Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Stig-Arne Grönroos

Semi-supervised induction of a concatenative morphology with simple morphotactics

A model in the Morfessor family

Master's Thesis
Espoo, Mar. 4, 2014

Supervisor: Professor Mikko Kurimo
Advisors: Sami Virpioja, D.Sc. (Tech.)
Peter Smit, M.Sc. (Tech)

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Stig-Arne Grönroos

Title:
Semi-supervised induction of a concatenative morphology with simple morphotactics: A model in the Morfessor family

Date: Mar. 4, 2014 **Pages:** ix + 74

Major: Information and Computer Science **Code:** T-61

Supervisor: Professor Mikko Kurimo

Advisors: Sami Virpioja, D.Sc. (Tech.)
Peter Smit, M.Sc. (Tech)

Machine learning methods are increasingly applied to automated processing of natural language data. One motivation for this stems from the different amounts of resources available to languages: knowledge-heavy manual approaches are only available for well resourced languages.

Morphological segmentation, the splitting of words into their smallest meaning-bearing components, is an essential task in automatic processing of compounding and highly-inflecting languages. In these languages, the number of unique word forms may be very large, causing problems for word-based language models.

This thesis presents Morfessor FlatCat, a new method in the Morfessor family of methods for learning morphological segmentations of words. Morfessor FlatCat hybridizes two existing Morfessor methods, combining the hidden Markov model morphotactics from Morfessor Categories-MAP with the semi-supervised training previously applied to Morfessor Baseline. The semi-supervised training is enabled by the use of a flat instead of a hierarchical lexicon.

The morphotactics help the model avoid using correct morphs in incorrect positions, such as splitting the English suffix "s" from the beginning of a word. Semi-supervised learning allows using small amounts of annotated data for significant improvements when evaluated against gold standard segmentations.

Our experiments show that while unsupervised FlatCat does not reach the accuracy of Categories-MAP, FlatCat provides state-of-the-art results for English and Finnish, when trained in a semi-supervised manner. Information retrieval experiments demonstrate the applicability of FlatCat to a natural language processing task.

Keywords: morpheme segmentation, morphology induction, machine learning, language modeling, probabilistic modeling, concatenative morphology, morphotactics, semi-supervised learning, unsupervised learning, information retrieval

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Stig-Arne Grönroos		
Työn nimi:	Puoliohjattu konkatenatiivisen morfologian oppiminen yksinkertaisella morfotaksilla: Morfessor-malliperheen jäsen		
Päiväys:	4. maaliskuuta 2014	**Sivumäärä:**	ix + 74
Pääaine:	Informaatiotekniikka	**Koodi:**	T-61
Valvoja:	Professori Mikko Kurimo		
Ohjaajat:	Tekniikan tohtori Sami Virpioja Diplomi-insinööri Peter Smit		

Koneoppimismenetelmiä hyödynnetään yhä enemmän luonnollisen kielen käsittelyyn. Eri kielten vaihtelevat resurssit ovat osasyy tähän kehitykseen: vahvasti kieliopilliseen tietämykseen nojautuvat manuaaliset menetelmät ovat realistisia vain hyvin resursoiduille kielille.

Morfologinen segmentointi, eli sanojen jakaminen pienimpiin merkitysellisiin osiin, on keskeinen tehtävä yhdyssanoja ja taivutusta paljon hyödyntävien kielten automaattisessa käsittelyssä. Näissä kielissä erilaisten sanamuotojen määrä kasvaa niin suureksi, että sanoihin pohjautuvat kielimallit kärsivät.

Tämä diplomityö esittelee Morfessor FlatCat -menetelmän, Morfessor-menetelmäperheen uuden jäsenen. Menetelmäperhe sisältää koneoppimismenetelmiä morfologisen segmentoinnin oppimiseen. Morfessor FlatCat yhdistää osia kahdesta olemassaolevasta Morfessor-menetelmästä, liittäen kätkettyyn Markov-malliin perustuvan morfotaksin Morfessor Categories-MAP -menetelmästä yhteen Morfessor Baseline -menetelmässä käytetyn puoliohjattun oppimisen kanssa. Puoliohjatun oppimisen mahdollistaa ei-hierarkisen leksikon käyttäminen.

Morfotaksi auttaa mallia välttämään olemassaolevien morfien käyttämistä virheellisissä kohdissa. Esimerkkinä virheestä on englanninkielen jälkiliitteen ”s” käyttäminen sanan alussa. Puoliohjattu oppiminen mahdollistaa pienen annotoidun aineiston hyödyntämisen merkittävän hyödyn saavuttaamiseen verrattaessa tuloksia normatiiviseen segmentointiin.

Kokeet osoittavat, että vaikka ohjaamaton FlatCat ei saavuta Categories-MAP -menetelmän tarkkuutta, puoliohjattuna FlatCat-menetelmän tulokset suomen- ja englanninkielelle ovat nykyistä huipputasoa. Tiedonhakukokeet osoittavat menetelmän soveltuvuuden erääseen luonnollisen kielen käsittelyn tehtävään.

Asiasanat:	morfeemipilkonta, morfologian oppiminen, koneoppiminen, kielimallit, todennäköisyysmallit, konkatenatiivinen morfologia, morfotaksi, puoliohjattu oppiminen, ohjaamaton oppiminen, tiedonhaku
Kieli:	Englanti

Aalto-universitetet
Högskolan för teknikvetenskaper

Aalto-universitetet
Högskolan för teknikvetenskaper
Examensprogram för datateknik

SAMMANDRAG AV
DIPLOMARBETET

Utfört av: Stig-Arne Grönroos

Arbetets namn:
Halvstyrd inlärning av konkatenativ morfologi med en enkel morfotax: En medlem i Morfessor-modellfamiljen

Datum:	Den 4 mars 2014	**Sidantal:**	ix + 74
Huvudämne:	Informationsteknik	**Kod:**	T-61
Övervakare:	Professor Mikko Kurimo		
Handledare:	Teknologie doktor Sami Virpioja Diplomingenjör Peter Smit		

Maskininlärningsmetoder utnyttjas allt mer för automatisk behandling av data i form av naturligt språk. En orsak är variationen i tillgängliga resurser hos olika språk: det är endast realistiskt att tillämpa kunskapsbaserade manuella metoder för resursrika språk.

Morfologisk segmentering, det vill säga uppdelande av ord i deras minsta meningsfulla beståndsdelar, är en central uppgift i behandling av språk som innehåller många sammansatta ord och böjningsformer. I dessa språk kan antalet unika ordformer vara mycket stort, vilket kan orsaka problem för ordbaserade språkmodeller.

Detta diplomarbete presenterar metoden Morfessor FlatCat, en ny medlem i Morfessor-metodfamiljen. Metodfamiljen består av maskininlärningsmetoder för morfologisk segmentering. Morfessor FlatCat kombinerar den på en dold Markov-modell baserade morfotaxen från Morfessor Categories-MAP med den halvstyrda inlärningen som tidigare tillämpats i Morfessor Baseline. Den halvstyrda inlärningen möjliggörs av användandet av ett lexikon utan inre struktur.

Modellens morfotax hjälper att undvika användandet av existerande morfer på fel ställen. Ett exempel av denna typ av fel är att använda det engelska suffixet ”s” i början av ett ord. Halvstyrd inlärning låter små mängder av annoterat data utnyttjas för betydlig förbättring av resultatet då man jämför med en normativ segmentering.

Våra experiment visar att, även om ostyrd FlatCat inte når samma noggranhet som Categories-MAP, är resultaten för halvstyrd FlatCat av rådande toppnivå för engelska och finska. Experiment i informationssökning demonstrerar metodens lämplighet för en uppgift inom behandling av naturligt språk.

Nyckelord:	morfemsegmentering, inlärning av morfologi, maskininlärning, sannolikhetsmodeller, konkatenativ morfologi, morfotax, halvstyrd inlärning, ostyrd inlärning, informationssökning
Språk:	Engelska

Acknowledgements

This work has received funding from the European Community's Seventh Framework Programme (FP7/2007–2013) under grant agreement n°287678. The experiments were performed using computer resources within the Aalto University School of Science "Science-IT" project.

First, I would like to thank my supervisor, Prof. Mikko Kurimo, for welcoming me into his Speech group and for his excellent guidance and attention to the quality of my work.

I would like to offer my gratitude to my two advisors, Peter Smit and Dr. Sami Virpioja, for sharing their knowledge with me. Their advice has made me a better scientist.

I greatly appreciate my officemate Reima Karhila showing me the ropes in the lab.

I would also thank Dr. Mathias Creutz, Oskar Kohonen, Teemu Ruokolainen and Matti Varjokallio, for their comments and many valuable discussions.

Finally, I wish to thank my family and friends. I am especially grateful to my fiancée Riikka for all the support and encouragement I have received, and also for patiently discussing my thoughts, even when they turned out to be esoteric details of matters outside her academic expertise.

Espoo, Mar. 4, 2014

Stig-Arne Grönroos

Abbreviations and acronyms

BPR	Boundary Precision and Recall
FST	Finite-State Transducer
HMM	Hidden Markov Model
IA	Item-and-Arrangement
IR	Information Retrieval
MAP	Maximum A Posteriori
MDL	Minimum Description Length
ML	Maximum Likelihood
POS	Part Of Speech
PRE	Prefix
STM	Stem
SUF	Suffix
NON	Non-morpheme

Notation

Introduced terms are indicated by *italics*.
Examples are indicated by **boldface**.

Symbols

$a, b, \ldots, \alpha, \beta, \ldots$	Scalars
$A, B, \ldots$	Random variables
$\boldsymbol{a}, \boldsymbol{b}, \ldots$	Strings or other sequences
$P\left(A\right)$	Probability of A
$\mathrm{I}(A)$	Indicator function (1 if A is true, 0 otherwise)
$\lvert\boldsymbol{a}\rvert$	Length of sequence $\boldsymbol{a}$
$\mathcal{O}\left(\bullet\right)$	Time complexity
$\Re$	Real numbers
$\lceil \bullet \rceil$	Rounding towards ceiling
$\boldsymbol{\theta}$	Model parameters
Θ	Parameter space
$\mathcal{G}$	Model grammar
$\mathcal{L}$	Model lexicon
$L(\boldsymbol{\theta}, \boldsymbol{D}, \ldots)$	Cost function
$\phi(\boldsymbol{\sigma}, \boldsymbol{\theta})$	Tokenization for string $\boldsymbol{\sigma}$
$\phi^{-1}(\boldsymbol{a}, \boldsymbol{\theta})$	Detokenization for analysis $\boldsymbol{a}$
$\boldsymbol{D}_W$	Unannotated data
$\boldsymbol{D}_A$	Annotated data
A	Morphs in the analysis of a single word
$\boldsymbol{Y}$	The active analysis of the whole corpus
w	A word
m_i	The i^{th} morph in a sequence
$m_{(i)}$	The i^{th} morph in the lexicon
$\tau_{(i)}$	Occurrence count of the i^{th} morph in the lexicon
$\boldsymbol{\sigma}$	String representation of a morph
c	Morph category
$\#_w$	Word boundary symbol
$\#_m$	Morph boundary symbol

Contents

Chapter 1

Introduction

A large part of the knowledge that humans have collected is in the form of natural language, the naturally occurring form of language that is used in communication between human beings. English and Finnish are examples of natural languages. Natural language can consist of speech, writing or signing. It can be distinguished from artificial languages (such as Lojban or Sindarin) and from formal languages (such as mathematical notation or programming languages).

As natural languages have evolved over long time spans and large numbers of speakers, they commonly exhibit more irregularity and linguistic complexity than artificial languages. This results in some challenges that are particular to natural language processing.

Natural language processing (NLP) refers to the application of artificial intelligence to the task of automating the processing of natural language data. NLP technologies may not always be very visible to end users, as they are commonly used as components of larger systems, and the better they work the less the user pays attention to them.

However, making knowledge stored in the form of natural language available for consumption both by humans and by machines is a difficult challenge. Computer systems with a level of understanding of human language are a requirement for an efficient solution to this challenge. It is also much easier for humans to communicate using natural language. If computers could understand natural language well enough, this could lead to improved human–computer interfaces. Therefore NLP plays an increasingly important part in advancing technology.

Siri, the voice-driven user interface on Apple's iOS devices is an example of an NLP application that aims to hide the technology from users. Apple recommends that the user speak to the device as if it was another human. The Google search engine is another example of unobtrusively using NLP, in

this case to automatically expand the search beyond the literal keywords in the user's query.

1.1 Resource-free natural language processing

While the traditional way to construct NLP tools has involved significant expenditure of expert labor, there has recently been large interest in language-independent methods that do not require expert knowledge to use. A prominent example of this is the system of Collobert et al. (2011), which uses unsupervised neural networks to generate features that are shown to be useful for four central NLP tasks: Part-Of-Speech (POS) tagging, chunking, named entity recognition and semantic role labeling. The authors choose to avoid extensive preprocessing of the data and engineering task-specific features, in order to emphasize the generality of their method. Another example of a popular resource-free NLP method is WEBSOM by Honkela et al. (1997), a method primarily intended for the exploration of large document collections.

This trend towards resource-free methods can also be seen in the field of morphology induction (Hammarström and Borin, 2011), which is the field that this work belongs to.

One reason for this development stems from the fact that not all languages are equally resourced. Manual construction of morphologies may be unacceptably expensive for a low resource language.

The unequal status of languages is evident from the fact that many languages are in danger of becoming extinct. While the UNESCO *Atlas of the World's Languages in Danger* (Moseley, 2010) categorizes 57% of the worlds languages as safe from extinction, Kornai (2013) raises concerns about the ability of many of these languages to achieve a significant level of digital use. While nothing can substitute for a vital community of language users, the existence of NLP tools can help support language use and make digital use of the language more attractive.

Another benefit of data-driven models of natural language is the potential for patterns and regularities gathered through such means to inform new linguistic theory. Gries (2006) surveys a number of corpus-based linguistic findings.

1.2 Morphology in natural language processing

Morphology is the study of the internal structure of words. There are two different ways to divide a word into parts. The first method, dividing into syllables, follows pronunciation. An example syllabification is **syl - la - bles**. The syllables have no meaning individually, and sharing one of the syllables with another word, for example **ta - bles**, does not indicate any semantic relation.

An alternative way to divide the word into parts ignores pronunciation and focuses on meaning instead. Dividing into the smallest parts that have meaning gives morphemes. The morphological segmentation for the same example word is **syllable + s**. The suffix **s** in the example carries a grammatical meaning: it indicates the plural.

In a morphologically rich language, such as Finnish, the number of possible inflected word forms is very large. This causes sparsity in the data: one cannot assume that all valid inflections of words will be present even in large corpora. Modeling language on a sub-word level becomes essential for good performance in NLP for such languages.

The sub-word units should be easy to extract, and a compact lexicon of the units should give a good coverage of the data being modeled. In addition to these criteria, the units should also be relevant to solving the problem at hand. Morphemes provide an attractive compromise between these criteria for many NLP tasks.

Morphological information has been successfully used in diverse NLP applications, such as Information Retrieval (Kurimo et al., 2010a), speech synthesis (Demberg et al., 2007), speech recognition (Creutz et al., 2007; Hirsimaki et al., 2009), spell checking (Pirinen et al., 2010), machine translation (Virpioja et al., 2007; El-Kahlout and Oflazer, 2010), and predictive text input (Garay-Vitoria and Abascal, 2006).

1.3 Problem statement

Morfessor is a family of methods for morphological segmentation. There is a previous variant of Morfessor Baseline (Kohonen et al., 2010) that can be trained making use of small amounts of annotated data in addition to the usual unannotated data. Another variant of Morfessor, Categories-MAP (CatMAP) (Creutz and Lagus, 2005a), imposes dependencies between consecutive morphemes in order to avoid using valid morphemes in the wrong places. An example of this kind of error is to use the past tense suffix **d** in

the beginning of the word **d + art**. These two improvements are not directly compatible with each other.

The aims of this thesis are to formulate and implement a new method in the Morfessor family, which hybridizes some of the advantages of these two existing methods: Morfessor Baseline and Morfessor CatMAP.

The method has been implemented as Python software, and will be released under a permissive FreeBSD-style license.

The results of direct evaluation by comparison to linguistic gold standard are presented. Applicability of the method to NLP problems is demonstrated through an indirect evaluation using Information Retrieval (IR).

1.4 Structure of the thesis

The thesis is divided into a literature review followed by the contribution.

The literature review begins with an overview of background (Chapter 2), which is divided into machine learning and linguistics background. The overview does not aim to be comprehensive, but instead focuses on the topics relevant to the current work. Related topics are briefly explained when needed for context. The chapter also presents evaluation methods and a review of related work.

Chapter 3 begins with a historical perspective to the previous work in the Morfessor family. The chapter continues with a unified formalism for the Morfessor family, and concludes with a brief description of central Morfessor methods using this formalism.

The contribution part of the thesis begins with Chapter 4, describing the Morfessor FlatCat method, which is the main contribution of this work.

Chapter 5 describes the performed evaluations, presents their results and an analysis of some types of errors made by the system.

The thesis is concluded with a summary of the results and a discussion of directions for future research.

Chapter 2

Background

This chapter presents an overview of the concepts and methods on which the following chapters build, as well as literature on related work.

2.1 Machine learning background

One of the goals of artificial intelligence is to allow an automated system to reason about a problem. The traditional way of constructing a computer system that performs a particular task involves programming the rules that the computer will follow by hand. For many problems this kind of explicit encoding of knowledge requires prohibitively large amounts of expert labor.

It is also common that the space of possible combinations is large compared to the available data, and that negative examples are rare or not available at all. Under these circumstances the problems are under-constrained, and it is no longer enough to find a single model that could describe the data. Instead there is uncertainty about which of the correct models is optimal, and the different models must be ranked according to how well they perform.

An appealing solution to reducing the amount of work and to account for uncertainty is to use *machine learning*. Machine learning is a subfield of artificial intelligence, which takes a data driven approach to the problem. The goal of machine learning is to automatically learn from experience in the form of observed data.

In machine learning a general model is formulated, after which a data set is used to fit the parameters of the model, thus giving a specific solution to the problem at hand.

Machine learning can be used either as a mere tool for guiding the expert in constructing useful rules, or its results can be used directly as the final system. In the latter case, it may be possible to significantly reduce the need

for expertise in the problem domain, allowing a naïve user with access to suitable data to train and evaluate a system.

Machine learning uses probability to account for uncertainty. In the following paragraphs we present the notation used for describing probabilities in this thesis, using as examples the random variables U and V.

We denote the probability density function of a random variable by $P(U)$. The probability of an event is $P(U = u)$.

The joint probability of multiple random variables $P(U, V)$ gives the probability of observing a combination of values simultaneously.

If on the other hand the value of some variable, in this case V, is considered to be known, the conditional probability $P(U \mid V)$ is used instead to account for this knowledge. The value can either be known based on some evidence, or we may just be enumerating all possible values for V, and seeing how the probability of U varies in each case.

A joint probability can always be rewritten as a product using the chain rule, for example

$$P(U, V) = P(U) P(V \mid U). \tag{2.1}$$

2.1.1 Generative and discriminative modeling

It is possible to directly proceed to solving a problem based on available data, using *discriminative learning*. Discriminative models only describe the conditional probability $P(\boldsymbol{Y} \mid \boldsymbol{D})$ of the labels $\boldsymbol{Y}$ given the observations $\boldsymbol{D}$. This results in a direct mapping from inputs to outputs.

An alternative approach is to first attempt to describe the data itself, and only then solve the problem by inference using the model of the data. This approach is called *generative modeling*. It separates the knowledge representation from the process of reasoning.

Generative models describe the joint probability distribution $P(\boldsymbol{D}, \boldsymbol{Y})$ of the observations and the desired labels associated with them.

A benefit of generative models is the ability to put unlabeled data to effective use. Unlabeled samples participate directly in the optimization of the generative model, as it learns the joint distribution. Using knowledge from unlabeled samples in discriminative learning must be implemented in a more indirect way.

Generative models aim to describe the data as a whole, not just the part of it that is relevant for solving a particular problem. This can be both an advantage and a disadvantage. Generative models are more suited when it is desirable to ask more than one type of question. Discriminative

models, however, may learn a more optimal solution to a particular problem by focusing all available resources on solving it.

Another way to describe the difference is that discriminative models have the ability to easily ignore features that are uninformative for solving a specific problem. As a generative model may waste resources describing the structures of these noise features, greater care must be taken to only introduce informative features when using generative models.

2.1.2 Supervision in machine learning

When discussing the theory of general purpose machine learning methods, a division into types of supervision can be made based directly on the structure of the observed and predicted data. Zhu (2005) defines four types of supervision: unsupervised, supervised, semi-supervised and reinforcement learning.

In *unsupervised learning* the label y that is being predicted is not included in the training data $\boldsymbol{x} = \{x_1, \ldots, x_n\}$. The predictions are based only on structure in the observed features $\boldsymbol{x}$.

Supervised learning is used when the training data $(\boldsymbol{x}, y)$ contains labels y paired with each observation x. The objective is to learn the mapping from $\boldsymbol{x}$ to y, so that the label y of future samples can be predicted given only $\boldsymbol{x}$.

Semi-supervised learning is an intermediary form of learning that combines supervised and unsupervised learning. A small set of labeled data and a large set of unlabeled data are available. The objective is again to learn a mapping from features to labels, but the methods employed differ from supervised learning.

The fourth type of learning described by Zhu (2005) is *reinforcement learning*. In reinforcement learning the system repeatedly observes the environment, decides on an action to perform, and receives a reward. The goal is to learn to predict actions that maximize the reward.

Active learning can be considered an intermediary form between semi-supervised and reinforcement learning. In active learning the system elicits feedback from the user by presenting predictions. The feedback is immediately used to improve the subsequent predictions. This mechanism can be used to gather annotations that are considered most useful for the system.

2.1.3 Expert knowledge in machine learning

When applying machine learning to a research problem, the division based directly on the presence of labels may be too coarse. It only takes into account the structure of the data, but ignores its provenance. In practice,

a factor of interest is the amount of expert labor and knowledge needed for constructing the system.

Useable knowledge can take various forms. One defining spectrum is the explicitness of the knowledge representation. At the explicit end of the spectrum are hand written rules that are directly applied by the system. Another explicit type of knowledge is the manual annotation of data sets. The annotations may not be in the internal format that the system will use, but they do explicitly state the desired output.

Implicit knowledge may take the form of user feedback on the output of the system, e.g. with the values "good" and "bad".

An example of an intermediary form is found data, that contains useful information for solving a certain task, but is originally collected for a different purpose. It may require some transformations before it is usable as annotation.

The type of expertise required to express the knowledge also varies. Writing explicit rules generally requires both domain knowledge and experience with the machine learning methods and tools.

The type and availability of implicit knowledge depends on the problem at hand. In this work the domain knowledge is that of linguistics. Implicit feedback may require only the domain knowledge of a native speaker naïve in linguistics, and may not require any knowledge of machine learning.

Hammarström and Borin (2011) use the established terminology of supervision, but define it in a broader sense. They define unsupervised learning of morphology as producing a description of the morphological structure, with as little supervision (parameters, thresholds, human intervention, model selection during development, etc.) as possible, from unannotated, non-selective natural language text data. Non-selective is understood to exclude text data that requires manual selection (e.g., curated singular–plural pairs).

While the human involvement may consist of explicitly labeling data, it also includes curation of data sets and elicitation of additional data during the training, regardless of the type of machine learning method involved.

To avoid confusion we use the term *knowledge-free* for methods that not only use unlabeled data, but also aim for language independence and minimal need for expert knowledge. Methods that are not knowledge-free are called *knowledge-based*.

We use the term *semi-supervised* strictly to refer to training with mixed labeled and unlabeled data.

2.1.4 Bayesian parameter estimation

When training a probabilistic model we are given a limited set of training data, but would like to have a model that generalizes to new data.[1] This means that the goal is to receive good predictions for new, unseen data. The probability of the training data given the model parameters is still a useful measure of fit: if the training data gets a low probability, then the model may be *underfit* or incorrectly biased. The probability of new data is also likely to be low using an underfit model.

There is, however, also a risk of *overfitting* the training data. Overfitting means that the model learns even the irrelevant details of the current data set, at the cost of losing the ability to generalize. While the probability of the training data is high, the probability of new data will be low.

Given a parametric model $\mathcal{M}$, maximum likelihood (ML) estimation chooses the parameter values

$$\boldsymbol{\theta}^{\mathrm{ML}} = \max_{\boldsymbol{\theta} \in \Theta} P\left(\boldsymbol{D} \mid \boldsymbol{\theta}; \mathcal{M}\right) \tag{2.2}$$

from the chosen parameter space Θ, that maximize the probability assigned by the model to the training data.

A problem with ML estimation is that the parameters are chosen with full reliance on the training data. The method does not take into account the amount and quality of the training data, nor does it allow supplying any prior knowledge to guide the parameter selection. This makes ML prone to overfitting when used with data that is small in proportion to the model complexity.

Overfitting in ML can be controlled by either choosing a more rigid model $\mathcal{M}$, or by heuristically limiting the parameter search space Θ. Restricting the model flexibility imposes a hard constraint on the solutions, forbidding models that are flexible enough to fully describe the data set. Heuristic controls, also called *regularization*, provide a softer force guiding the choice to simpler models. Both of these require the system builder to choose the preferred model complexity. (Koller and Friedman, 2009)

A more principled alternative to heuristic controls is provided by *Bayesian parameter estimation*. Prior knowledge is formalized in a prior probability distribution $P\left(\boldsymbol{\theta}\right)$ over the parameter space. The shape of this distribution encodes the strength of the prior beliefs: a peaky distribution that assigns most of its probability mass to a small region represents a strong belief.

The prior provides a soft constraint: as long as the prior is not zero at any point where the true posterior is non-zero, the observations are able

[1] Unless we are visualizing a single data set. In the current work we do not consider this type of visualization problem.

to override the prior. The amount of data required for this depends on the strength of the prior. A prior that attempts, in some regard, to be as agnostic as possible about the shape of the posterior is called *non-informative*.

The prior knowledge is combined with the likelihood of the data using Bayes' formula, resulting in the posterior probability

$$P(\boldsymbol{\theta} \mid \boldsymbol{D}) = \frac{P(\boldsymbol{\theta})\, P(\boldsymbol{D} \mid \boldsymbol{\theta})}{P(\boldsymbol{D})}. \tag{2.3}$$

A full Bayesian treatment dispenses with setting the parameters to any single value, instead performing inference by taking the expectation, i.e. integrating the quantity to be estimated over the posterior probability. The result is that all possible parameter values contribute to the result, weighted according to their posterior probability.

When a point estimate for the parameters is desired, either for reducing the computational cost or because the parameters themselves are of interest, *maximum a posteriori* (MAP) estimation can be used. MAP finds parameter values

$$\boldsymbol{\theta}^{\mathrm{MAP}} = \max_{\boldsymbol{\theta} \in \Theta} P(\boldsymbol{\theta})\, P(\boldsymbol{D} \mid \boldsymbol{\theta}; \mathcal{M}) \tag{2.4}$$

that maximize the posterior probability. The normalization can be omitted when optimizing, as the probability of the data $P(\boldsymbol{D})$ is constant with regard to the optimized parameters.

2.1.5 Local search in MAP learning

Finding the parameters that maximize the posterior probability is an optimization problem, where the cost function can e.g. be defined as the negative logarithm of the unnormalized posterior

$$L(\boldsymbol{\theta}, \boldsymbol{D}) = -\log\big(P(\boldsymbol{\theta})\big) - \log\big(P(\boldsymbol{D} \mid \boldsymbol{\theta})\big). \tag{2.5}$$

Unless the cost function $L : \Theta \mapsto \Re$ is simple enough that an analytical solution is available, finding the optimal parameters requires a search procedure.

The search can either be over partial assignments or full assignments. When partial assignments are used, the states in the search space contain values for some parameters leaving others unassigned. The solution is then constructed by incrementally growing the set of assigned parameters until a full assignment corresponding to a local or global optimum is found. Search with partial assignments requires that the cost function can evaluate such partial assignments usefully.

Another approach is to search over full assignments that are generated from the current assignment by changing some of the values. The *search operator* or *neighborhood function* $\textsc{Op} : \Theta \mapsto \Theta^n$ maps an assignment to the set of assignments that is considered to be nearby to it. Using the full assignment approach allows performing *local search*. (Aarts and Lenstra, 1997)

The starting state of local search is some arbitrary full assignment $\boldsymbol{\theta}_0$ of the parameters. At each step the cost of each assignment in the neighborhood of the current assignment is compared to the current cost. In the first-improvement scheme the first neighbor with a better cost is selected as the new current state, and a new search iteration is started. In the best-improvement scheme the whole neighborhood is evaluated, and only at the end of the iteration is the next state chosen. (Aarts and Lenstra, 1997)

2.1.6 Minimum description length principle

Both learning and compression can be said to have the same goal: to use regularities in data to express it more concisely. In compression focus lies on the data, which we desire to store in a smaller number of bits. In learning it is the description of the regularities that is of interest, together with using the found regularities to predict new data.

Minimum description length (MDL) by Rissanen (1978) is an information theoretic framework for designing codes that achieve optimal compression for a given data set.

The field of MDL research has moved on from the older two-part MDL, also known as crude MDL, to refined one-part schemes with theoretically more optimal compression properties (Grünwald, 2007). However, these refined schemes can only be efficiently calculated for models of particular forms (e.g. Kontkanen et al., 2003). For this reason and following previous work on Morfessor, this work applies the two-part MDL scheme.

In two-part MDL, the length of the compressed data is defined as the length of the description of the code combined with the length of the data after compression using the code. The code is found from an arbitrarily chosen family of codes, assumed to be known by both communicating parties.

If the hypothesis space is restricted to a single parametric model, meaning that we are performing parameter estimation instead of more general model selection, and if a complete code is used, two-part MDL is equivalent to MAP estimation using the description length of the parameters as prior. (Virpioja, 2012)

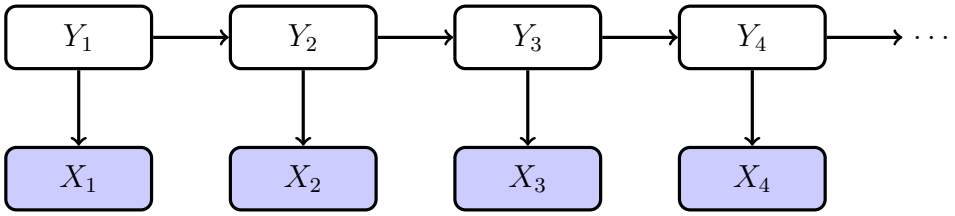


Figure 2.1: A graph representation of a first-order hidden Markov model. The indices of the random variables indicate the sequence number of the observation that the variable is associated with. Each pair of consecutive hidden states Y_i and Y_{i+1} are connected by the transition probability. The emission probability links the hidden state Y_i to the corresponding observation X_i. The blue shading indicates the observed variables.

2.1.7 Graphical probabilistic models

The same joint probability density can be parameterized in different ways, with different number of parameters. Explicit representation of joint distributions of a large number of variables by tabulating each value is intractable, because the number of parameters becomes too large to learn from the available training data and even too large to keep in memory. (Koller and Friedman, 2009)

Probabilistic graphical models use the (assumed) independence structure of a high-dimensional joint distribution to allow for a compact description. These assumptions about the independence structure are represented as a graph. The nodes represent random variables, either observed or latent. Edges represent factors of the probability density function. The shape of the graph may allow for efficient inference algorithms. An example graphical model can be seen in Figure 2.1.

A *Bayesian network* encodes conditional independence properties in a directed acyclic graph. The directed edges between the nodes represent a direct influence between the variables. Even though two variables are not directly connected by an edge, there can still be a long-range dependence trough a trail of connecting edges. However, each node is independent of its non-descendants given its parents.

2.1.8 Hidden Markov models

In its basic formulation a Bayesian network has a fixed size and a fixed number of parameters. By using the formalism of dynamic Bayesian networks the network can be extended to an arbitrary size. Parameter sharing between

the duplicated parts of the network ensures that the number of parameters does not increase. (Koller and Friedman, 2009)

A Markov model is a simple dynamic Bayesian network. It gives the probability of a sequence of states, under the assumption that the probability of transitioning to a state only depends on a fixed length history of the n previous states. This independence assumption allows the history exceeding the order of the Markov chain to be ignored when performing prediction.

In a trivial zero-order Markov model all observations are independent. The simplest non-trivial Markov model conditions the transition probability only on one preceding state.

In a hidden Markov model (HMM) the states are not directly observed. Indirect information about the state sequence is received through the observations, which depend only on the current HMM state through the emission probability. The model depicted in Figure 2.1 on the previous page is an example of an HMM.

The *Viterbi algorithm* is an efficient method for calculating the most probable state sequence for a given sequence of observations, using dynamic programming.

2.1.9 Powell's conjugate direction method

Minimizing a function of multiple variables is a more difficult problem than optimizing a function of one variable. An attractive approach to solving the problem involves reducing it to a sequence of one-variable optimizations. To achieve this, an initial point $\boldsymbol{\theta}_0$ in the N-dimensional space is selected, together with a vector $\boldsymbol{n}$ of length N. N is the number of variables in the function to be optimized. Search along this vector can be parameterized as $f(\lambda) = \boldsymbol{\theta}_0 + \lambda \boldsymbol{n}$, which is a function of one new variable λ.

Once the minimum constrained to the search direction has been selected as the new point $\boldsymbol{\theta}_1$, a new search vector can be chosen and the search repeated. As the optimum of the previous search also lies on the line along which the following search is performed, each iteration is guaranteed not to increase the objective function.

The above description fits several multidimensional optimization methods, which differ only in the manner in which the search vector is chosen.

Powell's conjugate direction method is an optimization method based on this schema of successive line searches, which does not involve computing the gradient of the function. The next search direction is chosen from a set of search directions. The set is initialized to consist of unit vectors parallel to each coordinate axis. The sequence of line searches then begins, with the search vectors chosen from this set in order.

When all vectors in the set of search directions have been used once, the set is modified by replacing one of the vectors. The vector to replace is chosen to be the one along which the objective function made its largest decrease. The vector used as replacement is the *displacement vector*, defined as the difference between the current point and the initial point.

The motivation for this choice of vector to replace is that the best direction of the previous iteration is likely to be a major component of the new direction. Including both vectors in the direction set would only bring the set of directions closer to linear dependence. (Press et al., 1992)

2.2 Linguistic background

The following section gives a brief description of the linguistic concepts most relevant to the task of morphological segmentation. The descriptions are based on the doctoral dissertations of Virpioja (2012) and Creutz (2006).

2.2.1 Structures in language

Natural language contains structure on several levels. A natural starting point is the sentence. The sentence can be divided into phrases, which are sometimes delineated by punctuation, but at other times unmarked.

The phrase can be further divided into words. In phonemic writing systems the boundaries between words are marked with empty space, while in logosyllabic languages, such as Chinese and Japanese, word boundaries are unmarked.

When dividing words into yet smaller components there are two alternative approaches. One is to follow pronunciation, dividing into syllables and further into phonemes. These units only relate to the form of the word, and do not individually carry any meaning.

Alternatively words can be divided into the smallest meaning-bearing components, called *morphemes*.

The set of rules describing how linguistic units are composed is called *grammar*. Grammar is traditionally split into syntax and morphology. *Syntax* describes the rules of constructing utterances from words. The study of the meaning-bearing internal structure of words is called *morphology*. Morphology considers both form (orthography) and meaning (semantics). Analogously to how syntax describes how words may be combined, *morphotactics* describes the rules for combining morphemes into words.

The term *word* is ambiguous. If taken to mean an exact sequence of letters, we use the term *word form* or *surface form*. A second sense links

together different inflections, e.g. **actor** and **actors**, to the same word. This "dictionary form" is called a *lexeme*.

The same division between abstract "dictionary form" (*morpheme*) and concrete surface realization (*morph*), is also made for sub-word units.

A second ambiguity comes from the manner of counting words. If each repetition of the same word form in a corpus is counted, we refer to *word tokens*. We can also collect each distinct word form into a *lexicon*. The size of the lexicon then gives the number of *word types*. The same type/token distinction also applies when counting morphemes or morphs.

2.2.2 Forms of morphological complexity

There is significant variation in the morphological complexity between languages.

The primary division is between *analytic* and *synthetic* languages. Analytic languages isolate each morpheme into a separate word, while synthetic languages allow several morphemes to be combined in a word. The division is not strict, but instead forms a continuum indicated by the average number of morphs per word. Languages that combine morphemes to the extent that the difference between words and sentences becomes blurred are called *polysynthetic*.

Vietnamese is an example of a language in the analytic end of the spectrum, while Turkish and Finnish are considered synthetic. Inuit languages are in the polysynthetic extreme. English is usually considered to be a mixed type between analytic and synthetic.

A second continuum is defined by the interaction between several morphemes occurring in the same word. *Agglutinative* or *concatenative* languages simply concatenate morphemes with minimal modification. In *fusional* languages the morphemes interact, transforming the boundary between them, or morphemes may even be completely overlaid so that a single affix simultaneously marks several morphological aspects.

There are two main types of morphological phenomena: inflectional morphology and word formation. Inflectional morphology, or morphosyntax, describes how a lexeme adapts to its syntactical context, such as agreement with gender, number, case, tense or mood. An example is the suffix **s** in **she write*s***.

Word formation, also known as morphosemantics, describes how new lexemes are formed through compounding, affixation or conversion. The Part-Of-Speech of the new lexeme can be different from the root form. An example is the affixation of the suffix **er** in **writ*er***, which turns the verb **(to) write** into a noun.

In compounding, two or more lexemes are combined into a new lexeme. An example is **smalltalk**, which combines **small** and **talk**.

In conversion the surface form of the word does not change, but the Part-Of-Speech does. In **chair a meeting** the verb **chair** has been formed from a noun through conversion.

Conversion and affixation together are called *derivation*.

A morpheme that can only occur together with some other morpheme is called a *bound morpheme*. A *free morpheme* can form a word on its own.

2.2.3 Models of morphology

There are several linguistic models of morphology. Word-and-paradigm is the oldest and most widely known, commonly encountered e.g. in language education. It is a model of inflectional morphology only. The core concept in the word-and-paradigm model is not the morpheme but instead the paradigm. A paradigm collects words that inflect according to the same pattern, and enumerates all the inflection patterns that can be applied to those words.

The item-and-process model is a versatile model for morphology, in which new lexemes are formed from other lexemes through a transforming process. The first item of a word must always be a free morpheme, to which any number of processes are applied in sequence.

In the item-and-arrangement (IA) model all morphemes in the word are of equal standing. The arrangement of the morphemes is determined by morphotactic constraints.

In this thesis, we use a restricted concatenative item-and-arrangement model in which items are *morphs*, meaning surface realizations of morphemes, which can be concatenated together to form the lexeme. Concatenative IA is unable to fully represent some types of morphology, including allomorphy, non-segmental morphemes (ablaut), root-and-pattern morphology, infixation and circumfixation.

Allomorphs are morphs that share a meaning and occur in complementary distribution, meaning that two different allomorphs of the same morpheme cannot occur in the same context. An example is the past tense suffixes **d** and **ed**, in the words **invite*d*** and **discuss*ed***.

Non-segmental morphemes modify one or several letters in another morpheme, such as **s*i*ng**–**s*a*ng**–**s*u*ng**. Infixes are morphs that are placed inside another morph, while circumfixes consist of two parts that surround another morph.

Root-and-pattern morphology is used in some Semitic languages, in which words are formed by interleaving a pattern of vowels in a root formed of

consonants. Examples of this are **kitāb** (book), **kutub** (books) and **kātib** (writer), which are formed from the root **k–t–b** (write).

The general IA model supports allomorphy and non-segmental morphology by considering the items to be abstract morphemes, which are realized as a specific surface form depending on the arrangement.

2.2.4 Morphological analysis and segmentation

The central task in morphology is producing a morphological analysis of utterances. The words in the utterance are divided into their constituent morphemes, annotated by their semantic and functional characteristics. Full morphological analysis must account for both meaning and form, and must be able to associate different allomorphs to their common abstract source morpheme. (Hammarström and Borin, 2011)

Morphological segmentation is a relaxed variant of morphological analysis, only considering the form aspect and leaving semantics aside. The term *segment* is used to emphasize the substring nature of the produced segments. Morphological segmentation is based on the concatenative item-and-arrangement model of morphology described in Section 2.2.3.

Some methods mentioned in Section 2.4 relax the problem of morphological segmentation by assuming that a word only contains two parts, a stem and a single suffix. This assumption is generally not true even for non-compound words, as it is able to describe neither prefixes nor sequences of multiple suffixes.

A further relaxation of the segmentation task results in stemming, where the objective is to project morphologically related words to the same stem, which does not need to be a true morpheme. No attempt is made to retain information about the differences between the lexemes that result in the same stem. Stemming is useful for example in Information Retrieval, where the inflection of words is often not relevant to the problem. Stemming suffers from a tendency to project unrelated words to the same stem, making them impossible to distinguish. A prominent stemming algorithm is the Porter (1980) stemmer.

In this work a phonemic writing system is assumed, allowing us to focus on segmenting words into morphemes. While segmentation of utterances into words in logosyllabic languages has some similarity to morphological segmentation, the challenges encountered and the methods employed differ significantly enough that limiting the study to morphological segmentation is motivated.

2.3 Evaluation of morphological segmentation

This work studies morphological segmentation in absence of non-concatenative processes. As we have access to linguistic gold standard segmentations in multiple languages, and following previous work (Kurimo et al., 2006; Dasgupta and Ng, 2007) we use direct evaluation that is based on correct positioning of the morph boundaries. The exact evaluation score is the *Boundary Precision and Recall* (BPR) described by Virpioja et al. (2011b). Precision and Recall are calculated for the boundaries between morphs, and defined as

$$\text{Precision} = \frac{|P \cap G|}{|P|}; \;\; \text{Recall} = \frac{|P \cap G|}{|G|}, \tag{2.6}$$

where the set P contains the positions of predicted boundaries and G the gold standard boundaries. In addition we report the F_1-measure, which is the harmonic mean of the Precision and the Recall.

Even though the studied methods only return one analysis for each word, we give each alternative equal weight in the presence of alternative gold standard analyses, instead of only choosing the best match. This is done to make results comparable with previously published results.

BPR was chosen over graph-based assignment algorithms such as EMMA (Spiegler and Monson, 2010) and CoMMA (Virpioja et al., 2011b), due to their much larger computational complexity, which requires sampling a smaller test set. Using the more simple BPR measure makes testing on the full test set feasible.

2.3.1 Indirect evaluation using Information Retrieval

Indirect evaluation can be performed by measuring the performance of an application that uses the method to be evaluated. In the current work we use an Information Retrieval (IR) task as indirect evaluation. The evaluation methodology follows Morpho Challenge 2010 (Kurimo et al., 2010a).

In IR, the user provides a query string, to which the system responds by returning a number of documents considered relevant to the query. The order in which the relevant results are returned is usually also considered to be important.

In the IR experiments, all word forms in both the corpora and the queries were replaced by their respective segmentations. The experiments were performed using Lemur Toolkit (Ogilvie and Callan, 2001) with Okapi BM25 ranking. A stoplist of segments with frequency above 75 000 (Finnish) or 150 000 (English) was used to exclude detrimentally common segments. The evaluation criterion is the Mean Average Precision.

The Average Precision is defined for a ranked list of results to a query, as

$$AP = \frac{1}{R} \sum_{k} \text{Precision}(k)\text{rel}(k), \tag{2.7}$$

where R is the number of relevant documents, k is the rank, Precision(k) is the precision for the list of results truncated after rank k, and rel(k) is 1 if the document at rank k is relevant and 0 otherwise. Mean Average Precision is the mean of Average Precision over all queries.

2.4 Related work on morphological segmentation

There is a long tradition of heavily knowledge-based morphological analyzers, which involve constructing rules for morphological analysis by hand. A very successful paradigm for knowledge-based morphological analyzers is the two-level morphology by Koskenniemi (1984). The term *two-level* comes from only considering an abstract lexical level and a phonological or orthographical surface level. This model can be efficiently implemented as a finite-state transducer (FST).

Hand built morphological analyzers continue to receive much research attention. To name just a few: the efforts of Deléger et al. (2009) to transfer an analyzer from French to English, a freely available FST-based two-level morphological analyzer for Turkish by Çöltekin (2010), and adaptation of an Arabic morphological analyzer to a local language variant by Habash et al. (2012). Pirinen (2011) develops omorfi, an open source Finnish morphological analyzer based on FST. Lindén et al. (2011) present tools for the development of FST-based hand built morphological analyzers.

Hand built morphological analyzers can achieve excellent performance (e.g. the Information Retrieval task in Kurimo et al. (2010b)), but require large amounts of labor by expert linguists to build and adapt to new data, which is a limiting factor in their application to poorly resourced languages.

Machine learning can be used to alleviate the need for expert labor, by automatically inducing morphologies from large text corpora. The following section gives an overview of some previous machine learning approaches to morphological segmentation. The main categorization will be the level of expert knowledge involved in the training.

As described in Section 2.1.3, we call a method *knowledge-based* if it requires substantial expert knowledge of any form. We use the term *knowledge-free* if the method fits the definition of Unsupervised Learning of Morphology by Hammarström and Borin (2011).

2.4.1 Knowledge-free methods

The work of Harris (1955), later continued by Hafer and Weiss (1974), can be considered the starting point for unsupervised morphological segmentation. These methods use letter successor variety (LSV) to guide the choice of morph boundaries. LSV is defined as the number of distinct letters following a given prefix substring, counted from a corpus. Peaks in the LSV indicate potential morph boundaries. These LSV methods have since been reinvented many times and used as heuristics in many later methods.

Schone and Jurafsky (2000) use latent semantic analysis to restrict segmenting to only such splits where the proposed stem is semantically similar to the unsegmented form.

The well-known Linguistica program by Goldsmith (2001) applies MDL to the problem of unsupervised morphological segmentation, but is limited to one stem and one affix per analysis. Linguistica learns so called signatures, which can be likened to statistical paradigms.

Kazakov and Manandhar (2001) use a pipeline approach: first a genetic algorithm is used to segment the corpus, after which segmentation rules are extracted with a first order decision list learner.

Snover et al. (2002) present a generative model in the word-and-paradigm framework. It performs a global search in which hypotheses of paradigm sets are combined, until only one hypothesis remains.

Keshava and Pitler (2006) combine the heuristics mentioned above with a simple punishment and reward algorithm. The RePortS algorithm iterates over a corpus, evaluating all possible affixes in each word using the predictability of the letter immediately before and after the proposed split, given the preceding substring. Only affixes with a positive final score are included in the final morph lexicon. A limitation of the algorithm is that it requires stems to occur as free morphemes.

Dasgupta and Ng (2007) extend the RePortS algorithm by detecting incorrect affix attachments using relative corpus frequency and suffix level similarity. They also induce orthographic transformation rules, which slightly alleviate the requirement of stems occurring as free morphemes.

The method by Demberg (2007) is also based on RePortS, but removes the restriction to free morpheme stems by replacing the stem candidate step of the algorithm.

Bernhard (2008) presents a heuristic method that uses substring predictability in long words to identify affixes, and then strips the found affixes to form list of stems. It imposes morphotactic constraints using four categories: stem, prefix, suffix and linking element.

ParaMor by Monson (2008) finds paradigms by manipulating networks of *schemes*, which describe inflection rules and suffixation. The search uses paradigmatic, syntagmatic and phoneme sequence constraints.

Can and Manandhar (2009) exploit syntactic categories found by induced POS-like tags learned in an unsupervised manner. The method uses a pipeline approach, first inducing the POS-like tags, and then inducing morphological paradigms from the tagged words. The POS induction uses the sentence context in which the words occur.

Naradowsky and Goldwater (2009) formulate a generative model, which is able to learn spelling rules that transform the end of the stem based on two or three characters of context at the morpheme boundary, simultaneously to learning morphological segmentation. The model is limited to a single stem and suffix per word.

Golénia et al. (2009) apply MDL to find pseudo-stems as the first step in a pipeline, followed by graph decomposition for the extraction of paradigms. The method is limited to words with one stem, but can handle any number of prefixes and suffixes.

The Promodes algorithm by Spiegler et al. (2010) uses a generative model with letter features and latent variables for the morph boundaries.

MorphoNet by Bernhard (2010) uses community structures found in a network based on orthographic similarity between word forms. The network connects each word to similar words through transformations in the form of regular expressions. All words in a community can then be formed from the most frequent shortest word in the community by using these transformations. This means that MorphoNet is not limited to concatenative morphology, and places it in the item-and-process model of morphology. MorphoNet has difficulty with compound words.

Lavallée and Langlais (2010) extract formal analogies of the form "x is to y as z is to t". The Rali-Ana algorithm uses these analogies directly for segmentation, while Rali-Cof cluster together morphologically related words to extract segmentation rules.

Lignos (2010) models derivation and inflection as transformation rules applied to a base form. The selection of transforms is informed by the frequency of words it applies to and the amount it changes the words. Three different mechanisms for handling of compound words are evaluated. Chan and Lignos (2010) impose a limitation of monotonic online learning in order to model human learning. The model is based on simple splitting of frequent suffixes, combined with transformations that replace letters from the end of a stem with a found suffix. Only one stem and suffix per word are supported.

Lee et al. (2011) simultaneously induce a POS-like tagging and encourage both morphological consistency within a POS-like category and morpholog-

ical agreement between adjacent words, based on the tagging. It allows up to two prefixes and two suffixes per word, but only one stem. The number of POS-like categories is set to five.

The Morfessor family of methods, into which the method described in the current work also belongs, is inspired by MDL. The history of the Morfessor family is recounted in Section 3.1 on page 24.

2.4.2 Supervised knowledge-based methods

Some early work on automatic induction of morphology uses fully supervised learning, meaning that all the data used in training is labeled. One frequently attempted task is the learning of English past tense forms from a corpus of pairs of past and present tense for each word. Rumelhart and McClelland (1985) use a neural network, while Ling (1994) trains decision trees, and Mooney and Califf (1995) employ logic programming for this task.

Theron and Cloete (1997) propose a system that learns rules for a two level morphology in a supervised manner. Van den Bosch and Daelemans (1999) train a system capable of full morphological analysis including functional labels, using memory-based learning.

Ruokolainen et al. (2013) use conditional random fields, trained on small amounts of labeled data, to predict morph boundaries from the local letter context.

2.4.3 Other knowledge-based methods

In addition to the fully supervised methods already described in the previous section, there are also knowledge-based morphological segmentation methods that are not fully supervised. There are different approaches to combining different types of data, from the use of annotated data to automatically tune parameters, and active learning, to semi-supervised machine learning.

Yarowsky and Wicentowski (2000) learn rules for inflectional morphology by iteratively refining an alignment between roots and inflected forms. The method uses supervision in the form of a list of POS-tags with corresponding canonical suffixes, a list of candidate noun, verb and adjective roots, and a list of consonants and vowels of the language. The lists do not need to be exhaustive and may contain noise. Relative frequency of word forms in the corpus and their contextual similarity are used to rank the alignments.

Oflazer et al. (2001) construct FST-based analyzers through a feedback loop eliciting new data from a user with linguistic expertise.

Snyder and Barzilay (2008) show that morphological structure transferred from another language can be useful when performing morphological

segmentation. They simultaneously learn segmentations and cross-lingual morpheme alignment. The method can use any combination of labeled and unlabeled data in the source and target languages. They use a small data set on the order of 6000 aligned phrase pairs, of which half is annotated.

Monson et al. (2010) trains stochastic taggers to mimic existing morphological analyzers. The benefit comes from the possibility to tune and/or combine several analyzers, using an annotated data set. This tuning approach makes the system knowledge-based, even if the input analyzers are trained in a knowledge-free manner.

While the main result of Poon et al. (2009) is unsupervised, the method can also be used in a semi-supervised or fully supervised manner. The proposed model is a discriminative log-linear model. The log-linear model can use overlapping features that cannot easily be broken down into a series of generative steps. In the experiments using labeled data, a large proportion (25-100%) of the data set is annotated.

Shalonova and Golénia (2010) use a two-stage pipeline approach where stems are first identified using a rule-based system, after which multiple affixes are split with a graph algorithm. The system is semi-supervised using training data where only the stems are annotated.

Tchoukalov et al. (2010) apply Multiple Sequence Alignment (MSA), a method from biological sequence processing, to morphological segmentation. The method can be considered knowledge-based for two reasons: The output alignment of MSA is not as such usable for segmentation, but segmentation points in the produced alignment can be chosen in a supervised manner using a segmented data set. In the experiments the authors use the output of unsupervised ParaMor-Morfessor for this purpose, to keep the complete system unsupervised, but substituting gold standard segmentations would be possible. The second reason to consider the method knowledge-based is that the authors find that results on a large uncurated data set do not reach state-of-the-art levels, but that the method gives better results when trained on a curated corpus of orthographically related words.

Tepper and Xia (2010) post-process the output of Morfessor Categories-MAP using handwritten transformations. The choice of transformation to apply is learned in an unsupervised manner.

Naradowsky and Toutanova (2011) transfer a segmentation from one language to another, using a log-linear hidden semi-Markov model to combine lexical and word dependency features. Even though the title of the paper contains the word "unsupervised", the system needs supervision in the form of segmentations, POS-tags and dependency tree analysis in the source language. On the target side the system is unsupervised.

Chapter 3

Methods

Morfessor is a well-established family of methods for morphological segmentation. The Morfessor methods are formulated in a generative probabilistic framework with inspiration from the Minimum Description Length principle. The Morfessor family is language-independent, but designed to work well on agglutinative languages with long sequences of morphemes.

This chapter begins with a historical perspective to the development of the Morfessor family.

The second section presents a unified mathematical formulation for the Morfessor family. The original formulation was given by Creutz (2006), and was extended by Virpioja (2012). The mathematical notation used in this work follows Virpioja (2012), except where disambiguation has required new notations to be introduced.

The chapter concludes with a brief mathematical presentation of earlier methods in the Morfessor family, building on the unified formulation. While the first section presents the work in a historical context, the third section takes a more formal approach.

3.1 The Morfessor family

Morfessor has been in active development since 2002. Figure 3.1 on the next page shows the influences between Morfessor methods in a visual form. For more detail about the generative framework, see Section 2.1.1. The Minimum Description Length principle is explained in Section 2.1.6. A unified mathematical formalism for the methods in the family can be found in Section 3.2.

Creutz and Lagus (2002) give the original formulation of the method, which was at the time of writing not yet called Morfessor, but instead referred

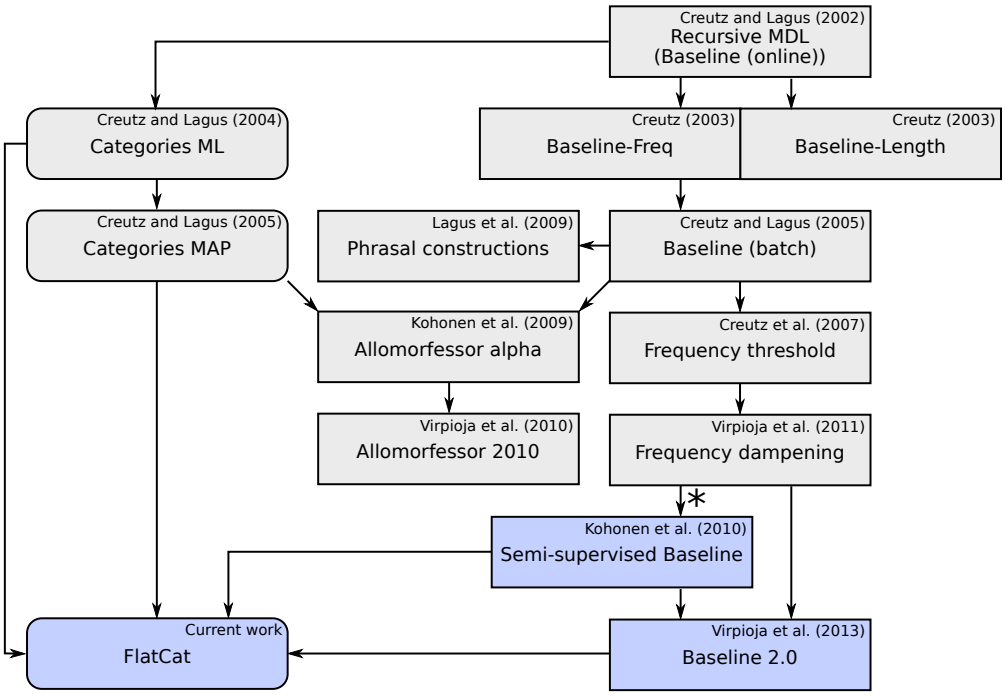


Figure 3.1: The Morfessor family tree. Each node represents a new method or a significant improvement. Nodes with rounded corners represent methods that use morph categories. Nodes with a blue background represent methods that can be trained semi-supervised. Edges represent a significant inspiration between methods. The edge marked with an asterisk reflects the order in which the work was performed, even if the chronology of publication is reversed.

to as Recursive MDL. The model was originally formulated in the Minimum Description Length framework, but can equivalently be formulated as a generative model. Only online learning is supported, meaning that the method has access to training data only one word at a time and does not know the final size of the training data. An update procedure called "dreaming" is repeated at regular intervals. The current Morfessor Baseline is largely based on this model.

The model is reformulated in the Bayesian generative framework by Creutz (2003), which also introduces priors on morph length and frequency. The gamma distribution is used as a prior for the length of morphs, while the distribution for frequency is based on Mandelbrot's correction of Zipf's law. Batch mode learning, where all the data is introduced at the beginning of

training, is now available for the parameter search. The name Morfessor is still not used, but this model variant will later be called Morfessor Baseline-Freq-Length.

A shortfall of the Baseline methods is the assumption of context independence for morphs. This can lead to situations where e.g. a valid suffix is incorrectly used as a prefix. An example of this is segmenting **swing** as **s + wing**. To alleviate this Creutz and Lagus (2004) introduce morph categories. To impose morphotactic rules each morph is assigned one of four categories: prefix, stem, suffix or non-morpheme. A hidden Markov model is used for assigning probabilities to segmentations. Transitions between certain pairs of categories are forbidden. Category membership probabilities based on the length of the morph and the predictability of the context in which it occurs are used for an initial category tagging of the corpus, after which emission probabilities are calculated as maximum likelihood estimates. This method is initially called just Morfessor Categories, but is later known as Morfessor CatML.

Morfessor CatML is a two-stage pipeline system, initialized using a segmentation of the corpus produced by applying the Baseline method. As CatML is a maximum likelihood method, it needs heuristic controls on the lexicon size. The used heuristics are the removal of redundant morphs (the less probable of supermorph or submorphs is removed), forbidding the splitting of morphs into non-morphemes, and finally joining together non-morphemes with their neighboring morphs until they form a stem. The algorithm is divided into four steps: 1) baseline segmentation, 2) initialization of model parameters, 3) removal of redundant morphs, and 4) removal of non-morphemes.

Returning to maximum a posteriori estimation for the model cost, Creutz and Lagus (2005a) present Categories-MAP (CatMAP). Heuristic controls are no longer needed to control the growth of the lexicon. CatMAP also introduces a hierarchical lexicon, intended to reduce the under-segmentation of words with frequently occurring morph sequences. The reduction in lexicon cost caused by the use of hierarchy also makes room for new morphs, which alleviates the over-segmentation of rare words.

The technical report (Creutz and Lagus, 2005b) and the public release of the Morfessor 1.0 Open Source software have undoubtedly contributed to the popularity of Morfessor as a reference method and as preprocessing step in other methods.

Hirsimäki et al. (2006) give a morph frequency prior based on Rissanen's universal prior for non-negative integers. The explicit morph length prior of Baseline-Freq-Length is replaced by an implicit exponential prior from an end-of-morph symbol.

Creutz et al. (2007) introduce word frequency thresholding, in order to restrict the growth of the lexicon as the corpus size increases. This is intended as a mechanism for reducing the sensitivity of the method to the size of unannotated data.

Another motivation for word frequency thresholding stems from that corpora used in unsupervised morphology induction are noisy. Typical errors are misspelled words and words in foreign languages. As these errors typically occur very infrequently, they can be removed by thresholding on the frequency of corpus words. A disadvantage is that the thresholding also removes some correct but rare word forms.

Most methods in the Morfessor family perform a surface segmentation of the corpus, which means that they can only model purely concatenative morphological phenomena. Allomorfessor (Kohonen et al., 2009; Virpioja et al., 2010) is able to model allomorphic variation in stems. Allomorfessor associates a transformation[1] with each morph. The transformation consists of character deletions and substitutions, which are applied to the previous morph scanning from the end toward the beginning of the morph. The transformation may be empty. A modified dynamic programming based algorithm for Levenshtein edit distance is used to calculate the optimal transformation between two strings, yielding potential transformations.

The first Allomorfessor, presented by Kohonen et al. (2009), uses a hierarchical lexicon. Each level of the hierarchical lexicon is called a virtual morph, and consists of a virtual prefix, a transformation, and a virtual suffix. The model has a tendency to under-segment, which is attributed to the hierarchy by Kohonen et al. (2009), and to the independence between transformations and morphs by Virpioja et al. (2010).

An improved version of Allomorfessor is introduced by Virpioja et al. (2010). It abandons the use of hierarchy in the lexicon, with each transformation now affecting the single morph immediately preceding the boundary. The probability of a transformation is conditioned on the morph following the boundary. These changes alleviate the problem of under-segmentation. Despite the usage of transformations remaining lower than linguistic analysis would indicate, the results in the evaluation are good.

Lagus et al. (2009) present a Morfessor-inspired model for learning phrasal constructions, and apply it to a Finnish language corpus of stories told by children. While Morfessor models have typically been applied to segmenting words into morphs, this model instead segments utterances into phrasal constructions.

[1] In the original paper these are called *mutations*, but following Virpioja (2012) and established terminology we use *transformation*.

A new feature of the model is the possibility to define slots[2] in the constructions, which may be filled by any word or word sequence from a set of alternatives.

Up to this point Morfessor models have been unsupervised. The motivation for this choice is that producing enough annotated data for supervised learning requires too much expensive expert labor to produce, and must be created for each language separately. Kohonen et al. (2010) argue that using small amounts of annotated data in a semi-supervised manner is acceptable. They show that an annotated data set that is much too small for supervised learning, when combined with an unannotated data set, can lead to significantly improved results over the unsupervised method.

When using comparison against a linguistic gold standard segmentation, Morfessor models in most cases perform better using word types than using word tokens (Creutz and Lagus, 2005b; Virpioja, 2012). Frequencies of words should, however, be useful at least for some applications of morphology, to allow a more concise representation of the most common words. Virpioja et al. (2011a) introduce an arbitrary transformation for the frequencies. The special case using a constant function $d(x) = 1$ is equivalent to training on word types, while using the identity function $d(x) = x$ trains on word tokens. It is also possible to use an intermediary form, e.g. logarithmic dampening, to achieve a compromise between the types and tokens.

As this chapter shows, significant improvements extending Morfessor Baseline have been published since the release of Morfessor 1.0 (Creutz and Lagus, 2005b). In order to incorporate these improvements, and to address some technical aspects in which Morfessor 1.0 has become outdated, such as the lack of support for Unicode character encodings, a new version designated Morfessor 2.0 was released by Virpioja et al. (2013).

3.2 Unified mathematical formulation of Morfessor

Morfessor is a family of generative models for morphological segmentation. Being generative models, they describe the joint distribution of words and their analyses $P(W, A \mid \boldsymbol{\theta})$. The words W are observed, but the analyses A are a latent variable in the model. The latent variable A takes on values

$$\boldsymbol{a} = (m_1, \ldots, m_{|\boldsymbol{a}|}) \tag{3.1}$$

[2]In the paper these are called *categories*, but we use *slot* to avoid confusion with morph categories.

that consist of a sequence of morphs. The lexicon of used morphs is encoded in the model parameters $\boldsymbol{\theta}$.

The model is trained using data $\boldsymbol{D}$, which can be divided into unannotated $\boldsymbol{D}_W$ and annotated $\boldsymbol{D}_A$. The unannotated training data

$$\boldsymbol{D}_W = \left[(w_1, f_1), \ldots, (w_{|\boldsymbol{D}_W|}, f_{|\boldsymbol{D}_W|})\right] \tag{3.2}$$

consists of pairs of the word type and the number of times it occurred in the corpus. The annotated data additionally contains a set of alternative correct analyses for the word form.

3.2.1 Parameter estimation

It is difficult to estimate the complete posterior distribution $P(\boldsymbol{\theta} \mid \boldsymbol{D})$, due to the large number of parameters in the model. For this reason, a point estimate for the parameters is used instead. Both maximum likelihood (ML) and maximum a posteriori (MAP) estimates are used in Morfessor family models. We will here consider only MAP estimation, as ML estimation can be seen as a special case of MAP estimation using a constant prior.

In Morfessor, MAP estimation is preferred over ML, which only considers the accuracy of the model's description of the training data without regard to the model complexity. ML estimation is known under some conditions to require heuristic measures to control overfitting. For more on this topic, see Section 2.1.4.

The MAP estimate for the parameters is given by maximizing the posterior probability

$$\begin{aligned} \boldsymbol{\theta}^{\text{MAP}} &= \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{\theta} \mid \boldsymbol{D}) \\ &= \arg\max_{\boldsymbol{\theta}} \left(P(\boldsymbol{\theta})\, P(\boldsymbol{D} \mid \boldsymbol{\theta})\right) \end{aligned} \tag{3.3}$$

3.2.2 Likelihood

Morfessor assumes that the probabilities of words are independent of each other. No dependencies between words on the sentence level or above are modeled.

The independence assumption allows writing the likelihood of the corpus $\boldsymbol{D}_W$ in product form

$$P(\boldsymbol{D}_W \mid \boldsymbol{\theta}) = \prod_{j=1}^{|\boldsymbol{D}_W|} f_j P(W = w_j \mid \boldsymbol{\theta}). \tag{3.4}$$

The probability of a word can be written as the joint probability of words and analyses marginalized over all analyses $\mathcal{A}$

$$P\left(\boldsymbol{D}_W \mid \boldsymbol{\theta}\right) = \prod_{j=1}^{|\boldsymbol{D}_W|} f_j \sum_{\boldsymbol{a} \in \mathcal{A}} P\left(W = w_j, A = \boldsymbol{a} \mid \boldsymbol{\theta}\right). \tag{3.5}$$

The joint probability of words and analyses can be rewritten using the chain rule

$$P\left(W = w, A = \boldsymbol{a} \mid \boldsymbol{\theta}\right) = P\left(A = \boldsymbol{a} \mid \boldsymbol{\theta}\right) P\left(W = w \mid A = \boldsymbol{a}, \boldsymbol{\theta}\right), \tag{3.6}$$

The probability of a word is zero given most analyses, because they are analyses for some different string of letters. Introducing the detokenization function ϕ^{-1}, which maps an analysis to the resulting word form, and using the indicator function I to select only relevant analyses, the joint probability becomes

$$P\left(W = w, A = \boldsymbol{a} \mid \boldsymbol{\theta}\right) = P\left(A = \boldsymbol{a} \mid \boldsymbol{\theta}\right) \mathrm{I}\left(\phi^{-1}(\boldsymbol{a}, \boldsymbol{\theta}) = w\right). \tag{3.7}$$

This equation yields two of the defining components for a member of the Morfessor family:

1. the probability of an analysis $P\left(A = \boldsymbol{a} \mid \boldsymbol{\theta}\right)$,
2. the detokenization function ϕ^{-1}.

The remaining components are related to the form of the prior $P\left(\boldsymbol{\theta}\right)$, defined by

3. the properties of morphs being stored in the lexicon,
4. and the parameters of the model grammar.

The latter two will be described in detail in the next section.

3.2.3 Prior

Morfessor models typically use MDL derived priors, which control the number of non-zero parameters to prevent overfitting, similarly to regularization. These sparse priors are non-informative except for a preference for efficient compression. The exception is CatML, which uses maximum likelihood estimation and thus omits the prior.

In the general form the prior $P\left(\boldsymbol{\theta}\right) = P\left(\mathcal{L}\right) P\left(\mathcal{G}\right)$ consists of two parts: the priors for the lexicon $\mathcal{L}$ and the grammar $\mathcal{G}$. For most Morfessor models

the number of parameters in the grammar is fixed and relatively small, allowing the prior for the grammar to be omitted and reducing $P(\boldsymbol{\theta})$ to $P(\mathcal{L})$.

Let μ be the number of morphs with non-zero parameters, in other words the number of morph types stored in the lexicon. The typical Morfessor lexicon prior consists of three parts

$$P(\mathcal{L}) = P(|\mathcal{L}| = \mu) \times P\big(\text{properties}(m_{(1)}), \ldots, \text{properties}(m_{(\mu)})\big) \times \mu!. \tag{3.8}$$

The first term is the prior for the lexicon size. It is often omitted as it is a single probability term of negligible effect.

The second term is the probability of generating morphs with a certain set of properties. The choice of properties affects the possible likelihood functions. The properties are divided into those related to the form of the morph and those related to the usage of the morph, for the sake of clarity when explaining the models.

The main distinction in the form properties is between flat and hierarchical lexicons. In a hierarchical lexicon, a morph can consist of submorphs, which introduces hierarchical references in the lexicon. In a flat lexicon a morph must always be spelled out.

Examples of form properties are the letters spelling out the morph or the submorphs of which the morph is built. Usage properties include occurrence counts of morphs and perplexity measures describing the predictability of the context in which the morph occur.

We denote the index of a morph type in the lexicon with parentheses around the index ($m_{(i)}$) to distinguish it from the position of a morph token in a (sub-sequence) of the corpus (m_i).

The factorial term arises from the fact that two lexicons with the same morphs drawn in different order are equivalent.

3.2.4 Training algorithms

Training algorithms used in Morfessor methods vary, but a common characteristic is that they use local search to optimize the model parameters.

While the most straightforward approach would be to define the search space over the parameters $\boldsymbol{\theta}$ only, and to marginalize over all possible analyses during the search, this approach is generally infeasible.

The search can be made feasible by selecting the most probable analysis as a point estimate. Let $\boldsymbol{Y}$ be a random variable that selects, for each word in the corpus, one of the possible analyses as the active one. The search space now consists of analyses $\boldsymbol{Y}$ and parameters $\boldsymbol{\theta}$ consistent with the analysis. The details of the search depend on the Morfessor method.

In unsupervised training the cost function L is the negative logarithm of the likelihood conditioned on the currently active analysis

$$\begin{aligned} L(\boldsymbol{\theta}, \boldsymbol{Y}, \boldsymbol{D}_W) &= -\log P(\boldsymbol{\theta}) - \log P(\boldsymbol{D}_W \mid \boldsymbol{Y}, \boldsymbol{\theta}) \\ &= -\log P(\boldsymbol{\theta}) - \sum_{j=1}^{|\boldsymbol{D}_W|} f_j \log P(\boldsymbol{Y}_j \mid \boldsymbol{\theta}) \end{aligned} \tag{3.9}$$

3.2.5 Tokenization and detokenization functions

The detokenization function ϕ^{-1} maps the morph sequence forming an analysis to the surface form of a word. In most Morfessor variants ϕ^{-1} is simply a concatenation of the morphs, with the notable exception of Allomorfessor. The detokenization function of Allomorfessor is described in Section 3.3.3.

The tokenization function ϕ, which maps words to their constituent morphs, is more complex. The tokenization function is used both during the training to reanalyze words, and after the model has been trained to analyze unseen data.

Morfessor variants typically use a generalized Viterbi algorithm for the tokenization, where the hidden states are morphs and the observations are letters. The Viterbi algorithm is modified by allowing states to span an arbitrary number of observations, instead of restricting the number of states to equal the number of letters. An explanation of the generalized Viterbi algorithm is given by Virpioja (2012).

Morfessor variants without morph categories use a degenerate state space with only one category, so that only the number of states and the extent of each state matters. This results in a time complexity of $\mathcal{O}(|w|^2)$. In Morfessor variants that use morph categories the state space is defined by the categories. In a model with K categories, this increases the time complexity to $\mathcal{O}(|w|^2 K^2)$.

In Morfessor variants with morph categories the (non-generalized) Viterbi algorithm is also used, when the categories of an analysis need to be reoptimized, without changing the segmentation.

3.2.6 Likelihood weighting and semi-supervised training

In Morfessor methods that use flat lexicons, the cost function divides into two parts that have opposing optima. Minimizing only the cost of the data (the likelihood), results in choosing a lexicon that consists of the words in the training data with minimal segmentation of the corpus. On the other

hand the cost of the model (the prior) is optimal when the lexicon consists only of the letters, and the corpus is maximally segmented.

Adjusting the relative weights of the parts of the cost function controls the amount of segmentation in a straightforward way.

Another way of describing this weighting is as a scaling of the counts of word tokens f_j with a factor α

$$L(\boldsymbol{D}_W; \boldsymbol{\theta}) = -\sum_{j=1}^{|\boldsymbol{D}_W|} \alpha f_j \log \sum_{\boldsymbol{a}} P\left(W = w_j, A = \boldsymbol{a} \,|\, \boldsymbol{\theta}\right). \tag{3.10}$$

A knowledge-based (but not semi-supervised) method can be achieved by using a held-out development set for tuning α. This type of training is called *likelihood weighted learning* to distinguish it from semi-supervised. However, better results can be achieved if enough annotated data is available to use also as labeled training data in semi-supervised learning.

When using *semi-supervised learning*, the likelihood of the annotated corpus $\log P\left(\boldsymbol{D}_A \,|\, \boldsymbol{\theta}\right)$ is added to the cost function. The counts in the annotated corpus can be scaled by a different weight, β. This makes it possible to compensate for the typically much smaller amount of annotated data compared to the unannotated data. The development set should then be used also for tuning β.

Moving α and β outside of the summation over the data set results in the weighting scheme presented by Kohonen et al. (2010). Instead of optimizing the MAP estimate, the cost function is

$$\begin{aligned} L(\boldsymbol{\theta}, \boldsymbol{D}_W, \boldsymbol{D}_A) = &- \log P\left(\boldsymbol{\theta}\right) \\ &- \alpha \log P\left(\boldsymbol{D}_W \,|\, \boldsymbol{\theta}\right) \\ &- \beta \log P\left(\boldsymbol{D}_A \,|\, \boldsymbol{\theta}\right). \end{aligned} \tag{3.11}$$

This weighting scheme cannot be used in methods with a hierarchical lexicon. The reason is that the probability of a morph is based on the number of references to it, and both the occurrences in the corpus as well as those in the hierarchical lexicon must be counted. Moving some of the segmentation into the lexicon subverts the use of α to control segmentation.

Implementing the semi-supervised learning together with a hierarchical lexicon by simply adding the likelihood of the annotated data is also not effective. The annotations give no direction to the choice of when to use hierarchy.

3.2.7 Frequency thresholding and dampening

Virpioja et al. (2011a) introduce a function d that extends the likelihood weighting transformation with two other purposes: frequency thresholding and dampening.

$$L(\boldsymbol{D}_W; \boldsymbol{\theta}) = -\sum_{j=1}^{|\boldsymbol{D}_W|} d(f_j) \log \sum_{\boldsymbol{a}} P\left(W = w_j, A = \boldsymbol{a} \,|\, \boldsymbol{\theta}\right) \tag{3.12}$$

$$d(x) = \begin{cases} 0 & \text{if } x < T \\ g(x) & \text{otherwise,} \end{cases} \tag{3.13}$$

where T is a frequency threshold, and g is an arbitrary function mapping from counts to a non-negative integer.

Frequency thresholding removes the contribution of any word with frequency less than T, by setting the modified count to zero.

The function g performs dampening on the counts. Using the constant function $g(x) = 1$ is equivalent to training on word types, while using the identity function $g(x) = x$ trains on word tokens. It is possible to use an intermediary form, e.g. logarithmic dampening $g(x) = \lceil \ln(1 + x) \rceil$.

Virpioja et al. (2011a) also included the corpus likelihood weight α in the dampening function (Equation 3.13). We have moved it outside the summation over the data set, into the cost function (Equation 3.11). The two formulations are equivalent.

3.3 Previous Morfessor methods

In this section, we will briefly present earlier methods in the Morfessor family. The descriptions will be limited to the ways in which the methods differ from the unified method description.

3.3.1 Morfessor Baseline

Morfessor Baseline (Creutz and Lagus, 2002, 2005b; Virpioja et al., 2013) is the branch of the Morfessor family with the most simplifying assumptions. The morphs of a word are assumed to occur independently. No morph categories are used, which can be made to fit the unified framework presented in Section 3.2 by treating it as a degenerate case with only one category.

These assumptions simplify the probability of an analysis, which forms the basis of the likelihood

$$P\left(A = \boldsymbol{a} \mid \boldsymbol{\theta}\right) = P\big(\#_w\big) \prod_{i=1}^{|\boldsymbol{a}|} P\left(m_i \mid \boldsymbol{\theta}\right), \tag{3.14}$$

where $\#_w$ is the word boundary symbol. The original Morfessor Baseline formulation (Creutz and Lagus, 2002) did not include word boundary symbols, which were introduced in Virpioja et al. (2013). The use of a boundary symbol introduces an implicit exponential length prior, which replaces the need for an explicit length prior.

The detokenization function ϕ^{-1} is simply the concatenation of the participating morphs.

The properties encoded in the lexicon are also straightforward. Morph frequency is the only morph usage property. The lexicon is flat, meaning that morphs have no substructure. The form properties of a morph are thus the string of letters to spell it out.

The search algorithm is based on recursive reanalysis of each word in turn. The assumption that morphs occur independently allows a compact representation of segmentations as a binary directed acyclic graph. The top nodes are entire words and the bottom nodes are real morphs used in the segmentation. All remaining internal nodes are virtual morphs, which have no significance outside the training algorithm.

For a complete description of the search algorithm used by Morfessor Baseline, see Creutz and Lagus (2005b).

3.3.2 Morfessor Categories-ML and Categories-MAP

The Categories variants of Morfessor use a hidden Markov model (HMM) for the morphotactics.

A brief description of the morph properties will be given in Chapter 4. Creutz and Lagus (2005a) describes the properties in depth.

Morfessor Categories-MAP (Creutz and Lagus, 2005a) uses a hierarchical lexicon, where a morph can be represented as a concatenation of two other morphs. The prior for the form properties is defined as

$$P\big(\mathrm{form}(m_{(i)})\big) = \begin{cases} \big(1 - P\left(\boldsymbol{\sigma}\right)\big) \prod_{j=1}^{|\boldsymbol{\sigma}_i|} P\left(\boldsymbol{\sigma}_{ij}\right) & \text{Spelling out} \\ P\left(\boldsymbol{\sigma}\right) P\left(c_{i1} \mid \boldsymbol{\sigma}\right) P\left(m_{i1} \mid c_{i1}\right) P\left(c_{i2} \mid c_{i1}\right) P\left(m_{i1} \mid c_{i1}\right) & \text{Substructure} \end{cases} \tag{3.15}$$

where $P(\boldsymbol{\sigma})$ is the probability that a morph has substructure, $\boldsymbol{\sigma}_{ij}$ is the j^{th} letter in the i^{th} morph, and m_{i1} and m_{i2} are the two submorphs in order.

The detokenization function ϕ^{-1} is simply the concatenation of the surface forms of participating morphs, ignoring the categories.

Morfessor Categories-ML (Creutz and Lagus, 2004) omits the prior entirely, instead relying on heuristics to control model complexity.

3.3.3 Allomorfessor

Allomorfessor moves beyond concatenative processes to model allomorphic variation in stems. Allomorfessor alpha, which used a hierarchical lexicon, is here considered to be superseded by Allomorfessor 2010.

In Allomorfessor, each morph is associated with a potentially empty transformation. The transformation affects a single morph immediately preceding the associated morph. The probability of a transformation is conditioned on the associated morph.

The likelihood of an analysis is given by the joint probability of the morphs and transformations

$$P(A = \boldsymbol{a} \mid \boldsymbol{\theta}) = \prod_{i=1}^{|\boldsymbol{a}|} P(\delta_i \mid m_i) P(m_i), \tag{3.16}$$

where δ_i is the transformation associated with morph m_i. Allomorfessor does not include the boundary symbol in the likelihood.

The detokenization function first transforms each morph by applying the transformation associated with the following morph, and then concatenates the resulting surface forms. For example in the analysis

$$\boldsymbol{a} = \Big[(\epsilon, \text{pretty}), \big((\text{y} \mid \text{i}), \text{er}\big)\Big]$$

the first morph **pretty** is transformed into **pretti** by the transformation (y | i), resulting in the detokenized word **prettier**. The transformation affects **pretty**, but its likelihood is conditioned on **er**.

The grammar in other Morfessor methods has been simple enough to omit. Allomorfessor encodes the set of used transformations Δ in the grammar, giving an additional prior

$$P(\mathcal{G}) = P(|\Delta|) \times P\big(\text{properties}(\delta_1) \ldots \text{properties}(\delta_{|\Delta|})\big) \times |\Delta|! \tag{3.17}$$

The properties for morphs are the same as in Morfessor Baseline.

Chapter 4

Morfessor FlatCat

This chapter presents Morfessor FlatCat, the main contribution of this work. Morfessor FlatCat combines components from two existing methods: Morfessor Baseline and Morfessor Categories-MAP, resulting in the first Morfessor method that both contains morphotactic constraints and can be trained in a semi-supervised manner.

In addition to sharing some components with Baseline, FlatCat also uses Baseline as a preprocessing step. The standard workflow can be seen in Figure 4.1 on the following page. FlatCat is initialized using a segmentation produced by Baseline, which can be trained with or without annotated data. The Baseline model trained in step (1) is used in step (2) to segment the training data. This Baseline segmentation, optionally together with the annotated data, is used as input to FlatCat training in step (3). The parameters of the trained model can finally be used in step (4) to analyze new data.

4.1 Aims and design criteria

There are three central assumptions on which Morfessor FlatCat is founded. The first assumption is that words in a sentence are independent, allowing the sentence context in the corpus to be discarded. This assumption is not true for natural language, but it does allow simplifying the method significantly.

The second assumption is that a morphology consists only of concatenative processes. This means that the surface form of the word can be retrieved by simply concatenating the component morphs. As result of this assumption allomorphs are regarded as separate morphemes.

The third assumption is that each morph is associated with a category from the set {*prefix, stem, suffix, non-morpheme*}, and that there is no lex-

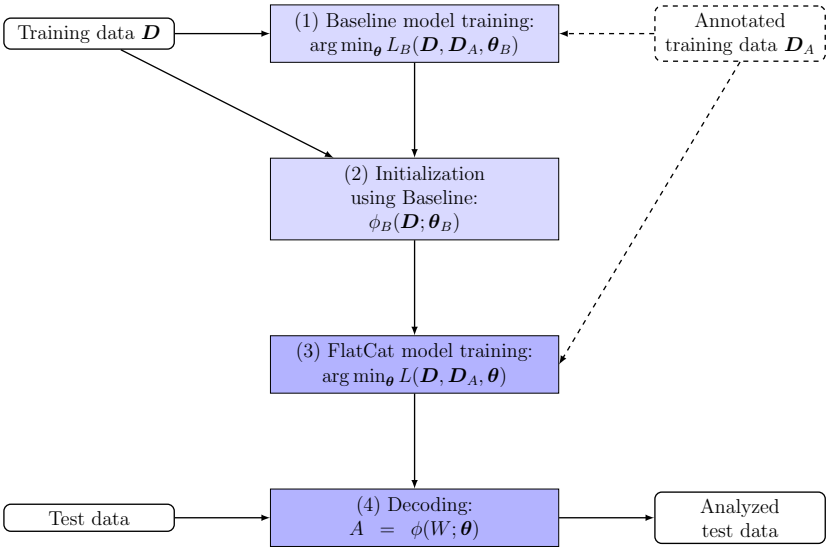


Figure 4.1: The standard workflow for Morfessor FlatCat.

ical dependence between consecutive morphs, given the sequence of morph categories. This is also a simplifying assumption that allows the use of an HMM to estimate the probability of a morph sequence.

An important design criterion for FlatCat was that both unsupervised and semi-supervised training of the model should be possible. The model should also contain weighting parameters useful for optimizing the segmentation for a particular problem. These weighting parameters, with values chosen by using an annotated development set, control the balance between Precision and Recall.

4.2 Model components

The cost function used in Morfessor FlatCat training is

$$\begin{aligned} L(\boldsymbol{\theta}, \boldsymbol{D}_W, \boldsymbol{D}_A) = & -\log P(\boldsymbol{\theta}) \\ & -\alpha \log P(\boldsymbol{D}_W \mid \boldsymbol{\theta}) \\ & -\beta \log P(\boldsymbol{D}_A \mid \boldsymbol{\theta}) . \end{aligned} \tag{4.1}$$

The second and third terms are the likelihoods of the unannotated and annotated corpora. To make the likelihoods feasible to calculate, conditioning on the current analysis $\boldsymbol{Y}$ is used. With this conditioning, the likelihoods are calculated as probabilities of morph sequences.

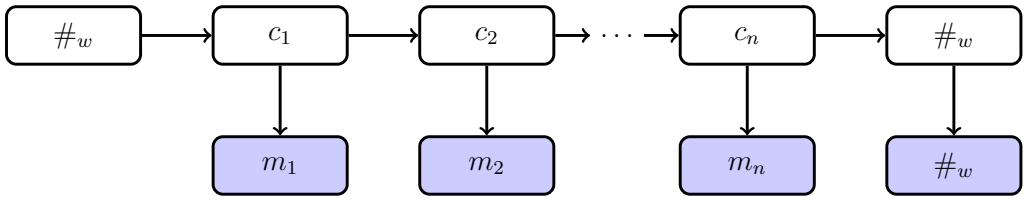


Figure 4.2: The hidden Markov model used for estimating the probability of an example morph sequence of length n, unrolled into a Bayesian network.

Morfessor FlatCat estimates the probability of a morph sequence using an HMM. The hidden states c are defined to be the four morph categories (prefix, stem, suffix and non-morpheme) and a word boundary state ($\#_w$). The probability of a single analysis is given by

$$P\left(A = \boldsymbol{a} \,|\, \boldsymbol{\theta}\right) = P\left(c_1 \,|\, \#_w\right) \prod_{i=1}^{|\boldsymbol{a}|} \left[P\left(m_i \,|\, c_i\right) P\left(c_{i+1} \,|\, c_i\right)\right] P\left(\#_w \,|\, c_{|\boldsymbol{a}|}\right). \quad (4.2)$$

Figure 4.2 shows a Bayesian network representation for the probability of a morph sequence of a particular length. The sequence of hidden states begins and ends with a word boundary state $\#_w$, and otherwise consists of categories c_i. Each observation m_i is the character string representation of a single morph. The final boundary state can only emit a boundary marker with probability 1.

The transition probabilities are estimated from the current analysis of the corpus. Some transitions are, however, forbidden: a suffix may not directly follow the initial word boundary, and a prefix may not be directly followed by a word boundary. A transition directly from a prefix to a suffix is also forbidden. All other transitions are allowed.

The morph properties encoded in the lexicon are a combination of Baseline and CatMAP properties. The morph usage properties are the same as in CatMAP, but the morph form properties define a flat lexicon as in Baseline, instead of containing a hierarchy of submorphs.

A flat lexicon was chosen because of the challenges of combining a hierarchical lexicon with semi-supervised learning and likelihood weighting. Using a flat lexicon, the emission probabilities of a morph can be estimated directly from the morph references in the current analysis of the corpus. If hierarchy is introduced, also the references in the lexicon add to the reference counts. This means that for a flat lexicon, the cost function divides into two parts

with opposing optima that can be weighted to achieve balance. For a hierarchical lexicon this division is muddled by the substructures.

Even though the properties are the same as in previous Morfessor methods, a brief description is given here.

The properties must contain information that allows the categories of morphs to be estimated. The estimated emission probabilities are based on two intuitions: stems are assumed to be generally longer than affixes, while affixes are assumed to be identifiable from the fact that they are used with many different stems. If the predecessor (for suffixes) or successor (for prefixes) is difficult to predict, then a morph is more likely to be an affix.

The usage properties are the intra-word right and left perplexity of the morph, and the occurrence counts of the morph with each category. The right perplexity of the i^{th} morph in the lexicon is defined as

$$\text{right-ppl}(m_{(i)}) = \left(\prod_{m_{(j)} \in \text{right-of}(m_{(i)})} P\left(m_{(j)} \,|\, m_{(i)}\right) \right)^{-\frac{1}{\tau_{(i)}}}, \tag{4.3}$$

where the product is taken over morphs immediately following $m_{(i)}$ in the current segmentation of the corpus, and $\tau_{(i)}$ is the total number of corpus occurrences of the i^{th} morph. The perplexity is estimated from the corpus. Left perplexity is analogous to right perplexity.

The conditional probabilities $P\left(c_i \,|\, m_i\right)$ of a morph belonging to a category are calculated from the morph properties. First stem-likeness, prefix-likeness and suffix-likeness are calculated by using sigmoidal functions to softly threshold the length, right-perplexity and left-perplexity respectively.

The sigmoid function for prefix-likeness is

$$\text{prefix-like}(m_{(i)}) = \Big(1 + \exp\Big[-a \times \big(\text{right-ppl}(m_{(i)}) - b\big)\Big]\Big)^{-1}, \tag{4.4}$$

where b is a threshold value and a is a parameter controlling the steepness of the sigmoid. Prefix-likeness and suffix-likeness are calculated analogously. The sigmoid maps the properties to the range from 0 to 1.

The non-morpheme-likeness is assigned a high value only if all other likeness values are low. As a morph can only belong to one category, the likeness values must be normalized in order to get probabilities.

The emission probabilities can then be estimated from the conditional class probabilities and the occurrence counts using Bayes' formula

$$P\left(m_i \,|\, c_i\right) = \frac{P\left(m_i\right) P\left(c_i \,|\, m_i\right)}{P\left(c_i\right)} \tag{4.5}$$

Because of the flat lexicon the form properties of a morph are always the string of letters to spell it out. The probability of generating the form properties of a morph is thus

$$P\big(\text{form}(m_{(i)})\big) = P\big(\#_m\big) \prod_{j=1}^{|\boldsymbol{\sigma}_i|} P\left(\boldsymbol{\sigma}_{ij}\right), \tag{4.6}$$

where $\boldsymbol{\sigma}_i$ is the string representation of the i^{th} morph in the lexicon, and $\#_m$ is a boundary symbol used to separate morphs. Letter probabilities are estimated from the current lexicon.

As in most other Morfessor methods, the detokenization function ϕ^{-1} is a simple concatenation of the morphs. The morph categories do not affect the detokenization function.

4.3 Training procedure

The FlatCat training procedure consists of a local search, which at each step chooses the best of several alternative analyses for a part of the corpus and updates the parameters accordingly.

The alternative modifications to the analysis and parameters are generated by applying a search operator, currently *split, join, shift* or *resegment*.

The general form of the neighborhood function is

$$\textsc{Op} : (\boldsymbol{Y}, \Theta) \mapsto (\boldsymbol{Y}, \Theta)^n, \tag{4.7}$$

where n is the number of neighbors. For description of FlatCat it is useful to introduce the concept of *unit* to divide the search operator further into two parts: selection of the next unit to optimize, and generating alternatives for the selected unit.

The unit defines which parameters are optimized at a certain step. The parameters are chosen so that they describe the way a small part of the corpus is segmented.

The algorithm then tries to simultaneously find the optimal segmentation for the unit and the optimal parameters consistent with that segmentation

$$(\boldsymbol{Y}^{(t)}, \boldsymbol{\theta}^{(t)}) = \underset{\textsc{Op}\left(\boldsymbol{Y}^{(t)}, \boldsymbol{\theta}^{(t)}\right)}{\arg\min} \big\{L(\boldsymbol{\theta}, \boldsymbol{Y}, \boldsymbol{D}_W)\big\}. \tag{4.8}$$

Units vary in complexity, from all occurrences of a certain morph to the occurrences of a morph bigram whose surrounding context fills certain criteria. The order in which units are processed also varies between frequency-based and length-based ordering.

4.3.1 Neighborhood function

Alternative segmentations for the selected unit are generated in a systematic manner. New morphs may be introduced in this process. The alternative only specifies a specific change in the segmentation: each word containing the unit must also be retagged with morph categories. Each word is individually retagged, with all categories outside the unit frozen to their current values. The retagging is done after the morph counts have been updated, in order to allow new morphs to be appropriately used. The emission and transition counts are updated after retagging.

This procedure differs from CatMAP, in which all instances of the unit are modified so that they receive the same categories. CatMAP must choose the tried categories in such a way that impossible transitions are not introduced.

Applying one of the operators to the entire corpus is called an iteration. The same operator may be applied for several iterations before going over to the next operator. An epoch is reached when each operator in the training sequence has been applied.

Convergence testing to end the training procedure is performed only between epochs.

A high-level overview of the training algorithm as pseudocode can be seen in Algorithm 1.

Algorithm 1 Top-level training algorithm.

function TRAIN($\boldsymbol{D}_W$)
  $\boldsymbol{Y}_{\text{seg}} \leftarrow$ MORFESSORBASELINE($\boldsymbol{D}_W$)
  $\boldsymbol{\theta}, \boldsymbol{Y} \leftarrow$ INITMODEL($\boldsymbol{Y}_{\text{seg}}$)
  for $i \in 1 \ldots I_{\text{epochs}}$ **do**
    for OP $\in \{$SPLIT, JOIN, SHIFT, RESEGMENT$\}$ **do**
      for $j \in 1 \ldots I_{\text{iterations,Op}}$ **do**
        $(\boldsymbol{Y}, \boldsymbol{\theta}) \leftarrow \arg\min_{\text{OP}(\boldsymbol{Y},\boldsymbol{\theta})} \{L(\boldsymbol{\theta}, \boldsymbol{Y}, \boldsymbol{D}_W)\}$
          ▷ Operators loop over units of different size
      end for
    end for
    if $L(\boldsymbol{D}, \boldsymbol{\theta}_i, \boldsymbol{Y}_i) > L(\boldsymbol{D}, \boldsymbol{\theta}_{i-1}, \boldsymbol{Y}_{i-1}) - l_{\text{threshold}}$ **then**
          ▷ Test for convergence
      break
    end if
  end for
  return $\boldsymbol{\theta}$, $\boldsymbol{Y}$
end function

4.3.2 Training operators

The training operators define the neighborhood of the current model state, in other words the possible changes to the model tried by the local search.

The first operator to be performed is *splitting*. The unit for the split operator is a single morph type. The split operator simultaneously targets all occurrences of a specific morph in the corpus, attempting to split them into two parts. All splits into two parts are compared against not splitting the morph at all. The whole corpus is processed by sorting the morphs currently in the lexicon by length from shortest to longest.

The second operator attempts to *join* morph bigrams. Occurrences of morph bigrams are grouped by the type of context they occur in. The four possible context types are word initial, word final, word internal and both initial and final. Morphs are considered word initial if they are preceded by a word boundary or a prefix, and word final if followed by a word boundary or a suffix. The context specific bigram counts are sorted by frequency, from most to least common.

The third operator is the *shifting* of the boundary between morph bigrams. It is context sensitive in the same way as joining, and the bigrams are sorted in the same way. Limits are imposed on the maximum distance for moving the boundary and on the minimum number of letters in the remaining morphs, to avoid pathological changes. In the experiments we set both of these limits to 2 letters.

The reason for introducing the shift operator is to allow an incorrectly placed boundary to be moved without needing to first rejoin the parts and then split at the correct position. The two-step change will only succeed if the intermediary state is better than the initial one.

Finally, *resegmenting* uses the generalized Viterbi algorithm to find the currently optimal segmentation for one whole word at a time. This operator targets each word in the corpus in increasing order of frequency.

4.3.3 Operators applied to an example corpus

In the following examples, the corpus consists of the analyses

{ **ageless/STM**,
ageless/STM + ly/SUF,
ageless/STM + ness/SUF,
critical/STM + ly/SUF,
and **un/PRE + critical/STM + ly/SUF** }.

The units generated from this corpus by each training operator can be seen in Table 4.1 on the next page. All the context-dependent bigrams are considered

Table 4.1: Examples of units selected by different search operators. In the context-dependent morph bigrams, INITIALANDFINAL stands for the context type *both initial and final*.

Operator		Units	
Split	{	**ly**, **un**, **ness**, **ageless**, **critical**	}
Join & Shift	{	(**critical**, **ly**, INITIALANDFINAL), (**ageless**, **ly**, INITIALANDFINAL), (**ageless**, **ness**, INITIALANDFINAL), (**un**, **critical**, INITIALANDFINAL)	}
Resegment	{	**ageless**, **agelessly**, **agelessness**, **critically**, **uncritically**	}

both word initial and final, including the cases where the bigram does not cover the entire word but is preceded by a prefix or followed by a suffix.

When the split operator targets the unit **ageless**, all splits into two parts are tried: **a + geless**, **ag + eless**, ..., **ageles + s**. The split is applied to each occurrence of the morph. The words in which the morph occurs are then retagged.

For example when splitting into **age + less**, the analyses

age/STM + less/SUF,
age/PRE + less/STM + ly/SUF,
and **age/PRE + less/STM + ness/SUF**

would replace the previous analyses for these words. Observe that **less** received two different categories. In this example the second tagging was in error[1], but variations in tagging can indicate different morphemes with the same surface form.

The join operator needs to compare only two alternatives: keeping the morph bigram separate or joining it into a single morph. In the example, joining (**critical**, **ly**, INITIALANDFINAL) would result in

critically/STM,
and **un/PRE + critically/STM**.

Shifting uses the same units as the join operator, but considers a larger number of alternatives. Using the control parameters limiting both the length

[1] **age/STM + less/SUF + ly/SUF** and **age/STM + less/SUF + ness/SUF** are correct

of the remaining morphs and the shift distance to two letters, the example unit (**age**, **less**, InitialAndFinal) generates the alternatives

ag + eless,
agel + ess,
and **agele + ss**.

The alternative **a + geless** is not generated, because the shorter remaining morph is shorter than the two letter minimum. The alternative **ageles + s** would additionally violate the second limit by shifting the boundary by more than two letters, and is also not generated.

4.3.4 Characteristics of the search

The search is not guaranteed to converge to a local optimum, as the cost calculations within an epoch are approximate. This is because the perplexities of morphs are only periodically recalculated between epochs, and the perplexities of new morphs are estimated from the parent morph(s). In practice, this effect is small and the search is very likely to converge.

The operators are not fully symmetrical, because the same change is performed to all occurrences of the target, but the unit sizes of the targets vary between morph, morph bigram and word. In addition to this, the join and shift operators are context sensitive while splitting is not.

4.4 Challenges and alternative approaches

Even though this work was built on solid foundations set by previous Morfessor methods, the process of designing the method was not without challenges and trade-offs. This section describes some decisions made during the formulation of the model, and their consequences.

4.4.1 Flat versus hierarchical lexicon

Omitting the hierarchy from the lexicon was necessary in order to use the established weighting scheme by Kohonen et al. (2010). The decision is not without cost, however, as a hierarchical lexicon can alleviate undersegmentation. Even in applications where undersegmenting frequent words is desirable, it may be beneficial to detect structure in such words to inform the segmentation of less frequent words. Recombining the frequent words would then be delegated to a post-processing step. It might therefore be beneficial

if an alternate weighting scheme compatible with the use of hierarchy was developed.

The reintroduction of hierarchy would also remove the need for heuristic post-processing to remove non-morphemes. In Morfessor CatMAP, the hierarchy provides a straightforward way to ensure that no non-morphemes remain in the final segmentation: the hierarchy is only expanded to a level that does not include any morphs categorized as non-morphemes. The use of a flat lexicon makes this strategy unavailable, so a heuristic method for non-morpheme removal is needed.

The initial heuristic to be examined proved to be detrimental to segmentation results, as it was too aggressive in joining morphs. The initial heuristic was defined as: All successive non-morphemes are joined together. If the resulting morph is longer than 4 characters, it is accepted as a stem. If any short non-morphemes remain, they are joined either to the preceding or following morphs (the latter only for those in the initial position).

The current heuristic adds a step before the joining: non-morphemes preceded by a suffix and followed by only suffixes or other non-morphemes are recategorized as suffixes without joining with their neighbors.

The new heuristic especially improves over the performance of the old heuristic for Turkish, in which long sequences of short suffixes are frequent.

4.4.2 Morph categorization scheme

Short linking elements, such as hyphens between compound words, do not fit into the current set of morph categories. They are likely to be categorized as non-morphemes due to their shortness, which makes them eligible to be joined by the heuristic post-processing for non-morpheme removal. This can be alleviated by explicitly listing the characters for forced splitting, as a form of light supervision.

An unsupervised approach to the problem is presented by Bernhard (2008), who includes a separate category for linking elements, intended to also capture linking single-letter morphemes such as the **o** in **psych*o*logy** or **i** in **insect*i*cide**.

The FlatCat software is designed to be flexible with regard to the categorization, so the challenge lies mainly in formulating the conditional category probability appropriately.

Currently the morph property based conditional class probabilities are used throughout the training. Switching to ML-estimation for the emission probabilities would remove the need to recalculate the right and left perplexities of morphs during training. A second benefit would be increased sparsity in the categorization, resulting in less morphs with multiple categories in the

segmentation. A downside would be the increased complexity of using two different methods for calculating emission probabilities.

4.4.3 Implementation choices

An implementation choice was to internally represent morphs as strings instead of following the example of Morfessor Baseline 2.0 by numerically indexing the letters in the corpus. The choice was motivated by simplicity and extensibility of the code, as a tradeoff with increased memory consumption.

Another, to the author initially unintuitive, implementation choice is shared with Morfessor Baseline 2.0. In Baseline 2.0 the search is performed by changing the model back and forth, instead of basing the decision on calculated changes in model cost, and only modifying the model when a decision has been made. While the latter approach may seem easier to make robust, it would involve heavier, more complex calculations or a less memory efficient algorithm.

Morfessor FlatCat introduces several options that are not present in Morfessor Baseline, including the perplexity threshold, the limits on the shift operator and the training operator sequence. While the method is less sensitive to these options than to the main options (dampening and likelihood weighting), they do still require values to be optimized. Combined with the relatively high computational cost of training FlatCat models, this does make use of FlatCat somewhat time consuming.

4.4.4 Learning of corpus likelihood weights

The typical way to set the likelihood weights is through grid search. For large corpora this can become cumbersome due to the large amount of computation required.

To alleviate the computational requirements, several attempts were made to devise a scheme for automatically learning the likelihood weights. None of the approaches achieved a satisfactory compromise between robustness of the learned weights and the amount of computation.

The first approach was similar to the one in Morfessor Baseline 2.0, adjusting the corpus weight parameter α between each epoch, if evaluation against a development set shows a large enough imbalance between Precision and Recall. A single step is taken in the direction assumed to reduce the imbalance: the corpus weight is increased if Recall is higher than Precision indicating over-segmentation, and is reduced if Precision dominates indicating under-segmentation. Whether or not the change was sufficient

is not evaluated until the next epoch of training has been completed. The annotation weight parameter β is not adjusted.

This approach is less suited for FlatCat, due to the asymmetry of the operators. If training is started far from the optimal corpus weight, the damage done by the first epoch of training might not be reversed in later epochs.

In an attempt to alleviate the problem a new training procedure was developed. The procedure relies on performing and evaluating tests consisting of (partial) training of the model with different weight values. Once a value for the corpus weight has been evaluated, all changes made during the weight learning test are reverted.

The selection of weight values to be tried uses a modified Powell's conjugate direction algorithm for finding local extrema. Powell's algorithm finds a local optimum of a multivariate function, without need for derivatives. The modifications to the algorithm were motivated by the unusually heavy objective function.

A downside of the approach is that it becomes more challenging to exploit parallelism in the weight optimization. In grid search the individual runs with different weights are independent and can trivially be executed in parallel. When using an optimization method to select weight values, later iterations of the optimization must wait for the earlier results[2].

As performing full training for each tried value of the corpus weight in series is prohibitively expensive, it becomes attractive to reduce the amount of computation for each tried weight value. One attempted solution is limiting the depth of the test training to a single epoch. While one epoch is not enough to reach convergence, it may be enough to evaluate the appropriateness of a weight value.

Another more radical approach is to limit the part of the corpus from which the training units are selected, instead of performing a full training epoch. This is done by sampling one or more subsets of the corpus, to which the unit selection is limited. The changes still affect the whole model, and the costs are calculated from the whole model, but the extent of the local search is limited.

The subsampling had a detrimental effect on the robustness of the results, which was not adequately addressed by using a majority vote of several sampled subsets.

[2]While some parallelism could be achieved in the line search, to exploit it in context of the larger optimization algorithm would require communication between the parallel processes. This potential for parallelism was not exploited in this work.

Neither of the attempts to speed up the weight learning tests could achieve a significant enough speedup to make it an attractive alternative to grid search, without detrimental effects on the quality of the learned weights.

There are alternatives to the current direction cue used in the weight learning.

A limited direction cue for β is given by the number of violated annotations after a resegmentation of the corpus. The number of violated annotations does, however, not go to zero even for clearly too high values of β, making use of the cue difficult. This cue is not currently used.

In addition to the two weight parameters α and β, there is a third important parameter in the FlatCat model: the threshold value for the sigmoid function used as a part in transforming perplexities into category probabilities. Optimizing also this parameter is possible, but problematic due to the indirect effect of the parameter, and the lack of a direction cue.

A potential direction cue for the perplexity threshold could be based on the distribution of categories used in the analysis of the corpus. Further research is required to define the cue in a language independent way.

Chapter 5

Experiments

In this chapter, we present the performed experiments, consisting of both direct and indirect evaluations. The results of the experiments are presented and analyzed.

5.1 Evaluation and data sets

The main evaluation used in this thesis is a direct evaluation comparing the produced segmentations against linguistic gold standard segmentations in multiple languages. The evaluation score is the *Boundary Precision and Recall* (BPR). The evaluation method is described in Section 2.3.

We used the English, Finnish and Turkish data sets from Morpho Challenge 2010 (Kurimo et al., 2010a). The purposes and the number of word forms (types) in the data sets are shown in Table 5.1.

In addition we performed indirect evaluation through Information Retrieval experiments, using the English and Finnish data sets from Morpho

Table 5.1: Number of word forms in data sets used for BPR evaluation.

		Word forms		
Data set	Annotations	English	Finnish	Turkish
Training	Unannotated	878 036	2 928 030	617 298
	Annotated	1000	1000	1000
Development	Annotated	694	835	763
Test	Annotated	10×1000	10×1000	10×1000

Challenge 2010. The English data set consists of 170 000 short documents, 50 test queries and 20 000 relevance assessments. The Finnish data set has 55 000 documents, 50 queries and 23 000 relevance assessments. The corpora in both languages consist of newspaper text.

The testing methodology for the IR experiments followed Morpho Challenge 2010. The experiments were performed using Lemur Toolkit (Ogilvie and Callan, 2001) with Okapi BM25 ranking. All word forms in both the corpora and the queries were replaced by the morpheme segmentations produced by the evaluated methods. A stoplist of terms occurring more than 75 000 (Finnish) or 150 000 (English) were constructed, separately for each method.

Morfessor Baseline and FlatCat were trained with three levels of supervision. In *unsupervised training*, only the unannotated training data was used during training, and the development set was only used to select the value of the perplexity threshold. In *likelihood weighted training*, the development set was also used to choose α, and training was performed only with the unannotated training data. In *semi-supervised training* the development set was used to set all hyperparameters, and training was performed with both the unannotated and annotated training data. For a more thorough description of the levels of supervision, see Sections 2.1.2 and 3.2.6.

5.2 Grid optimization of hyperparameters

All weights and the perplexity threshold parameters were optimized separately for each method, using a grid search with a held-out data set.

As Morfessor FlatCat uses a Morfessor Baseline segmentation as input, the hyperparameters of Baseline were optimized first. The search for FlatCat hyperparameters could then be directed by the assumption that the optimal values for FlatCat would be near (but not necessarily identical to) the corresponding values for Baseline.

During the optimization of hyperparameters, only the best matching segmentation from the gold standard was used. This gives fair results under the assumption that only one analysis per word is returned, which is true for the evaluated methods. In the final tests, equal weight is given to each alternative gold standard analysis. This is done to make results comparable with previously published results.

In the following sections, some details of hyperparameter optimization for FlatCat will be discussed.

5.2.1 Perplexity threshold

The perplexity threshold is the threshold value for the sigmoid function used as a part in transforming perplexities into category probabilities. As it is needed on all levels of supervision, it was the first hyperparameter to be optimized.

However, optimizing a hyperparameter with a known bad likelihood weight α is undesirable. As (unweighted) unsupervised learning was assumed to be far from optimal for some languages, the perplexity threshold was initially optimized using the optimal α for Baseline, under the assumption that the optimum for FlatCat would be similar.

This initial optimization resulted in the values 10 for English, 75 for Finnish and 75 for Turkish. These values were used in all the subsequent experiments. Later the perplexity threshold was reoptimized with the optimal α. This resulted in different values for the perplexity threshold for English and Turkish, but only a small improvement in F-measure. The new optima were 15 for English, 75 for Finnish and 30 for Turkish.

The perplexity threshold of Morfessor CatMAP was optimized separately. The found optima were 75 for English and Turkish, and 300 for Finnish.

5.2.2 Corpus likelihood weight

The weight hyperparameter α for the unlabeled corpus has a straight-forward effect on the amount of segmentation performed by the model. Figure 5.1 illustrates this, by plotting Precision against Recall for different values of α. The values of α were evaluated using the development set, and the plot shows these development set results. Heuristic removal of non-morphemes was disabled, leaving non-morphemes in the segmentation unchanged.

When the corpus is minimally segmented, Precision reaches its maximal value but Recall is zero. When boundaries are placed between all letters in the corpus, Recall is maximized but Precision is poor. A suitable compromise between Precision and Recall can be selected by setting α. The values resulting in the highest F-measure do not equal the values that cause Precision to equal Recall.

5.2.3 Weights for semi-supervised training

Using the semi-supervised training requires setting values for both α and β. Values for these hyperparameters were found using grid search. β also affects the amount of segmentation performed by the model. Figure 5.2 shows the

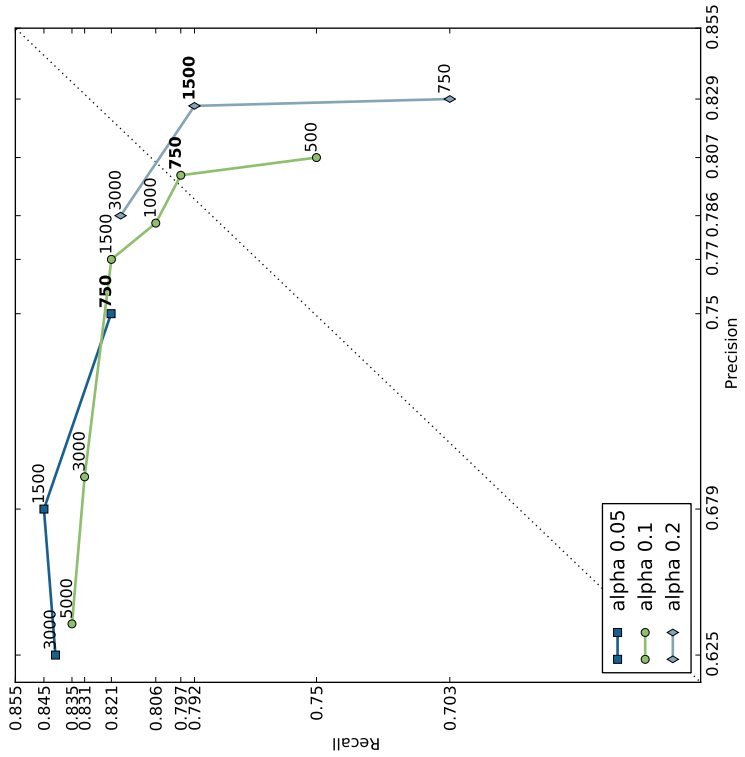


Figure 5.2: Precision versus Recall on Finnish development set, for semi-supervised training with different combinations of α and β. Lines connect runs with the same α. Labels show the values for β. The label in boldface shows the β resulting in the highest F-measure for each α.

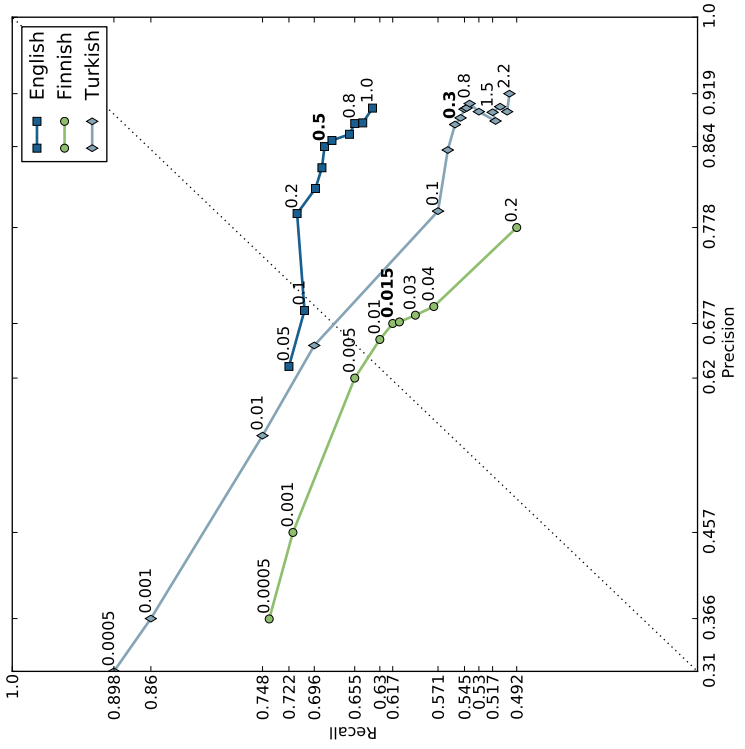


Figure 5.1: Precision versus Recall on development set, for different values of α. The supervision was limited to corpus likelihood weighting, no annotated corpus was used during training. Labels show the values for α. The label in boldface shows the α resulting in the highest F-measure for each language.

effect of varying β for three different values of α, using the Finnish data set. The dependency between α and β complicates the optimization.

The optimization was performed by evaluating against the development set, and the annotated training set was used in the training. Heuristic removal of non-morphemes was disabled, leaving non-morphemes in the segmentation unchanged.

5.2.4 Limits for the shift operator

Based on preliminary experiments, the limits for the shift operator were set to forbid moving the boundary by more than two letters, and to require the resulting morphemes to be at least two letters long.

The results of repeating the experiment with the optimized weights for semi-supervised learning are shown in Table 5.2. For Turkish, the $(2, 2)$ combination is optimal. For English and Finnish, increasing the maximum distance for moving the boundary to 3 is better, but setting both parameters to 2 is a more conservative language-independent choice. Disabling the shift operator reduces the F-measure for all languages, indicating that the new operator increases the flexibility of the model in a beneficial way.

Table 5.2: F-measures when comparing against development set for different values for the limits on the shift operator. *Maximum distance* limits the number of letters a boundary can be moved. *Minimum remainder* defines the shortest morph that can be created. Best results have been hilighted using boldface.

	Shift	Maximum	Minimum remainder		
Language	disabled	distance	1	2	3
English	0.866	1	0.861	0.866	0.864
		2	0.863	0.867	0.865
		3	0.863	**0.868**	0.866
Finnish	0.801	1	0.807	0.809	0.807
		2	0.810	0.810	0.809
		3	0.805	**0.813**	0.809
Turkish	0.832	1	0.826	0.829	0.828
		2	0.824	**0.836**	0.830
		3	0.830	0.832	**0.836**

5.3 Results

The remainder of this chapter presents the results of the evaluation.

5.3.1 Comparison against gold standard boundaries

The results of the BPR evaluation against a gold standard segmentation are shown in Table 5.3.

Semi-supervised FlatCat achieves the highest F-measure for both the English and Finnish data sets. Semi-supervised FlatCat surpasses the unsupervised CatMAP for all languages. FlatCat fails to reach the state of the art for the Turkish data set. For Turkish likelihood weighted models, FlatCat improves the F-measure compared to the weighted Baseline input. Weighted FlatCat does, however, not surpass the F-measure of unsupervised CatMAP.

The difference between the highest and second highest scoring methods are statistically significant for all three languages. The significance testing was performed using the Wilcoxon signed-rank test for paired samples. One-sided tests with $p < 0.01$ were used.

FlatCat suffers from undersegmentation in Turkish, indicated by high Precision and low Recall. A majority of the errors are caused by undersegmenting long sequences of short morphemes. Especially two letter morphemes, which are common in Turkish, seem to cause problems. This type of error is introduced both during training by a poorly fitting categorization scheme, and at the analysis phase by poorly fitting heuristics for non-morpheme removal.

5.3.2 Varying the size of the annotated data set

The results of varying the size of the annotated data are plotted in Figure 5.3. In addition to the semi-supervised FlatCat, the results for the semi-supervised Baseline with the same data sets are plotted for comparison. The unsupervised (U) FlatCat and CatMAP do not use annotated data, but are plotted as horizontal lines for comparison. The likelihood weighted FlatCat and Baseline are plotted as extending the semi-supervised results to zero annotated words.

The smaller data sets were random subsets of the list of 1000 annotated word types. Each subset was independently drawn. The unannotated corpus likelihood weight parameter α was set to the values optimized for the full 1000 word annotated data set, instead of being reoptimized for each data set size, due to time constraints. As the amount of unannotated data affected by α does not change, the optimal value is assumed to remain constant. The

Table 5.3: Boundary Precision and Recall results in comparison to gold standard segmentation. Abbreviations have been used for unsupervised (U), likelihood weighted (W) and semi-supervised (S) methods. Best results for each measure have been hilighted using boldface. The F-measure in parenthesis is for the development set. The F-measure after the slash is for the test set.

(a) English.

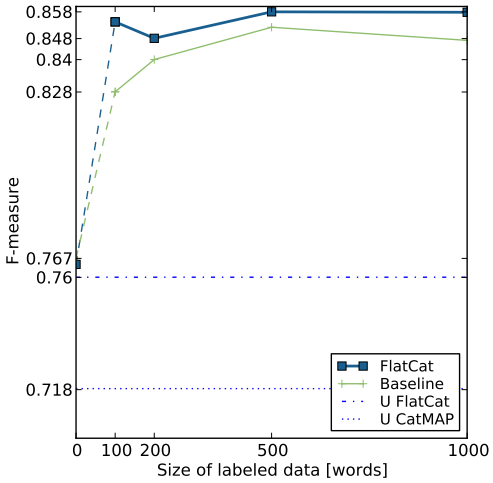
Method	α	β	Precision	Recall	F_1
U Baseline	1.0	–	.88	.59	(.72)/.71
U CatMAP	–	–	.89	.51	(.67)/.65
U FlatCat	1.0	–	**.90**	.57	(.72)/.69
W Baseline	0.7	–	.83	.62	(.73)/.71
W FlatCat	0.5	–	.84	.60	(.72)/.70
S Baseline	1.0	3000	.83	**.77**	(.82)/.80
S FlatCat	0.9	2000	.86	**.77**	(.83)/**.81**

(b) Finnish.

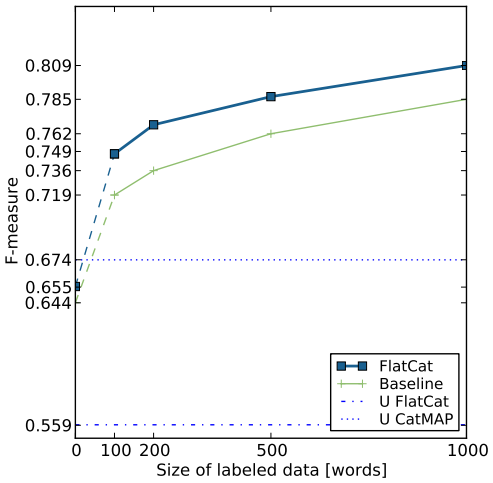
Method	α	β	Precision	Recall	F_1
U Baseline	1.0	–	**.84**	.38	(.53)/.53
U CatMAP	–	–	.76	.51	(.63)/.61
U FlatCat	1.0	–	**.84**	.38	(.52)/.52
W Baseline	0.02	–	.62	.54	(.61)/.58
W FlatCat	0.015	–	.66	.52	(.62)/.58
S Baseline	0.1	15000	.75	**.72**	(.75)/.73
S FlatCat	0.2	1500	.79	.71	(.77)/**.75**

(c) Turkish.

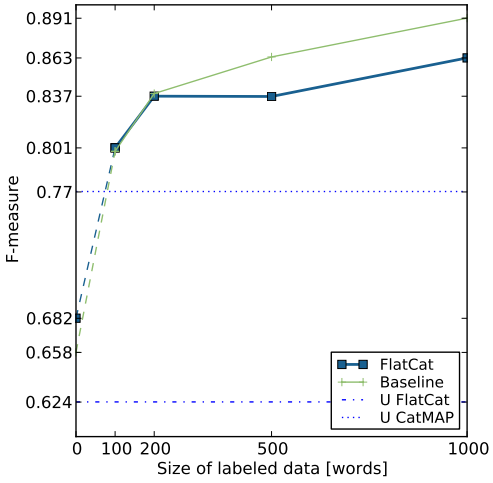
Method	α	β	Precision	Recall	F_1
U Baseline	1.0	–	.85	.36	(.48)/.51
U CatMAP	–	–	.83	.50	(.60)/.62
U FlatCat	1.0	–	.87	.36	(.47)/.51
W Baseline	0.1	–	.71	.41	(.51)/.52
W FlatCat	0.3	–	**.88**	.38	(.52)/.53
S Baseline	0.4	2000	.86	**.60**	(.70)/**.71**
S FlatCat	0.4	800	**.88**	.51	(.65)/.65



(a) English.



(b) Finnish.



(c) Turkish.

Figure 5.3: Effect of varying the size of the annotated data. Results are for the development set, which was also used to optimize the weight parameters α and β. The line between likelihood weighted training (0 words) and the first semi-supervised result (100 words) is plotted with a dashed line.

optimal value for the annotated corpus likelihood weight β depends on the size of the annotated data set. The parameter was reoptimized for each set, using a separate grid search for each set size, with varying β but constant α.

The largest increase in F-measure is seen for the first 100 words, with diminishing returns for larger sets. Already 100 annotated words was found to yield a significant improvement over unsupervised and likelihood weighted training. FlatCat trained with 100 annotated words surpasses the results of CatMAP for all three tested languages.

However, FlatCat was found to be highly sensitive to the β parameter. With suboptimal values for β, the results deteriorated faster than for Baseline. The range of variation in the optimized values for β was also larger for FlatCat as shown in Table 5.4.

The sensitivity to β may cause some difficulty when using FlatCat with varying amounts of annotation. This can occur if more annotations become available, or when training a final production model with all the available annotations after using a split data set to choose the hyperparameters.

There is some variance in the results due to the use of a single random sample, as evidenced by the dips in F-measure at 200 words for FlatCat and 1000 words for Baseline.

Not all annotations are equally beneficial for Baseline and FlatCat, however, as both methods have been trained with the same sampled data sets, but only one of the methods show a dip in F-measure.

Table 5.4: Ranges of variation for optimal β, with varying size of annotated data set.

		Size of annotated data set	
Language	Method	100 words	1000 words
English	FlatCat	20 000	2000
	Baseline	5000	3000
Finnish	FlatCat	20 000	1500
	Baseline	17 000	15 000
Turkish	FlatCat	10 000	800
	Baseline	15 000	2000

Table 5.5: IR results for English. Results of the method developed in this thesis are hilighted using boldface. The results marked with [1] are from (Virpioja, 2012). Mean Average Precision is abbreviated as MAP.

Rank		Method	Short affix removal	MAP
1	–	Snowball Porter[1]	–	0.4092
2	U	ParaMor-Morfessor 2008[1]	–	0.4092
3	–	TWOL first[1]	–	0.4020
4	S	Baseline	–	0.3855
5	**S**	**FlatCat**	**Yes**	**0.3826**
6	**S**	**FlatCat**	**No**	**0.3788**
7	W	Baseline	–	0.3761
8	U	Baseline	–	0.3695
9	U	CatMAP	No	0.3682
10	U	CatMAP	Yes	0.3653
11	**W**	**FlatCat**	**No**	**0.3651**
12	**W**	**FlatCat**	**Yes**	**0.3606**
13	**U**	**FlatCat**	**No**	**0.3486**
14	**U**	**FlatCat**	**Yes**	**0.3451**
15	–	(Words)	–	0.3303

5.3.3 Information Retrieval

IR experiment results are shown in Tables 5.5 (English) and 5.6 (Finnish). Likelihood weighted FlatCat achieves the highest IR result for Finnish, surpassing the handcrafted topline reference method TWOL first and the top scoring method participating in Morpho Challenge 2010, Lignos Aggressive Comp 2010. The TWOL method is the lemma from the first analysis alternative given by the TWOL morphological analyzer from Lingsoft.

The semi-supervised FlatCat also improves on the results of the corresponding Baseline model in Finnish. For English, FlatCat fails to surpass the corresponding Baseline model for all three levels of supervision.

The poor performance of FlatCat compared to Baseline in the English IR experiment may be caused by undersegmentation of frequent morphs. Baseline is more aggressive in using morphs that are included in the morph lexicon, while FlatCat may under some conditions choose to save one transition and emission by emitting a supermorph instead, even if the submorphs are in the lexicon. The use of a hierarchical lexicon could alleviate this problem.

Table 5.6: IR results for Finnish. Results of the method developed in this thesis are hilighted using boldface. The results marked with [1] are from (Virpioja, 2012). Mean Average Precision is abbreviated as MAP.

Rank		Method	Short affix removal	MAP
1	**W**	**FlatCat**	**No**	**0.5057**
2	**W**	**FlatCat**	**Yes**	**0.5029**
3	**S**	**FlatCat**	**Yes**	**0.4973**
	–	TWOL first[1]	–	0.4973
5	U	Lignos Aggressive Comp. 2010[1]	–	0.4914
6	U	CatMAP	Yes	0.4884
7	**S**	**FlatCat**	**No**	**0.4868**
8	U	CatMAP	No	0.4865
9	S	Baseline	–	0.4722
10	W	Baseline	–	0.4582
11	U	Baseline	–	0.4378
12	**U**	**FlatCat**	**Yes**	**0.4349**
13	**U**	**FlatCat**	**No**	**0.4334**
14	–	Snowball Finnish[1]	–	0.4275
15	–	(Words)	–	0.3483

5.3.4 Affix removal in Information Retrieval

Stemming has been shown to improve Information Retrieval results (Kurimo et al., 2010b). An explanation for this improvement is that inflection is often not relevant to the query. A successful morphological segmentation lessens the effect of inflection by making the stems available as search terms, but the presence of the affixes still acts as noise. The most frequent of these are removed by the stoplisting.

The morph categories make it possible to simulate stemming by removing morphs categorized as prefixes or suffixes. This removes the noise caused by rare or uninformative affix morphs.

However, some meaningful morphs that are common as latter parts of compound words are categorized as suffixes. In Finnish, prefixes also tend to be more semantic while suffixes tend more towards syntactic roles. These meaningful morphs tend to be longer than the suffixes used for inflection. To avoid removing these meaningful morphs, we imposed a limit of at most

3 letters for the removed morphs. For methods with morph categories, we report the results both with and without this removal of short affixes.

With the help of short affix removal, Finnish semi-supervised FlatCat reaches the level of the TWOL topline method, but not the level of likelihood weighted FlatCat.

For English, semi-supervised FlatCat with short affix removal is the best performing configuration for FlatCat, but it fails to improve on the semi-supervised Baseline.

The IR performance of FlatCat benefits from removal of short affixes for semi-supervised learning for both languages, and unsupervised learning for Finnish. In other cases the affix removal is detrimental.

The same procedure was also applied to CatMAP, and was found to be beneficial for Finnish but detrimental for English.

5.3.5 Analysis of errors

The next experiment was made in order to examine differences between FlatCat and the reference methods, in the types of segmentation errors. The experiment uses lists of manually verified words that have an analysis with a desired morph pattern as the only valid analysis.

Figure 5.4 on page 63 shows the effects of supervision on words consisting of different morph category patterns. Each subfigure plots results on a list of 500 words from the Finnish gold standard, with one of four morph patterns. Precision versus Recall is plotted for three levels of supervision.

The four lists consist of words with the following morph patterns: compound words consisting of exactly two stems (STM + STM), a prefix followed by a stem (PRE + STM), a stem followed by a single suffix (STM + SUF) and a stem and exactly two suffixes (STM + SUF + SUF).

In Finnish, all morph patterns except PRE + STM benefit from supervision. Supervision improves both of the morph patterns containing suffixes (STM + SUF and STM + SUF + SUF) significantly, with improvement especially for the Recall.

The clearest difference to the Baseline results is seen in the semi-supervised STM + STM results. For FlatCat the supervision improves the Precision for these compound words, while for Baseline the weighted model has both better Precision and better Recall than the semi-supervised model for these words.

Some of these differences are words where Baseline has been unable to find the boundary between the compound parts, while FlatCat has succeeded in finding it. Examples are **valosaaste** and **pölypilvi**, which Baseline segments as **va + losaaste** and **pölypilv + i** instead of the correct **valo + saaste** and **pöly + pilvi**.

A larger part of the differences, however, consist of words where affixes have been used in incorrect positions, such as **näkötesti** and **sieniseos** which Baseline segments as **nä + kö + te + sti** and **si + en + i + seos** instead of the correct **näkö + testi** and **sieni + seos**. FlatCat segments these examples correctly.

Semi-supervised FlatCat improves over the results of CatMAP for both patterns containing suffixes. The STM + SUF pattern improves strongly in Recall without much decrease in Precision. The STM + SUF + SUF pattern shows an even clearer improvement in both measures. A small decrease is seen in the Recall of two stem compound words. The largest decrease compared to CatMAP is seen for the PRE + STM pattern, for which both Precision and Recall are lower for FlatCat. It is worth noting that FlatCat performs clearly better than Baseline for this pattern.

The English results are not included, as they show a very similar pattern to the Finnish results. The main difference is that also the segmentation of STM + STM words suffers from supervision.

The analysis presented in this section indicates that, for the examined languages, supervision improves the modeling of suffixation, while morphotactics improve the Precision when modeling compounding.

These results may be explained by the differences in the usage of morphemes belonging to these categories. Suffixes in English and Finnish are much closer to a closed class of morphemes than stems are, which means that a large enough coverage of suffixes can be achieved with an annotated data set. Oversegmentation of rare compound words is alleviated when spuriously occurring substrings that happen to match suffixes are not spit from the beginning or middle of the word.

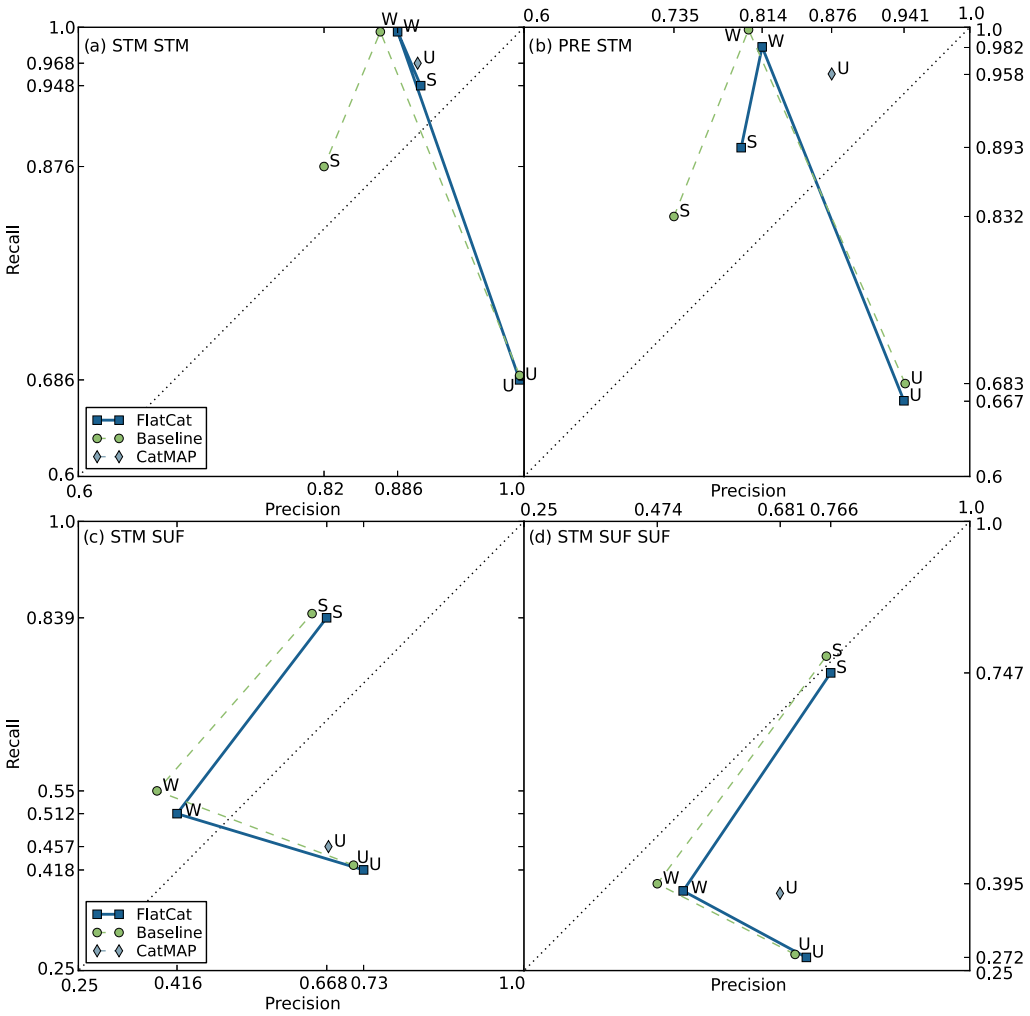


Figure 5.4: Precision versus Recall on manually constructed Finnish lists of words with a specific analysis pattern. The panes shows the results for: (a) compound words consisting of exactly two stems, (b) words consisting of a prefix followed by a stem, (c) words consisting of a stem followed by a single suffix, (d) words consisting of a stem and exactly two suffixes. Note the differing scale between the top and bottom panes. The three levels of supervision are marked (U) unsupervised, (W) weighted and (S) semi-supervised. Baseline and CatMAP results are plotted for comparison.

Chapter 6

Conclusions and future directions

In this thesis, we set out to develop and evaluate a new method for morphological segmentation. The method, called Morfessor FlatCat, combines morphotactics based on a hidden Markov model with semi-supervised training. FlatCat builds on previous work in the Morfessor family of methods, reusing some components from Morfessor Baseline and Morfessor CatMAP.

The model contains weighting parameters, α and β, that are useful for optimizing the segmentation for a particular problem. After setting values for these weighting parameters and other hyperparameters using an annotated development set, the method was evaluated by comparing the output of the method against a gold standard segmentation.

Semi-supervised FlatCat achieves the highest F-measure for both the English and Finnish data sets. FlatCat fails, however, to reach the state of the art for the Turkish data set.

The applicability of the method to a natural language processing problem was demonstrated through Information Retrieval (IR) experiments. Likelihood weighted FlatCat achieves the highest IR result for Finnish, surpassing the handcrafted topline reference method TWOL first. Removal of short affixes from the segmentation was shown to improve the IR performance of FlatCat under some conditions.

The results presented in this thesis show that it is possible to simultaneously leverage morphotactics and supervision for improved results in morphological segmentation.

Semi-supervised training allows efficient use of small amounts of annotated data, together with much larger unannotated data. FlatCat trained with 100 annotated words surpasses the results of the unsupervised CatMAP method for all three tested languages. Producing an annotated data set of a few hundred words should be within reach even for low resource languages.

This thesis has in no way exhausted the potential experiments with this model. During the work some ideas for extensions and improvements of the method were brought up, but did not fit within the scope of the thesis and were not pursued.

Undersegmentation and the issue of non-morphemes could be addressed by devising an alternate weighting scheme compatible with the use of hierarchy. Alternatively better heuristics for non-morpheme removal could be developed.

Studying the effects of alternative morph categorization schemes would be likely to yield language-specific benefits, but could also result in a better language-independent scheme.

A procedure for learning of corpus likelihood weights would be beneficial as an alternative to grid search.

Applying active learning for noise robustness and for solicitation of annotations could make FlatCat even more attractive for low resource languages.

Even further possibilities would be opened by loosening the central assumptions. The first assumption is that words in a sentence are independent. Using sentence context would, however, allow making Part-Of-Speech -like (POS) distinctions that could disambiguate inflections of different lexemes that have the same surface form but should be analyzed differently. POS-like categories could also be used for finding paradigm-like patterns to alleviate data sparsity and reduce the number of incoherent analyses.

The second assumption is that a morphology consists only of concatenative processes. Introducing transformations to model allomorphy would allow finding the shared abstract morphemes underlying different allomorphs. This could be especially beneficial in Information Retrieval and machine translation applications.

Modifying the final assumption of the structure of the morphotactics by replacing the HMM, e.g. with some other graphical probabilistic model, would be a very fundamental change resulting in something that can only be considered an entirely new model.

Bibliography

Aarts, E. and Lenstra, J. K. (1997). *Local search in combinatorial optimization*. Princeton University Press, Princeton, New Jersey.

Bernhard, D. (2008). Simple morpheme labelling in unsupervised morpheme analysis. In *Advances in Multilingual and Multimodal Information Retrieval*, pages 873–880. Springer.

Bernhard, D. (2010). Morphonet: Exploring the use of community structure for unsupervised morpheme analysis. In *Multilingual Information Access Evaluation I. Text Retrieval Experiments*, pages 598–608. Springer.

Can, B. and Manandhar, S. (2009). Unsupervised learning of morphology by using syntactic categories. In *Working Notes, CLEF 2009 Workshop*.

Chan, E. and Lignos, C. (2010). Investigating the relationship between linguistic representation and computation through an unsupervised model of human morphology learning. *Research on Language and Computation*, 8(2-3):209–238.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

Çöltekin, Ç. (2010). A freely available morphological analyzer for Turkish. In *LREC 2012*.

Creutz, M. (2003). Unsupervised segmentation of words using prior distributions of morph length and frequency. In Hinrichs, E. W. and Roth, D., editors, *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 280–287, Sapporo, Japan. Association for Computational Linguistics.

Creutz, M. (2006). *Induction of the morphology of natural language: Unsupervised morpheme segmentation with application to automatic speech recognition*. PhD thesis, Helsinki University of Technology.

Creutz, M., Hirsimäki, T., Kurimo, M., Puurula, A., Pylkkönen, J., Siivola, V., Varjokallio, M., Arisoy, E., Saraçlar, M., and Stolcke, A. (2007). Analysis of morph-based speech recognition and the modeling of out-of-vocabulary words across languages. In *HLT-NAACL*, pages 380–387.

Creutz, M. and Lagus, K. (2002). Unsupervised discovery of morphemes. In Maxwell, M., editor, *Proceedings of the ACL-02 workshop on Morphological and phonological learning*, volume 6, pages 21–30, Philadelphia, PA, USA. Association for Computational Linguistics.

Creutz, M. and Lagus, K. (2004). Induction of a simple morphology for highly-inflecting languages. In *Proceedings of the 7th Meeting of the ACL Special Interest Group in Computational Phonology: Current Themes in Computational Phonology and Morphology*, pages 43–51, Barcelona, Spain. Association for Computational Linguistics.

Creutz, M. and Lagus, K. (2005a). Inducing the morphological lexicon of a natural language from unannotated text. In Honkela, T., Könönen, V., Pöllä, M., and Simula, O., editors, *Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR05)*, pages 106–113, Espoo, Finland. Helsinki University of Technology, Laboratory of Computer and Information Science.

Creutz, M. and Lagus, K. (2005b). Unsupervised morpheme segmentation and morphology induction from text corpora using Morfessor 1.0. Technical Report A81, Publications in Computer and Information Science, Helsinki University of Technology.

Dasgupta, S. and Ng, V. (2007). High-performance, language-independent morphological segmentation. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics*, pages 155–163, Rochester, NY.

Deléger, L., Namer, F., and Zweigenbaum, P. (2009). Morphosemantic parsing of medical compound words: Transferring a French analyzer to English. *International Journal of Medical Informatics*, 78:S48–S55.

Demberg, V. (2007). A language-independent unsupervised model for morphological segmentation. In *Annual meeting of the Association for Computational Linguistics*, volume 45, page 920.

Demberg, V., Schmid, H., and Möhler, G. (2007). Phonological constraints and morphological preprocessing for grapheme-to-phoneme conversion. In *Annual meeting of the Association for Computational Linguistics*, volume 45, page 96.

El-Kahlout, I. D. and Oflazer, K. (2010). Exploiting morphology and local word reordering in English-to-Turkish phrase-based statistical machine translation. *Audio, Speech, and Language Processing, IEEE Transactions on*, 18(6):1313–1322.

Garay-Vitoria, N. and Abascal, J. (2006). Text prediction systems: a survey. *Universal Access in the Information Society*, 4(3):188–203.

Goldsmith, J. (2001). Unsupervised learning of the morphology of a natural language. *Computational linguistics*, 27(2):153–198.

Golénia, B., Spiegler, S., and Flach, P. (2009). Ungrade: unsupervised graph decomposition. In *Working Notes, CLEF 2009 Workshop*.

Gries, S. Th.. (2006). Some proposals towards more rigorous corpus linguistics. *Zeitschrift für Anglistik und Amerikanistik*, 54(2):191–202.

Grünwald, P. D. (2007). *The minimum description length principle*. MIT Press, Cambridge, MA.

Habash, N., Eskander, R., and Hawwari, A. (2012). A morphological analyzer for Egyptian Arabic. In *Proceedings of the Twelfth Meeting of the Special Interest Group on Computational Morphology and Phonology*, pages 1–9. Association for Computational Linguistics.

Hafer, M. A. and Weiss, S. F. (1974). Word segmentation by letter successor varieties. *Information storage and retrieval*, 10(11):371–385.

Hammarström, H. and Borin, L. (2011). Unsupervised learning of morphology. *Computational Linguistics*, 37(2):309–350.

Harris, Z. S. (1955). From phoneme to morpheme. *Language*, 31(2):190–222.

Hirsimäki, T., Creutz, M., Siivola, V., Kurimo, M., Virpioja, S., and Pylkkönen, J. (2006). Unlimited vocabulary speech recognition with morph language models applied to Finnish. *Computer Speech & Language*, 20(4):515–541.

Hirsimaki, T., Pylkkonen, J., and Kurimo, M. (2009). Importance of high-order n-gram models in morph-based speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 17(4):724–732.

Honkela, T., Kaski, S., Lagus, K., and Kohonen, T. (1997). WEBSOM: self-organizing maps of document collections. In *Proceedings of WSOM*, volume 97, pages 4–6.

Kazakov, D. and Manandhar, S. (2001). Unsupervised learning of word segmentation rules with genetic algorithms and inductive logic programming. *Machine Learning*, 43(1-2):121–162.

Keshava, S. and Pitler, E. (2006). A simpler, intuitive approach to morpheme induction. In *Proceedings of 2nd Pascal Challenges Workshop*, pages 31–35.

Kohonen, O., Virpioja, S., and Klami, M. (2009). Allomorfessor: Towards unsupervised morpheme analysis. In *Evaluating Systems for Multilingual and Multimodal Information Access*, pages 975–982. Springer.

Kohonen, O., Virpioja, S., and Lagus, K. (2010). Semi-supervised learning of concatenative morphology. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, pages 78–86. Association for Computational Linguistics.

Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT Press, Cambridge, MA.

Kontkanen, P., Buntine, W., Myllymäki, P., Rissanen, J., and Tirri, H. (2003). Efficient computation of stochastic complexity. In *Proceedings of the Ninth International Conference on Artificial Intelligence and Statistics*, pages 233–238.

Kornai, A. (2013). Digital language death. *PloS one*, 8(10):e77056.

Koskenniemi, K. (1984). A general computational model for word-form recognition and production. In *Proceedings of the 10th international conference on Computational linguistics*, pages 178–181. Association for Computational Linguistics.

Kurimo, M., Creutz, M., and Lagus, K. (2006). Unsupervised segmentation of words into morphemes — Challenge 2005, an introduction and evaluation report. In Kurimo, M., Creutz, M., and Lagus, K., editors, *Proceedings of the PASCAL Challenge Workshop on Unsupervised segmentation of words into morphemes*, pages 1–11, Venice, Italy. PASCAL European Network of Excellence.

Kurimo, M., Virpioja, S., and Turunen, V. T. (2010a). Overview and results of Morpho Challenge 2010. In *Proceedings of the Morpho Challenge 2010 Workshop*, pages 7–24, Espoo, Finland. Aalto University School of Science and Technology, Department of Information and Computer Science. Technical Report TKK-ICS-R37.

Kurimo, M., Virpioja, S., Turunen, V. T., Blackwood, G. W., and Byrne, W. (2010b). Overview and results of Morpho Challenge 2009. In *Multilingual Information Access Evaluation I. Text Retrieval Experiments*, pages 578–597. Springer.

Lagus, K., Kohonen, O., and Virpioja, S. (2009). Towards unsupervised learning of constructions from text. In *Proceedings of the Workshop on Extracting and Using Constructions in NLP of 17th Nordic Conference on Computational Linguistics, NODALIDA*.

Lavallée, J.-F. and Langlais, P. (2010). Unsupervised morphological analysis by formal analogy. In *Multilingual Information Access Evaluation I. Text Retrieval Experiments*, pages 617–624. Springer.

Lee, Y. K., Haghighi, A., and Barzilay, R. (2011). Modeling syntactic context improves morphological segmentation. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, pages 1–9. Association for Computational Linguistics.

Lignos, C. (2010). Learning from unseen data. In *Proceedings of the Morpho Challenge 2010 Workshop*, pages 35–38.

Lindén, K., Axelson, E., Hardwick, S., Pirinen, T. A., and Silfverberg, M. (2011). HFST–framework for compiling and applying morphologies. In *Systems and Frameworks for Computational Morphology*, pages 67–85. Springer.

Ling, C. X. (1994). Learning the past tense of English verbs: The symbolic pattern associator vs. connectionist models. *Journal of Artificial Intelligence Research*, 1:209–229.

Monson, C. (2008). *ParaMor: from Paradigm Structure to Natural Language Morphology Induction*. PhD thesis, Carnegie Mellon University.

Monson, C., Hollingshead, K., and Roark, B. (2010). Simulating morphological analyzers with stochastic taggers for confidence estimation. In *Multilingual Information Access Evaluation I. Text Retrieval Experiments*, pages 649–657. Springer.

Mooney, R. J. and Califf, M. E. (1995). Induction of first-order decision lists: Results on learning the past tense of English verbs. *Journal of Artificial Intelligence Research*, 3:1–24.

Moseley, C. (ed.). (2010). Atlas of the world's languages in danger, 3rd edn. Online version: `http://www.unesco.org/culture/en/endangeredlanguages/atlas` [Accessed 04.12.2013]. Paris, UNESCO Publishing.

Naradowsky, J. and Goldwater, S. (2009). Improving morphology induction by learning spelling rules. In *IJCAI*, pages 1531–1536.

Naradowsky, J. and Toutanova, K. (2011). Unsupervised bilingual morpheme segmentation and alignment with context-rich hidden semi-markov models. In *ACL*, pages 895–904.

Oflazer, K., Nirenburg, S., and McShane, M. (2001). Bootstrapping morphological analyzers by combining human elicitation and machine learning. *Computational Linguistics*, 27(1):59–85.

Ogilvie, P. and Callan, J. P. (2001). Experiments using the Lemur toolkit. In *TREC*, volume 10, pages 103–108.

Pirinen, T., Lindén, K., et al. (2010). Finite-state spell-checking with weighted language and error models. In *Proceedings of LREC 2010 Workshop on creation and use of basic lexical resources for less-resourced languages*.

Pirinen, T. A. (2011). Modularisation of Finnish finite-state language description–towards wide collaboration in open source development of a morphological analyser. In *Proceedings of Nodalida*, volume 18.

Poon, H., Cherry, C., and Toutanova, K. (2009). Unsupervised morphological segmentation with log-linear models. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 209–217. Association for Computational Linguistics.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge Univ. Press.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14(5):465–471.

Rumelhart, D. E. and McClelland, J. L. (1985). *On learning the past tenses of English verbs*. Institute for Cognitive Science, University of California, San Diego.

Ruokolainen, T., Kohonen, O., Virpioja, S., and Kurimo, M. (2013). Supervised morphological segmentation in a low-resource learning setting using conditional random fields. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 29–37, Sofia, Bulgaria. Association for Computational Linguistics.

Schone, P. and Jurafsky, D. (2000). Knowledge-free induction of morphology using latent semantic analysis. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, pages 67–72. Association for Computational Linguistics.

Shalonova, K. and Golénia, B. (2010). Weakly supervised morphology learning for agglutinating languages using small training sets. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 976–983. Association for Computational Linguistics.

Snover, M. G., Jarosz, G. E., and Brent, M. R. (2002). Unsupervised learning of morphology using a novel directed search algorithm: Taking the first step. In *ACL-02 workshop on Morphological and phonological learning*, volume 6, pages 11–20. Association for Computational Linguistics.

Snyder, B. and Barzilay, R. (2008). Cross-lingual propagation for morphological analysis. In *AAAI*, pages 848–854.

Spiegler, S., Golénia, B., and Flach, P. A. (2010). Unsupervised word decomposition with the Promodes algorithm. In *Multilingual Information Access Evaluation I. Text Retrieval Experiments*, pages 625–632. Springer.

Spiegler, S. and Monson, C. (2010). EMMA: a novel Evaluation Metric for Morphological Analysis. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1029–1037. Association for Computational Linguistics.

Tchoukalov, T., Monson, C., and Roark, B. (2010). Morphological analysis by multiple sequence alignment. In *Multilingual Information Access Evaluation I. Text Retrieval Experiments*, pages 666–673. Springer.

Tepper, M. and Xia, F. (2010). Inducing morphemes using light knowledge. *ACM Transactions on Asian Language Information Processing (TALIP)*, 9(1):3.

Theron, P. and Cloete, I. (1997). Automatic acquisition of two-level morphological rules. In *Proceedings of the fifth conference on Applied natural language processing*, pages 103–110. Association for Computational Linguistics.

Van den Bosch, A. and Daelemans, W. (1999). Memory-based morphological analysis. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 285–292. Association for Computational Linguistics.

Virpioja, S. (2012). *Learning constructions of natural language: Statistical models and evaluations*. PhD thesis, Aalto University.

Virpioja, S., Kohonen, O., and Lagus, K. (2010). Unsupervised morpheme analysis with Allomorfessor. In *Multilingual Information Access Evaluation I. Text Retrieval Experiments*, pages 609–616. Springer.

Virpioja, S., Kohonen, O., and Lagus, K. (2011a). Evaluating the effect of word frequencies in a probabilistic generative model of morphology. In *Proceedings of the 18th Nordic Conference of Computational Linguistics (NODALIDA 2011)*, volume 11, pages 230–237.

Virpioja, S., Smit, P., Grönroos, S.-A., and Kurimo, M. (2013). Morfessor 2.0: Python implementation and extensions for Morfessor Baseline. Report 25/2013 in Aalto University publication series SCIENCE + TECHNOLOGY, Department of Signal Processing and Acoustics, Aalto University.

Virpioja, S., Turunen, V. T., Spiegler, S., Kohonen, O., and Kurimo, M. (2011b). Empirical comparison of evaluation methods for unsupervised learning of morphology. *Traitement Automatique des Langues*, 52(2):45–90.

Virpioja, S., Väyrynen, J. J., Creutz, M., and Sadeniemi, M. (2007). Morphology-aware statistical machine translation based on morphs induced in an unsupervised manner. *Machine Translation Summit XI*, 2007:491–498.

Yarowsky, D. and Wicentowski, R. (2000). Minimally supervised morphological analysis by multimodal alignment. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 207–216. Association for Computational Linguistics.

Zhu, X. (2005). *Semi-supervised learning with graphs*. PhD thesis, Carnegie Mellon University, Language Technologies Institute, School of Computer Science.