in your other product expressly authorized in writing by Google (each, a "Property"), subject to [...] limitations and conditions [...]. You are allowed to use the API only to display, and to make such uses as are necessary for You to display, Google Search Results on your Property."([Google Inc., 2009a])

If the meta data that a query using the API returns along with the search results for any query is considered as consitutes part of whatever is denoted by "Google Search Results", the user is prohibited to store them by the following restriction.

> "[*You agree that when using the Service, You will not, and will not permit users or other third parties to*] copy, store, archive, republish, or create a database of Google Search Results, in whole or in part, directly or indirectly, except that You may display Google Search Results that have been "clipped" through an end user-requested action [...].;"([Google Inc., 2009a])

[1] The relevant module of my system has been made capable of running as a dedicated "application" in a standalone mode for precizely this purpose.

It should be noted however that the mere *caching* of only meta data for the purpose of later display might not pose a violation of these restrictions.[2] Nor should this forbid the (statistical) post-processing of meta data before "displaying" it, in which case the original meta data should still be displayed alongside for reasons that become apparent in the following paragraph.

> "You agree that You will not, and You will not permit your users or other third parties to: (a) modify or replace the text, images, or other content of the Google Search Results, including by (i) changing the order in which the Google Search Results appear, (ii) intermixing Search Results from sources other than Google, or (iii) intermixing other content such that it appears to be part of the Google Search Results; or (b) modify, replace, obscure, or otherwise hinder the functioning of links to Google or third party websites provided in the Google Search Results."([Google Inc., 2009a])

Meta data is here considered to fall into the category of "other content" and the above conditions do not appear to restrict the post procession of the "Search Results" in any way, as long as the original is retained alongside. Finally, while the following paragraph restricts the crawling and indexing of the search results, the use of the API for automated querying the search engine is not explicitly forbidden here or in any other part of the ToU.

[2] As actual results are of no interest, "Search Result Clipping" can well be considered an "end user-requested action" that justifies their disregarding by the system.

Affirmation

I hereby confirm that I worked on this thesis independently and that I have not made use of any resources or means other than those indicated.

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbständig erstellt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Thorben Krüger, Amsterdam, November 25, 2009