Department of Modern Languages
2016

Morphological Disambiguation using Probabilistic Sequence Models

Miikka Pietari Silfverberg

Academic dissertation to be publicly discussed, by due permission of the Faculty of Arts at the University of Helsinki in lecture hall P674, on the 22th of October, 2016 at 10 o'clock.

University of Helsinki
Finland

Supervisors
Krister Lindén, University of Helsinki, Finland
Anssi Yli-Jyrä, University of Helsinki, Finland

Pre-examiners
Prof. Jan Hajič, Charles University in Prague, Czech Republic
Prof. Joakim Nivre, Uppsala University, Sweden

Opponent
Prof. Joakim Nivre, Uppsala University, Sweden

Custos
Prof. Jörg Tiedemann, Univeristy of Helsinki, Finland

Contact information

Department of Modern Languages
P.O. Box 24 (Unioninkatu 40)
FI-00014 University of Helsinki
Finland

Email address: mpsilfve at iki piste fi
URL: http://iki.fi/mpsilfve

ISBN 978-951-51-2509-5 (printed)
ISBN 978-951-51-2510-1 (PDF)
Helsinki 2016
Unigrafia Oy

Morphological Disambiguation using Probabilistic Sequence Models

Miikka Pietari Silfverberg

Abstract

A morphological tagger is a computer program that provides complete morphological descriptions of sentences. Morphological taggers find applications in many NLP fields. For example, they can be used as a pre-processing step for syntactic parsers, in information retrieval and machine translation. The task of morphological tagging is closely related to POS tagging but morphological taggers provide more fine-grained morphological information than POS taggers. Therefore, they are often applied to morphologically complex languages, which extensively utilize inflection, derivation and compounding for encoding structural and semantic information. This thesis presents work on data-driven morphological tagging for Finnish and other morphologically complex languages.

There exists a very limited amount of previous work on data-driven morphological tagging for Finnish because of the lack of freely available manually prepared morphologically tagged corpora. The work presented in this thesis is made possible by the recently published Finnish dependency treebanks FinnTreeBank and Turku Dependency Treebank. Additionally, the Finnish open-source morphological analyzer OMorFi is extensively utilized in the experiments presented in the thesis.

The thesis presents methods for improving tagging accuracy, estimation speed and tagging speed in presence of large structured morphological label sets that are typical for morphologically complex languages. More specifically, it presents a novel formulation of generative morphological taggers using weighted finite-state machines and applies finite-state taggers to context sensitive spelling correction of Finnish. The thesis also explores discriminative morphological tagging. It presents structured sub-label dependencies that can be used for improving tagging accuracy. Additionally, the thesis presents a cascaded variant of the averaged perceptron tagger. In presence of large label sets, a cascaded design results in substantial reduction of estimation speed compared to a standard perceptron tagger. Moreover, the thesis explores pruning strategies for perceptron taggers. Finally, the thesis presents the FinnPos toolkit for morphological tagging. FinnPos is an open-source state-of-the-art averaged perceptron tagger implemented by the author.

Tiivistelmä

Disambiguoiva morfologinen jäsennin on ohjelma, joka tuottaa yksikäsitteisiä morfologisia kuvauksia virkkeen sanoille. Tällaisia jäsentimiä voidaan hyödyntää monilla kielenkäsittelyn osa-alueilla, esimerkiksi syntaktisen jäsentimen tai konekäännösjärjestelmän esikäsittelyvaiheena. Kieliteknologisena tehtävänä disambiguoiva morfologinen jäsennys muistuttaa perinteistä sanaluokkajäsennystä, mutta se tuottaa hienojakoisempaa morfologista informaatiota kuin perinteinen sanaluokkajäsennin. Tämän takia disambiguoivia morfologisia jäsentimiä hyödynnetäänkin pääsääntöisesti morfologisesti monimutkaisten kielten, kuten suomen kielen, kieliteknologiassa. Tällaisissa kielissä käytetään paljon sananmuodostuskeinoja kuten taivutusta, johtamista ja yhdyssananmuodostusta. Väitöskirjan esittelemä tutkimus liittyy morfologisesti rikkaiden kielten disambiguoivaan morfologiseen jäsentämiseen koneoppimismenetelmin.

Vaikka suomen disambiguoivaa morfologista jäsentämistä on tutkittu aiemmin (esim. Constraint Grammar -formalismin avulla), koneoppimismenetelmiä ei ole aiemmin juurikaan sovellettu. Tämä johtuu siitä että jäsentimen oppimiseen tarvittavia korkealuokkaisia morfologisesti annotoituja korpuksia ei ole ollut avoimesti saatavilla. Tässä väitöskirjassa esitelty tutkimus hyödyntää vastikään julkaistuja suomen kie-

len dependenssijäsennettyjä FinnTreeBank ja Turku Dependency Treebank korpuksia. Lisäksi tutkimus hyödyntää suomen kielen avointa morfologista OMorFi-jäsennintä.

Väitöskirja esittelee menetelmiä jäsennystarkkuuden parantamiseen ja jäsentimen opetusnopeuden sekä jäsennysnopeuden kasvattamiseen. Väitöskirja esittää uuden tavan rakentaa generatiivisia jäsentimiä hyödyntäen painollisia äärellistilaisia koneita ja soveltaa tällaisia jäsentimiä suomen kielen kontekstisensitiiviseen oikeinkirjoituksentarkistukseen. Lisäksi väitöskirja käsittelee diskriminatiivisia jäsennysmalleja. Se esittelee tapoja hyödyntää morfologisten analyysien osia jäsennystarkkuuden parantamiseen. Lisäksi se esittää kaskadimallin, jonka avulla jäsentimen opetusaika lyhenee huomattavasi. Väitöskirja esittää myös tapoja jäsenninmallien pienentämiseen. Lopuksi esitellään FinnPos, joka on kirjoittaman toteuttama avoimen lähdekoodin työkalu disambiguoivien morfologisten jäsentimien opettamiseen.

General Terms:
morphological tagger, morphological analyzer, POS tagging, HMM, CRF, perceptron

Additional Key Words and Phrases:
morphologically complex languages

Preface

The work presented in this thesis was conducted at the University of Helsinki in the HFST research group led by Krister Lindén. I started the work in 2010 at the Department of General Linguistics and continued at the Department of Modern Languages after the linguistics departments of the Faculty of Humanities at the University of Helsinki merged. The work was financially supported by Langnet (Finnish doctoral program in language studies). I also received a three month finalization grant from the University of Helsinki.

I am very grateful to my advisors Krister Lindén and Anssi Yli-Jyrä. Discussions with Krister and Anssi have given rise to many fruitful ideas. I hope that some of those ideas have found their way into the publications that accompany this thesis. Krister and Anssi have also been a source of encouragement throughout my graduate studies and their work has served as an example of how to conduct research.

Måns (or Mans) Huldén and Kimmo Koskenniemi helped me a lot, especially in the early stages of my graduate studies. It has been a privilege to benefit from their vision and understanding of Natural Language Processing, particularly in the field of finite-state methods. I am also grateful to Måns for useful comments on this thesis as well as for many interesting joint projects in the past years. Both Måns and Kimmo have a wonderful approach to teaching. Their way of presenting material to students has served, and will keep serving, as a source of inspiration.

Without Teemu Ruokolainen this thesis would probably not exist. I am extremely lucky to have ended up in the same graduate school with Teemu. Throughout this process, I have been able to rely on his level-headed approach to machine learning, his clear thinking and his companionship.

The HFST research group has been tremendously important for me during the years of my graduate studies. I could not have wished for better colleagues than Erik Axelson, Senka Drobac, Sam Hardwick, Pekka Kauppinen and Tommi Pirinen. Working in the HFST group has taught me everything I know about building scientific software. Often this knowledge has been accumulated through a process of trial and error. While this has felt painful at times, it has nevertheless been extremely useful.

Jack Rueter employed me for three months for a project combining morphological analyzers and OCR engines. While I am not sure the work benefited my thesis, it was a fun project and the money was welcome.

Whenever I needed to travel or organize anything, Johanna Ratia was always able to help. I am not

sure I could have managed without her help. In any case, I am very happy that did not have to.

I wish to express my sincerest gratitude to the preliminary examiners of this work, Professor Joakim Nivre and Professor Jan Hajič. They provided useful comments on the manuscript. I also want to thank Professor Jörg Tiedemann for helpful comments on the thesis. I hope I have been able to incorporate their insights into the present work.

I am grateful to my family and friends for providing support and a welcome diversion during the process of my graduate studies. My family, mom, dad, Emmi and Max, and my friends, Veera, Jarmo, Antti, Marko, Reetta, Kaj and many others have made the process of getting a PhD degree bearable.

Lastly, I would like to thank Danny for his patience and support. I know that I spent many more nights writing this thesis than you would have liked. And, unfortunately, I spent a good deal of those solving tsumego instead of actually writing. Still, I think you were able to curb my procrastination to some extent.

Helsinki, September 20, 2016,

Miikka Silfverberg

Contents

List of Publications

I Silfverberg, M. and Lindén, K. (2010). Part-of-speech tagging using parallel weighted finite-state transducers. In *Proceedings of the 7th International Conference on NLP*, IceTAL2010, pages 369–380. Springer Berlin Heidelberg

II Silfverberg, M. and Lindén, K. (2011). Combining statistical models for POS tagging using finite-state calculus. In *Proceedings of the 18th Conference of Computational Linguistics*, NODALIDA-2011, pages 183–190. Northern European Association for Language Technology

III Pirinen, T., Silfverberg, M., and Lindén, K. (2012). Improving finite-state spell-checker suggestions with part of speech n-grams. In *Proceedings of the 13th International Conference on Intelligent Text Processing and Computational Linguistics*, CICLing2012. Springer Berlin Heidelberg

IV Ruokolainen, T., Silfverberg, M., Kurimo, M., and Linden, K. (2014). Accelerated estimation of conditional random fields using a pseudo-likelihood-inspired perceptron variant. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, EACL2014, pages 74–78. Association for Computational Linguistics

V Silfverberg, M., Ruokolainen, T., Lindén, K., and Kurimo, M. (2014). Part-of-speech tagging using conditional random fields: Exploiting sub-label dependencies for improved accuracy. In *Proceed-*

ings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL2014, pages 259–264. Association for Computational Linguistics

VI Silfverberg, M., Ruokolainen, T., Lindén, K., and Kurimo, M. (2015). FinnPos: an open-source morphological tagging and lemmatization toolkit for Finnish. *Language Resources and Evaluation*, pages 1–16. Online first article not yet assigned to an issue

Author's Contribution

Publication I: Part-of-Speech Tagging Using Parallel Weighted Finite-State Transducers

The paper presents a weighted finite-state implementation for hidden Markov models. I developed the system, performed the experiments and wrote the paper under supervision of the second author Krister Lindén.

Publication II: Combining Statistical Models for POS Tagging using Finite-State Calculus

The paper presents a continuation of the work in publication **I**. It presents extensions to the standard hidden Markov Model, which can be implemented using weighted finite-state calculus. I developed the system, performed the experiments and wrote the paper under supervision of the second author Krister Lindén.

Publication III: Improving Finite-State Spell-Checker Suggestions with Part of Speech N-Grams

The paper presents a spell-checker, which utilizes the POS taggers presented in publications **I** and **II** for language modeling. I performed the experiments together with the first author Tommi Pirinen and participated in writing the paper under supervision of the third author Krister Lindén.

Publication IV: Accelerated Estimation of Conditional Random Fields using a Pseudo-Likelihood-inspired Perceptron Variant

The paper presents a variant of the perceptron algorithm inspired by the pseudo-likelihood estimator for conditional random fields. I performed the experiments and participated in the writing of the paper under supervision of the third author Krister Lindén.

Publication V: Part-of-Speech Tagging using Conditional Random Fields: Exploiting Sub-Label Dependencies for Improved Accuracy

The paper presents a system which uses dependencies of the components of structured morphological labels for improving tagging accuracy. I implemented the system presented in the paper, performed the experiments and participated in the writing of the paper under supervision of the third author Krister Lindén.

Publication VI: FinnPos: an open-source morphological tagging and lemmatization toolkit for Finnish

The paper presents a morphological tagging toolkit for Finnish and other morphologically rich languages. The present author and the second author Teemu Ruokolainen designed the system and the methodology of the paper together. I implemented the FinnPos toolkit presented in the paper, performed the experiments and participated in the writing of the paper under supervision of the third author Krister Lindén.

Notation

Acronyms

CRF	Conditional Random Field
HMM	Hidden Markov Model
MEMM	Maximum-Entropy Markov Model
NB	Naïve Bayes
NLP	Natural Language Processing
PGM	Probabilistic Graphical Model
POS	Part-of-Speech
MAP	Maximum a posteriori

Mathematical Notation

$x[i]$	The element at index i in vector (or sequence) x.
$\mathbb{R}_n^m$	The space of $m \times n$ real valued matrices.
$x[s:t]$	Sub-sequence $(x_s,\ ...,\ x_t)$ of sequence $x = (x_1,\ ...,\ x_T)$.
X^t	The cartesian product of t copies of the set X.
$M^\top$	Transpose of matrix M.
$f(x;\theta)$	Function f, paramerized by vector θ, applied to x.
$x \mapsto f(x),\ x \overset{f}{\mapsto} f(x)$	A mapping of values x to expressions $f(x)$.
$\|v\|$	Norm of vector v.
$\hat{p}$	Estimate of probability p.

Chapter 1

Introduction

A morphological tagger is a piece of computer software that provides complete morphological descriptions of sentences. An example of a morphologically tagged sentence is given in Figure 1.1. Each of the words in the sentence is assigned a detailed morphological label, which specifies part-of-speech and inflectional information. Each word also receives a lemma. Morphological tagging is typically a pre-processing step for other language processing applications, for example syntactic parsers, machine translation software and named entity recognizers.

The task of morphological tagging is closely related to part-of-speech tagging (POS tagging), where the words in a sentence are tagged using coarse morphological labels such as noun and verb. These typically correspond to main word classes. POS taggers are sufficient for processing languages where the scope of productive morphology is restricted, for example English. Morphological taggers are, however, necessary when processing *morphologically complex languages*, which extensively utilize inflection, derivation and compounding for encoding structural and semantic information. For these languages, a coarse POS tag simply does not provide enough information to enable accurate downstream processing such as syntactic parsing.[1]

`Article+Indef`	`Noun+Sg`	`Verb+Pres+3sg`	`Prep`	`Article+Def`	`Noun+Sg`	.
A	DOG	SLEEP	ON	THE	MAT	.
A	dog	sleeps	on	the	mat	.

Figure 1.1: A morphologically tagged sentence

At first glance, the task of assigning morphological descriptions, or *morphological labels*, seems al-

[1] For example in Finnish, the subject and object of a sentence are distinguished by case and different verbs can require different cases for their arguments Hakulinen et al. (2004). Coarse POS tags do not capture such distinctions. Therefore, accurate parsing of Finnish cannot rely solely on coarse POS tags.

most trivial. Simply form a list of common word forms and their morphological labels and look up words in the list when tagging text. Unfortunately, this approach fails because of the following reasons.

1. A single word form can get several morphological labels depending on context. For example “dog” and “man” can be both nouns and verbs in English.

2. For morphologically complex languages, it is impossible to form a list of common word forms which would have sufficient coverage (say, higher than 95%) on unseen text.

Due to the reasons mentioned above, a highly accurate morphological tagger must model the context of words in order to be able to disambiguate between their alternative analyses. Moreover, it has to model the internal structure of words in order to be able to assign morphological labels for previously unseen word forms based on similar known words.

This thesis presents work on building morphological taggers for morphologically complex languages, in particular Finnish, which is the native language of the author. The thesis focuses on data-driven methods which utilize manually prepared training corpora and machine learning algorithms for learning tagger models.

1.1 Motivation

Data-driven methods have dominated the field of natural language processing (NLP) since the 1990’s. Although these methods have been applied to virtually all language processing tasks, research has predominantly focused on a few languages, English in particular. For many languages with fewer speakers, such as Finnish, statistical methods have not been applied to the same extent. This is probably a result of the fact that large training corpora required by supervised data-driven methods are available for very few languages.

The relative lack of work on statistical NLP for languages besides English is a problem for NLP as a field of inquiry because the languages of the world differ substantially with regard to syntax, morphology, phonology and orthography. These differences have very real consequences for the design of NLP systems. Therefore, it is impossible to make general claims about language processing without testing the claims on other languages in addition to English.

This thesis presents work that focuses on data-driven methods for morphological tagging of Finnish. Finnish and English share many characteristics but also differ in many respects. Both are written in Latin script using similar character inventories, although Finnish orthography uses three characters usually not found in English text “å”, “ä” and “ö”. Moreover, there are similarities in the lexical inventories of the languages because, like many modern languages, Finnish has been influenced by English and because both languages are historically associated with Germanic and Nordic languages. In some respects, however, Finnish and English are vastly different. Whereas English has fixed SVO word order, the word order in Finnish is quite flexible. Another major difference is the amount of inflectional morphology. For

example, English nouns usually only occur in three inflected forms: singular “cat”, plural “cats” and possessed “cat’s”. In contrast, thousands of inflected forms can be coined from a single Finnish noun.

Although data driven methods have dominated the field of POS tagging and, to a lesser extent, morphological tagging for the last twenty years, data driven work on Finnish morphological tagging has been scarce mostly because of the lack of high quality manually annotated broad coverage training corpora. However, other approaches like the purely rule based constraint grammar (Karlsson et al., 1995) and its derivative functional dependency grammar (Tapanainen and Järvinen, 1997) have been successfully applied for joint morphological tagging and shallow parsing.[2]

The recently published FinnTreeBank (Voutilainen, 2011) and Turku Dependence Treebank (Haverinen et al., 2014) represent the first freely available broad coverage Finnish manually prepared data sets that can be used for work on morphological tagging. These resources enable experiments on statistical morphological tagging for Finnish using a convincing gold standard corpus. Moreover, the broad coverage open-source Finnish morphological analyzer OMorFi (Pirinen, 2011) is a valuable resource for improving the performance of a tagging system.

The complex morphology present in the Finnish language leads to problems when existing tagging algorithms are used. The shear amount of possible morphological analyses for a word slows down both model estimation and application of the tagger on input text. Moreover, the large amount of possible analyses causes data sparsity problems.

Data driven methods typically perform much better on word forms seen in the training data than on out-of-vocabulary (OOV) words, that is words which are missing from the training data. In the case of English, this is usually not detrimental to the performance of the tagger. Especially when the training and test data come from the same domain, the amount of OOV words is typically rather low and the impact of OOV words on accuracy is consequently small. In contrast, this becomes a substantial problem when applying purely data driven systems on morphologically complex languages because productive compounding and extensive inflection lead to a large amount of OOV words even within one domain.

1.2 Main Contributions

This thesis presents an investigation into data-driven morphological tagging of Finnish both using generative and discriminative models. The aim of my work has been creation of practicable taggers for morphologically complex languages. Therefore, the main contributions of this thesis are practical in nature. I present methods for improving tagging accuracy, estimation speed, tagging speed and reducing model size. More specifically, the main contributions of the thesis are as follows.

- **A novel formulation of generative morphological taggers using weighted finite-state machines** Finite-state calculus allows for flexible model specification while still guaranteeing efficient application of the taggers. Traditional generative taggers, which are based on the Hidden Markov Model

[2] For example, the Finnish constraint grammar tagger FinCG is available online through the GiellaTekno Project `https://victorio.uit.no/langtech/trunk/kt/fin/src/fin-dis.cg1` (fetched on February 24, 2016).

(HMM), employ a very limited feature set and changes to this feature set require modifications to the core algorithms of the taggers. Using weighted finite-state machines, a more flexible feature set can, however, be employed without any changes to the core algorithms. This work is presented in Publications **I** and **II**.

- **Morphological taggers and POS taggers are applied to context sensitive spelling correction** Typically, context sensitive spelling correctors rely on neighboring words when estimating the probability of correction candidates. For morphologically complex languages, this approach fails because of data sparsity. Instead, a generative morphological tagger can be used to score suggestions based on morphological context as shown in Publication **III**.

- **Feature extraction specifically aimed at morphologically complex languages** As mentioned above, the large inventory of morphological labels causes data sparsity problems for morphological tagging models such as the averaged perceptron and conditional random field. Using sub-label dependencies presented in Publication **V**, data sparsity can, however, be alleviated. Moreover, sub-label dependencies allow for modeling congruence and other similar syntactic phenomena.

- **Faster estimation for perceptron taggers** Exact estimation and inference is infeasible in discriminative taggers for morphologically complex languages because the time requirement of exact estimation and inference algorithms depends on the size of the morphological label inventory which can be quite large. Some design choices (like higher model order) can even be impossible for morphologically complex languages using standard tagging techniques. Although the speed of tagging systems is not always seen as a major concern, it can be important in practice. A faster and less accurate tagger can often be preferable compared to more accurate but slower taggers in real world applications where high throughput is vital. Estimation speed, in turn, is important because it affects the development process of the tagger. For these reasons, Publications **IV** and **V** explore known and novel approximate inference and estimation techniques. It is shown that these lead to substantial reduction in training time and faster tagging time compared to available state-of-the-art tagging toolkits.

- **Pruning strategies for perceptron taggers** Model size can be a factor in some applications. For example, when using a tagger on a mobile device. In Chapter 6, I review different techniques for feature pruning for perceptron taggers and present some experiments on feature pruning in Chapter 8.

- **FinnPos toolkit.** Publication **VI** presents FinnPos, an efficient open source morphological tagging toolkit for Finnish and other morphologically complex languages. Chapter 8 presents a number of experiments on morphological tagging of Finnish using the FTB and TDT corpora. These experiments augment the results presented in Publication **VI**.

1.3 Research Questions

The work presented in this thesis mainly consists of practical contributions to the field of morphological tagging. However, the thesis also investigates a number of questions related to the design of morphological taggers. There are various approximations and optimizations which are required to build accurate and efficient morphological taggers and POS taggers. Examples include beam search, which is used to speed up training, and higher model order, which improves accuracy. Although some of these techniques are widely applied in POS tagging, it is interesting to study their effect in tagging of morphologically complex languages where label sets are large and the amount of OOV words is high. In Chapter 8, I have explored the impact of these techniques on estimation speed and tagging accuracy. The questions that are investigated in Chapter 8 fall into the following categories.

- **Approximate Estimation** In presence of large label sets, exact estimation of the tagger model is inconvenient or even impossible because of prohibitive computational cost. Therefore, different approximations such as beam search and label guessing, presented in Chapter 6, have to be utilized during estimation. I have investigated the impact of these approximations on accuracy and training time.

- **Improvements to Accuracy** Several different methods can be used to improve the accuracy of standard morphological taggers. These include increased model order, using lexical resources such as morphological analyzers and utilizing sub-label dependencies presented in Chapter 6. I was interested in exploring the impact of these methods on accuracy. Especially, I wanted to investigate the relative magnitude of the impact of different optimizations. I was also interested in knowing if the optimizations have a cumulative effect, that is, if a combination of several of these methods delivers greater improvement in accuracy than any of the methods in isolation.

- **Model Pruning** Morphological tagger models can give rise to very large binary files because the large amount of features that are extracted from the training data. I investigated two straight forward methods, value based and update count based filtering presented in Chapter 6, for reducing model size by filtering out parameters which are likely to have small impact on tagging accuracy. I was interesting in comparing the impact of these methods on tagging accuracy and model size.

1.4 Outline

This thesis can be seen as an introduction to the field of morphological tagging and the techniques used in the field. It should give sufficient background information for reading the articles that accompany the thesis. Additionally, Chapter 8 of the thesis presents detailed experiments using the FinnPos morphological tagger that were not included in Publication **VI**.

Chapter 2 establishes the terminology on morphology and morphological tagging as well as surveys the field of morphological tagging. Chapter 3 is a brief introduction to supervised machine learning and the experimental methodology of natural language processing. In Chapter 4, I introduce generative data-driven models for morphological tagging. Chapter 5 introduces finite-state machines and a formulation of generative morphological taggers in finite-state algebra. It also shows how finite-state algebra can be used to formulate generative taggers in a generalized manner encompassing both traditional HMM taggers and other kinds of models. Chapter 6 deals with discriminative morphological taggers and introduces contributions to the field of discriminative morphological tagging. Chapter 7 deals with the topic of data-driven lemmatization. Experiments on morphological tagging using the FinnPos toolkit are presented in Chapter 8. Finally, the thesis is concluded in Chapter 9.

Chapter 2

Morphology and Morphological Tagging

This Chapter introduces the field of linguistic morphology and morphological tagging. It will also present an overview of the current state-of-the-art in morphological tagging.

2.1 Morphology

Words Words are the most readily accepted linguistic units at least in Western written language. I define a word as a sequence of letters, and possible numbers, which is surrounded by white-space or punctuation. Matters are more complex in spoken language, written languages that do not use white space (such as Chinese), and sign language. Still, this definition covers most cases of interest from the point of view of the field of morphological tagging.

Morphemes Morphology is the sub-field of linguistics that studies the internal structure of words. According to Bybee (1985), morphology has traditionally been concerned with charting the *morpheme inventory* of language. That is, finding the minimal semantic units of language and grouping them into classes according to behavior and meaning. For example, the English word form "dogs" consists of two morphemes "dog" and "-s". The first one is a *word stem* and the second one is an *inflectional affix* marking plural number.

(Non-)Concatenative Morphology In many languages, such as English, words are mainly constructed by concatenating morphemes. For example,"dog" and "-s" can be joined to give the plural "dogs". This is called *concatenative morphology*. There are many phenomena that fall beyond the scope of concatenative morphology. For example, English plural number can be signaled by other, less transparent, means as demonstrated by the word pairs "m**ou**se/m**i**ce" and "m**a**n/m**e**n". In these examples, choice of vowel

indicates number. This type of inflection is called *ablaut*. In general, phenomena that fall beyond the scope of simple concatenation are called *non-concatenative*.

Cross-linguistically, the most common form of non-concatenative morphology is *suppletion*. Suppletion is the irregular relationship between word forms exemplified by the English infinitive verb "go" and its past tense "went". Such irregularity occurs in all languages. Even though suppletion is cross-linguistically common, most lexemes in a language naturally adhere to regular inflection patterns. For example, most English verbs form a past tense by adjoining the suffix "-ed" onto the verb stem.

Morphophonological alternations are a further example of non-concatenative morphology. They are sound changes that occur at morpheme boundaries. A cross-linguistically common example is nasal assimilation (Carr, 1993, p. 29), where the place of articulation of a nasal depends on the following stop. As an example, consider the English prefix "in-". The "n" in "input" and "inset" is pronounced as "m" and "n", respectively.

Languages differ with regard to the amount of non-concatenative morphology. Some, like Turkish, employ almost exclusively concatenation. Such languages are called *agglutinative*. Others, such as English, employ a mix of concatenation and non-concatenative phenomena. These languages are usually called *fusional*. Still, concatenative morphology is probably found to some degree in all languages. It is especially prevalent in languages with complex morphology, such as Finnish or Turkish. From the point of view of language technology for morphologically complex languages, it is therefore of paramount importance to be able to handle concatenative morphology.

Morphotax Stems in English can often occur on their own as words and are therefore called *free morphemes*. Inflectional affixes cannot. Therefore, they are called *bound morphemes*. Such restrictions belong to *morphotax*: the sub-field of morphology concerned with defining the rules that govern the concatenative morphology of a language. For example, "dog" and "dog+s" are valid from the point of view of English morphotax whereas "dogdog" and "s" (in the meaning plural number) are not.

Word Class The word forms "dogs" and "cats" share a common number marker "-s" but they have different stems. Still, there is a relation between the stems "dog" and "cat" because they can occur with similar inflectional affixes and in similar sentence contexts. Therefore, they can be grouped into a common *word class* or *part-of-speech*, namely nouns. The inventory of word classes in a language cannot be determined solely based on word internal examination. Instead, one has to combine knowledge about the structure of words with knowledge about interaction of the words in sentences. The concept of word class, therefore, resides somewhere between the linguistic disciplines morphology and *syntax* which is the study of combination of words into larger units: phrases and sentences.

Lexeme and Lemma Word forms such as "dog" "dogs" and "dog's" share a common stem "dog". Each of the word forms refers to the concept of dog, however, different forms of the word are required depending on context. Different word forms, that denote the same concept, belong to the same *lexeme*. Each lexeme has a *lemma* which is a designated word form representing the entire lexeme. In the case of English nouns,

the lemma is simply the singular form, for example “dog”. In the case of English verbs, the infinitive, for example “to run”, is usually used. The particular choice depends on linguistic tradition.

Lemmas are important for language technology because dictionaries and word lists, which can be used to derive information about lexemes, usually contain lemmas. Therefore, it is useful to be able to *lemmatize* a word form, that is produce the lemma from a given word form.

Categories of Bound Morphemes Whereas free morphemes are grouped into word classes, bound morphemes are grouped into their own categories according to meaning and co-occurrence restrictions. For example, Finnish nouns can take a plural number marker. Additionally, they can take one case marker from an inventory of 15 possible case markers, one possessive suffix from an inventory of 6 possible markers and a number of clitic affixes (Hakulinen et al., 2004). The categories of bound morphemes can belong to one particular word class, however, several word classes may also share a particular class of bound morphemes. For example, both adjectives and nouns take a number in English.

Morphological analysis In many applications such as information retrieval and syntactic parsing, it is useful to be able to provide an exhaustive description of the morphological information associated with a word form. Such a description is called a *morphological analysis* or *morphological label* of the word form. For example, the English word form “dogs” could have a morphological analysis “dog+Noun+Plural”. The granularity and appearance of the morphological analysis depends on linguistic tradition and the linguistic theory which is being applied, however, the key elements are the lemma of the word form as well as a list of the bound morphemes associated to the word form.

2.2 Morphological Analyzers

Word forms in natural languages can be ambiguous. For example, the English “dogs” is both a plural form of a noun and the present third person singular form of a verb. The degree of ambiguity varies between languages. To some degree, it is also a function of the morphological description: a coarse morphological description results in less ambiguity than a finer one. A *morphological analyzer* is a system which processes word forms and returns the complete set of possible morphological analyses for each word form.

Applications Morphological analyzers are useful both when the lemma of the word is important and when the information about bound morphemes is required. The lemma is useful in tasks where the semantics of the word form is of great importance. These task include information extraction and topic modeling. In contrast, bound morphemes predominantly convey structural information instead of semantic information. Therefore, they are more important for syntactic parsing and shallow parsing which aim at uncovering the structure of linguistic utterances.

Motivation The need for full scale morphological analyzers has been contested. For example, Church (2005) has argued that practical applications can mostly ignore morphology and focus on processing raw word forms instead of morphologically analyzed words. This may be a valid approach for English and other languages which mainly utilize syntactic means like word order to express grammatical information, especially when large training corpora are available. In these languages the number of word forms in relation to lexemes tends to be low. For example, in the Penn Treebank of English Marcus et al. (1993) spanning approximately 1 million words, three distinct word forms occur which have the lemma "dog", namely "dog", "dogs" and "dogged". It can be argued that no specific processing is required to process English word forms.

In contrast to English, many languages do utilize morphology extensively. For example, although the Finnish FinnTreeBank corpus (Voutilainen, 2011) only spans approximately 160,000 words, there are 14 distinct word forms which have the lemma "koira" (the Finnish word for "dog").[1] In total, the Penn Treebank contains some 49,000 distinct word forms whereas the FinnTreeBank contains about 46,000 word forms even though it is only 20% of the size of the English corpus. These considerations illustrate the need for morphological processing for morphologically complex languages like Finnish which make extensive use of inflective morphology. Methods which rely purely on word forms will simply suffer too badly from data sparsity. The experiments presented in Chapter 8 show that a morphological tagger is vital for accurate morphological tagging of Finnish.

Variants There are different types of morphological analysis systems. The first systems used for English information retrieval were *stemmers*, the most famous system being the Porter stemmer (Porter, 1997). It uses a collection of rules which strip suffixes from word forms. For example, "connect", "connection" and "connected" would all receive the stem "connect". The system does not rely on a fixed vocabulary and can thus be applied to arbitrary English word forms. The Porter stemmer, and stemmers in general, are sufficient for information retrieval in English but they fall short when more elaborate morphological information is required, for example, in parsing. Moreover, they are too simplistic for morphologically complex languages like Finnish and Turkish.[2]

Morphological segmentation software, such as Morfessor (Creutz and Lagus, 2002), are another type of morphological analyzers often utilized in speech recognition for languages with complex morphology. The Morfessor system splits word forms into a sequence of morpheme-like sub-strings. For example, the word form "dogs" could be split into "dog" and "-s". This type of morphological segmentation is useful in a wide variety of language technological applications, however it is more ambiguous than a traditional morphological analysis where the bound morphemes are represented by linguistic labels such as plural. Moreover, Morfessor output does not contain information about morphological categories that are not overtly marked. For example, in Finnish, the singular number of nouns is not overtly marked (only plural number is marked by an affix "-i-"). Although not overtly marked, such information can very useful for

[1] If different compound words of "koira", such as "saksanpaimenkoira" (German Shepard) are considered, there are 23 forms of koira in the FinnTreeBank corpus.

[2] However, they may suffice in some domains. Kettunen et al. (2005) show that a more elaborate stemmer, which can give several stem candidates for a word form, can perform comparable to a full morphological analyzer) in information retrieval.

further processing.

The current state-of-the-art for morphological analysis of morphologically complex languages are *finite-state morphological analyzers* (Kaplan and Kay, 1994, Koskenniemi, 1984). Full scale finite-state analyzers can return the full set of analyses for word forms. They can model morphotax and morphophonological alternations using finite-state rules and a finite-state lexicon (Beesley and Karttunen, 2003). In contrast to stemmers, which are quite simple, and segmentation systems like Morfessor which can be trained in an unsupervised manner, full-scale morphological analyzers typically require a lot of manual work. The most labor intensive part of the process is the accumulation of the lexicon.

Although, full-scale morphological analyzers require a lot of manual work, the information they produce is very reliable. Coverage is a slight problem because lemmas typically need to be manually added to the system before word forms of that lemma can be analyzed. However, morphological guessers can be constructed from morphological analyzers (Lindén, 2009). These extend the analyzer to previously unseen words based on similar words that are known to the analyzer.

The morphological analyzer employed by the work presented in this thesis is the Finnish Open-Source Morphology (OMorFi) (Pirinen, 2011). It is a morphological analyzer of Finnish implemented using the open-source finite-state toolkit HFST[3] (Lindén et al., 2009) and is utilized for the experiments presented in Chapter 8.

2.3 Morphological Tagging and Disambiguation

I define morphological tagging as the task of assigning each word in a sentence a unique morphological analysis consisting of a lemma and a morphological label which specifies the part-of-speech of the word form and the categories of its bound morphemes. This contrasts with POS tagging, where the task is to provide a coarse morphological description of each word, typically its part-of-speech.

One interesting aspect of the morphological tagging task is that both the set of potential inputs, that is sentences, and potential outputs, that is sequences of analyses, are unfathomably large. Since each word in a sentence $x = x_1, \ ..., \ x_T$ of length T receives one label, the complete sentence has n^T possible label sequences $y = y_1, \ ..., \ y_T$ when there are n possible labels for an individual word. Given a sentence of 40 words and a label set of 50 labels, the number of possible label sequence is thus $40^{50} \approx 10^{80}$ which according to Wolfram Alpha[4] is the estimated number of atoms in the observable universe.

The exact number of potential English sentences of any given length, say ten, is difficult to estimate because all strings of words are not valid sentences.[5] However, it is safe to say that it is very large – indeed much larger than the combined number of sentences in POS annotated English language corpora humankind will ever produce. Direct estimation of the conditional distributions $p(y \mid x)$, for POS label sequences y and sentences x, by counting is therefore impossible.

[3] `http://hfst.github.io/`

[4] `http://www.wolframalpha.com/input/?i=number+of+atoms+in+the+universe`

[5] Moreover, it is not easy to say how many word types the English language includes.

Because the POS labels of words in a sentence depend on each other, predicting the label y_t for each position t separately is not an optimal solution. Consider the sentence "The police dog me constantly although I haven't done anything wrong!". The labels of the adjacent words "police", "dog", "me" and "constantly" help to disambiguate each other. A priori, we think that "dog" is a noun since the verb "dog" is quite rare. This hypothesis is supported by the preceding word "police" because "police dog" is an established noun–noun collocation. However, the next word "me" can only be a pronoun, which brings this interpretation into question. The fourth word "constantly" is an adverb, which provides additional evidence against a noun interpretation of "dog". In total, the evidence points toward a verb interpretation for "dog".

The disambiguation of the POS label for "dog" utilizes both so called *unstructured* and *structured* information. The information that "dog" is usually a noun is unstructured information, because it refers to the POS label (the prediction) of an individual word "dog". The information, that words which precede pronouns are much more likely to be verbs than nouns, is a piece of structured information because it refers to the combination of several POS labels. Both kinds of information are very useful, but a model which predicts the label y_t for each position in isolation cannot utilize structured information.

Even though structured information is quite useful, this usefulness has limitations. For example, the labels of "dog" and "anything" in the example are not especially helpful for disambiguating each other. It is a sensible assumption that the further apart two words are situated in the sentence, the less likely it is that they can significantly aid in disambiguating each other. However, this does not mean that the interpretations of words that are far apart cannot depend on each other – in fact they frequently do. For example, embedded clauses and co-ordination can introduce long range dependencies inside sentences. Sometimes, even words in another sentence may help in disambiguation. It is, however, difficult to utilize this information in a tagger because most words that lie far apart are useless for disambiguating each other's morphological labels, which makes estimation of statistics from data quite difficult.[6]

Traditionally, morphological taggers have been classified into two categories: *data-driven* and *rule-based*. Data-driven taggers primarily utilize morphologically labeled training data for learning a *model* that represents the relationship between text and morphological labels. The model is typically based on very simple facts called *features* that can be extracted from labeled text. For example **the second word in the sentence is "dog" and its label is noun+sg+nom** and **the second word has label noun+sg+nom and the third word has label verb+pres+3sg**. Each feature corresponds to a weight which determines its relative importance and reliability. During training, these weights are optimized to describe the relationship between sentences and label sequences as closely as possible. Given an unlabeled input sentence, it is possible to find the label sequence that the model deems most likely. Thus the model can be used for tagging.

In contrast to data-driven systems, rule-based, or *expert-driven*, taggers do not primarily rely on training data. Instead they utilize information provided by domain experts (linguists in this case) using some rule formalism. These rules are assembled into a grammar and compiled into instructions that can be interpreted by a computer. In contrast to the weighted features in data-driven systems, the rules in expert-

[6]The primary problem is that it is difficult to distinguish co-occurrence by chance from a genuine tendency.

driven systems are typically categorical, that is they either apply or do not apply.

The division into data-driven and expert-driven systems is not clear-cut. For example, data-driven statistical taggers often employ a morphological analyzer which is typically a rule-based system. Conversely, rule-based systems can utilize statistics to solve ambiguities which cannot be resolved solely based on grammatical information. As seen below, it is also possible to integrate a rule-based and data-driven approach more deeply into a *hybrid tagger*.

The Brill tagger (Brill, 1992) is one of the early successes in POS tagging. It is in fact a hybrid tagger. The tagger first labels data using a simple statistical model (a unigram model of the distribution of tags for each word form). It then corrects errors introduced by the simple statistical model using rules that can be learned from data or specified by linguists. Several layers of rules can be used. Each layer corrects errors of the previous layer. Although the Brill tagger is an early system, it might still be quite competitive as shown by Horsmann et al. (2015), who compared a number of POS taggers for English and German on texts in various domains (these experiments included state-of-the-art models such as the averaged perceptron). According to their experiments, the Brill tagger was both the fastest and most accurate.

One of the major successes of the rule-based paradigm is the Constraint Grammar formalism (Karlsson et al., 1995). The formalism uses finite-state disambiguation rules to disambiguate the set of morphological labels given by a morphological analyzer. The approach may still produce the most accurate taggers for English. Voutilainen (1995) cite an accuracy of 99.3% on English. Direct comparison of tagging systems based on accuracies reported in publications is, however, difficult because they are trained on different data sets and use different morphological label inventories but experiments conducted by Samuelsson and Voutilainen (1997) show that constraint grammar performed better than a state-of-the-art data-driven tagger at the time.

Although, there are many highly successful and interesting rule-based and hybrid systems, my main focus is data-driven morphological tagging. The first influential systems by Church (1988) and DeRose (1988) were based on Hidden Markov Models (HMM) which are presented in detail in Chapter 4. These early data-driven systems achieved accuracy in excess of 95% when tested on the Brown corpus (Francis, 1964). Later work by Brants (2000) and Halácsy et al. (2007) refined the approach and achieved accuracies in the vicinity of 96.5%. Publications **I** and **II** continue this work. They set up the tagger as a finite-state system and experiment with different structured models for the HMM tagger.

HMM taggers are so called generative statistical models. They specify a probabilistic distribution $p(x, y)$ over sentences x and label sequences y. In other words, these systems have to model both sentences and label sequences at the same time. Unfortunately, this is very difficult without making simplistic assumptions about the labeled data. For example, a standard assumption is that the probability of a word is determined solely based on its morphological label. This assumption is obviously incorrect as demonstrated by word collocations.

In order to be able to use more sophisticated features to describe the relation between the input sentence and its morphological labels, Ratnaparkhi (1997) used a discriminative classification model instead of a generative one. Whereas, a generative model represents a joint probability $p(x, y)$ for a sentence and label sequence, a discriminative model only represents the conditional probability $p(y|x)$ of label sequence

y given sentence x. This means that the sentence x no longer needs to be modeled. Therefore, more elaborate features can be used to describe the relation between sentences and morphological labels. The model still has to account for the internal structure of y but because y can be anchored much more closely to the input sentence, the accurate modeling of relations between the individual labels in y is not as important in a discriminative tagger.[7]

The Maximum Entropy Markov Model (MEMM) used by Ratnaparkhi (1997) is a structured model but it is trained in an unstructured fashion. For each training sentence $x = (x_1, ..., x_T)$ and its label sequence $y = (y_1, ..., y_T)$ the model is trained to maximize the probability $p(y_t|x, y_1, ..., y_{t-1})$ in each position t. This means that the model relies on correct label context during training time. This causes the so called *label bias problem* described by Lafferty et al. (2001). Essentially, label bias happens because the model relies too much on label context. Another form of bias, namely observation bias investigated by Klein and Manning (2002) may in fact be more influential for POS tagging and morphological tagging. These biases seem to have a real impact on tagging accuracy. In fact, Brants (2000) showed that it is possible for a well constructed generative tagger to outperform a MEMM tagger, although direct comparison is difficult because the test and training sets used by Ratnaparkhi and Brants differ. Additional support for the superiority of the HMM model, is however provided by Lafferty et al. (2001) whose experiments indicate that the performance of the MEMM is inferior to the HMM on simulated data when using the same set of features.

Berg-Kirkpatrick et al. (2010) propose a model for part-of-speech induction which is an unsupervised task related to POS tagging. The model can be seen as a hybrid of the HMM and MEMM models. Like an HMM, it has emission distributions and transition distributions (the traditional HMM model is described in Chapter 4). However, these distributions are modeled as local logistic regression models. It would be interesting to apply this model to morphological tagging in a supervised setting.

Lafferty et al. (2001) proposed Conditional Random Fields (CRF) as a solution to the label bias problem. The CRF is trained in a structured manner (it is a so called globally normalized model) and does not suffer from label or observation bias. According to their experiments, the CRF model outperforms both the HMM and MEMM in classification on randomly generated data and POS tagging of English when using the same feature sets. Moreover, the CRF can employ a rich set of features like the MEMM which further improves its accuracy with regard to the HMM model.

Another discriminative model, the averaged perceptron tagger, is proposed by Collins (2002). The model is a structured extension of the classical perceptron (Rosenblatt, 1958). The main advantage of the perceptron tagger compared to the CRF model is that it is computationally more efficient and also produces sparser models.[8] Its training procedure is also amenable to a number of optimizations like beam search. These are explored in Chapter 6. The main drawback is that, while the classification performance of the CRF and averaged perceptron tagger is approximately the same[9], the averaged perceptron tagger is

[7] This is illustrated by the fact that an unstructured discriminative model which does not model relations between labels at all fares almost as well on tagging the Penn Treebank as a structured model when the taggers use the same unstructured features. According to experiments performed by the author on the Penn Treebank the difference in accuracy can be as small as 0.4%-points (a drop from 97.1% to 96.7%). Dropping the structured features from a typical HMM tagger reduces performance substantially more.

[8] Although, different regularization methods can give sparse models also for the CRF.

[9] For example experiments performed by Nguyen and Guo (2007) indicate that the classification performance of the averaged

optimized only with regard to classification. It does not give a reliable distribution of alternative morphological tags which can sometimes be useful in downstream applications like syntactic parsers. Nevertheless, the averaged perceptron tagger and its extensions, like the margin infused relaxed algorithm (MIRA) (Taskar et al., 2004) and the closely related structured Support Vector Machine (SVM) (Tsochantaridis et al., 2005), are extensively applied in sequence labeling tasks such as POS tagging.

The CRF, averaged perceptron, SVM and other related classifiers can be seen as alternative estimators for hidden Markov models (a terminology used by for example Collins (2002)) or linear classifiers. For example Publication **IV** and Nguyen and Guo (2007) explore different estimators for linear classifiers and compare them.

While these models have been extensively investigate for POS tagging, the focus of this thesis is morphological tagging. Generative taggers such as the HMM have been applied to morphological tagging by for example Halácsy et al. (2007) and Publication **II** but as in the case of English, generative models cannot compete with discriminative models with regard to accuracy.

Morphological tagging using discriminative models has been investigated by Chrupala et al. (2008) who use a MEMM and Spoustová et al. (2009) who utilize an averaged perceptron tagger. However, these works do not adequately solve the problem of slow training times for morphological taggers in the presence of large label sets. Spoustová et al. (2009) use a morphological analyzer to limit label candidates during training. This is a plausible approach when a morphological analyzer is available and when its coverage is quite high. This, however, is not always the case. Moreover, using only the candidates emitted by an analyzer during training can degrade classification performance. Publication **VI** and the experiments in Chapter 8 present alternative methods for accelerating model estimation using a cascaded model architecture.

The structure present in large morphological label sets can be leveraged to improve tagging accuracy. For example, it is possible to estimate statistics for sub-labels, such as “noun”, of complex labels “noun+sg+nom”. This approach is explored by for example Spoustová et al. (2009) who extract linguistically motivated sub-label features. Publication **V** further investigate this approach and shows that general unstructured and structured sub-label features lead to substantial improvement in accuracy. Additional experiments are reported in Chapter 8.

Recently, Müller et al. (2013) applied a cascaded variant of the CRF model to morphological tagging of several languages in order to both speed up training and combat data sparsity. Publications **V** and **VI** continue this line of research by setting up a cascade of a perceptron classifier and a generative classifier used for pruning label candidates. This combination delivers competitive results compared to the cascaded CRF approach as demonstrated by Publication **VI** while also delivering faster training times.

Morphological tagging can be done concurrently with parsing. Bohnet et al. (2013) present experiments on the Turku Dependency Treebank also used in Publication **VI**. Although, the data splits are different, it seems that the tagging results obtained in Publication **VI** are still better than the results of joint tagging and parsing.

Morphological tagging includes the task of lemmatization. Chrupala et al. (2008) sets up this task as a

perceptron algorithm can in fact be better than the performance of the Conditional Random field.

classification task as explained in Chapter 7 and Publication **VI** mostly follows this approach. Müller et al. (2015) explore joint tagging and lemmatization and shows that this improves both tagging and lemmatization results. Although, it would be very interesting to experiment with joint tagging and lemmatization, it remains future work for the author.

Data-driven classifiers can also be used for morphological disambiguation and, as the experiments in Chapter 8 demonstrate, the combination of a morphological analyzer and discriminative tagger performs substantially better than a purely data-driven morphological tagger. There are two principal approaches to data-driven morphological disambiguation. Firstly, the analyzer can simply be used to limit label candidates. For example, for English, the word “dog” could receive a verb label and a noun label but not a determiner label. The second approach is to use the morphological analyzer in feature extraction. In discriminative taggers, the labels and label sets given by the morphological analyzer can be directly used as features.

This thesis will mainly be concerned with a data-driven supervised learning setting but semi-supervised systems and hybrid systems that combine data-driven and linguist driven methods have also been investigated in the field. Spoustová et al. (2009) and Søgaard (2011) apply self-training where a large amount of unlabeled text is first tagged and then used to train a tagger model in combination with hand annotated training data. This leads to significant improvements for English and Czech. Spoustová et al. (2009) additionally uses a voting scheme where different taggers are combined for improved accuracy. This remains future work for the author.

Hulden and Francom (2012) investigate various combinations of HMM models and Constraint Grammars for tagging. They show that a hybrid approach can lead to improved tagging accuracy and also reduced rule development time. A nearly identical setup was also explored by Orosz and Novák (2013). A very similar setup was also used by Spoustová et al. (2007) who examined combinations of hand-written rules (very similar to constraint grammar rules) and an HMM, perceptron tagger and a MEMM. While semi-supervised training and hybrid methods are very interesting, they remain future work for the author at the present time.

Chapter 3

Machine Learning

This section outlines the basic methodology followed in machine learning research for NLP. I will briefly discuss machine learning from a general point of view and then present supervised machine learning in more detail using linear regression as example.

Supervised and Unsupervised ML There exists a broad division of the field of machine learning into three sub-fields.[1]

1. In *supervised* machine learning the aim is to learn a mapping from inputs x (such as sentences) to outputs y (such as morphological label sequences). To this aim, a supervised system uses training material consisting of input-output pairs (x, y) and a model which can represent the mapping $x \mapsto y$. Training of the model consists of tuning its parameters in such a way that the model accurately describes mapping between the inputs and outputs in the training data. Typically, supervised machine learning is employed for tasks such as classification and regression. Examples in the field of natural language processing include POS tagging and other tasks that can be framed as labeling (for example named entity recognition), speech recognition and machine translation.

2. In contrast to supervised machine translation, *unsupervised* approaches exclusively utilize unannotated data, that is the training data consists solely of inputs x. Unsupervised machine learning is most often used for various kinds of clustering tasks where inputs are grouped into sets of similar examples. Therefore, it has applications for example in exploratory data analysis.

3. Finally, *semi-supervised* systems use an annotated training set in combination with a, typically, very large unannotated training set to improve the results beyond the maximum achievable by either approach in isolation.

[1] However, for example reinforcement learning and active learning may not fit easily into this classification.

Unsupervised and Semi-supervised techniques have many applications in the field of tagging. For example, distributional similarity can be used to improve tagging accuracy for OOV words (Huang and Yates, 2009, Östling, 2013) and self-training can improve the accuracy of a tagging system (Spoustová et al., 2009, Søgaard, 2011). This thesis, however, focuses exclusively on supervised learning.

3.1 Supervised Learning

In this section, I will illustrate the key concepts and techniques in supervised machine learning using the very simple example of *linear regression*. I will explain the plain linear regression model and show how it can be fitted using training data. I will then briefly present *ridge regression* which is a *regularized* version of linear regression.

I choose linear regression as example because it is a simple model yet can be used to illustrate many important concepts in machine learning. Moreover, the model has several tractable properties such as smoothness and convexity. Additionally, it can be seen as the simplest example of a linear classifier which is a category of models encompassing conditional random fields, the hidden Markov model and average perceptron classifier presented in later chapters.

Linear Regression As a simple example, imagine a person called Jill who is a real estate agent.[2] She is interested in constructing an application, for use by prospective clients, which would give rough estimates for the selling price of a property. Jill knows that a large number of factors affect housing prices. Still, there are a few very robust predictors of price that are easy to measure. She decides to base the model on the following predictors:

1. The living area.
2. The number of rooms.
3. The number of bathrooms.
4. Size of the yard.
5. Distance of the house from the city center.
6. Age of the house.
7. Elapsed time since the last major renovation.

Jill decides to use the simplest model which seems reasonable. This model is linear regression which models the dependent variable, the house price, as a linear combination of the independent variables listed above and parameter values in $\mathbb{R}$. The linear regression model is probably not accurate. It fails in several regards. For example, increasing age of the house probably reduces the price up to a point but very old

[2] This example is inspired by the Machine learning course provided by Coursera and at the time taught by Andrew Ng.

houses can in fact be more expensive then newly built houses especially if they have been renovated lately. Although, the linear model is unlikely to be entirely accurate, Jill is happy with it because the intention is just to give a ball park estimate of the price for the prospective client.

To formalize the linear regression model, let us call the dependent variable price y and each of the independent variables living area, number of rooms and so on x_i. Given a vector $x = (x_1 \ ... \ x_n \ 1)^\top \in \mathbb{R}^{n+1}$, which combines the independent variables x_i, a bias term 1, and a parameter vector $\theta \in \mathbb{R}^{n+1}$ the linear regression model is given by Equation 3.1.[3]

$$y(x;\theta) = x^\top \theta \tag{3.1}$$

Two questions immediately arise: How to compute the price given parameters and predictors and how to compute the parameter vector θ. These questions are common for all supervised learning problems also when using other models than the linear regression model.

Inference The first question concerns *inference*, that is finding the values of the dependent variable given values for the independent variables. In the case of linear regression, the answer to this question is straightforward. To compute the price, simply perform the inner product in Equation 3.1. The question is, however, not entirely settled because one might also ask for example how close to the actual price the estimate y is likely to be. A related question would be to provide an upper and lower bound for the price so that the actual price is very likely to be inside the provided bounds. To answer these questions, one would have to model the expected error.

Inference is very easy and also efficient in the case of linear regression. With more complex models such as structured graphical models which are investigated in Chapters 4 and 6, it can however be an algorithmically and computationally challenging problem. The task is still the same: Find the y which is most likely given the input.

Training Data The second question concerns *estimation of model parameters* and it is more complex than the question of inference. First of all, Jill needs training data. In the case of house price prediction, Jill can simply use data about houses she has brokered in the past. She decides to use a training data set $\mathcal{D} = \{(x^1, y^1), ..., (x^T, y^T)\}$, where each $x^t = (x_1^t \ ... \ x_n^t \ 1)$ is a vector of independent variable values (living area, age of the house and so on) and y^t is the dependent variable value, that is the final selling price of the house. The last element 1 in x^t is the bias which is constant. Now Jill needs to make a choice. How many training examples (x^t, y^t) does she need? The common wisdom is that more data is always better. In practice, it is a good idea to start with a small training data and increase the number of training examples until the performance of the system plateaus.

Data Sparsity Whereas it is fairly easy to get a sufficient amount of training data for our example which only has a few parameters, it is vastly more difficult to accomplish with more complicated models in

[3] In reality, each of the predictors would probably be transformed to give all of them the same average and variance. Although this ii not required in theory, it tends to give a better model.

natural language processing. When there is insufficient data to estimate model parameters accurately, the data is called sparse. One central question in this thesis is how to counteract *data sparsity* in morphological tagging.

Loss Functions The objective in estimation is to find a parameter vector θ which in some sense minimizes the error of the house price predictions $y(x^t;\theta)$ when compared to the actual realized house prices y^t in the training data. The usual minimization criterion used with linear regression is the least square sum criterion given in Equation 3.2. It is minimized by a parameter vector θ which gives as small square errors $|y^t - y(x^t;\theta)|^2$ as possible.

$$\theta = \arg\min_{\theta' \in \mathbb{R}^n} \sum_{x^t \in \mathcal{D}} |y^t - y(x^t;\theta')|^2 \tag{3.2}$$

The square sum is an example of a *loss function* (also called the objective function). A loss function assigns a non-negative real loss for each parameter vector. Using the concept of loss function, the objective of estimation can be reformulated: Find the parameter vector θ that minimizes the average loss over the training data.

Iterative Estimation In the case of linear regression model, there is an exact solution for the optimization of parameter vector θ.[4] This does not hold for more complex models. Moreover, the exact solution might often not be the one that is desired because it does not necessarily generalize well to unseen examples. This is called *overfitting*. Fortunately, the loss function can be modified to counteract overfitting. After the modification, the parameter optimization problem might, however, no longer have a closed form solution.

Because the loss of the training data is a function of the model parameters, one can apply mathematical analysis for finding optimal parameter values. These methods include for example Newton's method which is an iterative procedure that can be used to find the zeros of a differentiable function or local extrema of a twice differentiable function. Approximations of Newton's method, so called Quasi-Newton methods (Liu and Nocedal, 1989), have also been developed because Newton's method requires evaluation and inversion of the Hessian matrix of a function. This is a very costly operation for functions where the domain has high dimension. Quasi-Newton methods use approximations of the inverse Hessian.

A simpler method called gradient descent can be applied to functions that are once differentiable. In general, gradient descent converges toward the optimum more slowly than Newton's method, however, the computation of one step of the iterative process is much faster when using gradient descent. Therefore, it may be faster in practice.

All gradient based methods rely on differentiability of the loss function.[5] For the models used in this thesis, differentiability holds. Gradient based methods work in the following general manner. Let $\mathcal{L}_\mathcal{D} : \mathbb{R}^n \to \mathbb{R}$ be the loss of the training data $\mathcal{D}$.

[4] The solution is given by $\theta = X^+ Y$ where $X^+ = (X^\top X)^{-1} X^\top$ is the More-Penrose pseudo-inverse of X.
[5] At least, differentiability almost everywhere.

1. Start at a random point θ_0 in the parameter space.
2. Determine the direction of steepest descent of the loss function. This is the negative gradient $-\nabla\mathcal{L}_{\mathcal{D}}(\theta_t)$ at point θ_t.
3. Determine a suitable step size $\alpha_t \in \mathbb{R}_+$.
4. Take a step of length α_t in direction v_t to get to the next point in the parameter space θ_{t+1}, that is $\theta_{t+1} = \theta_t - \alpha_t \nabla\mathcal{L}_{\mathcal{D}}(\theta_t)$.
5. If the difference in loss $|\mathcal{L}_{\mathcal{D}}(\theta_{t+1}) - \mathcal{L}_{\mathcal{D}}(\theta_t)|$ is smaller than a threshold ρ, set $\theta = \theta_{t+1}$. Otherwise, set $\theta_t = \theta_{t+1}$ and return to line 2.

The main difference between first and second order methods is the computation of the step size α_t. Second order methods can take longer steps when the loss is plateauing. Thus they typically take fewer steps in total. In first order methods such as gradient descent, α_t can be constant, a decreasing function of t or can also be determined by a line search in the direction of $-\nabla\mathcal{L}_{\mathcal{D}}(\theta_t)$. For example $\alpha_t = t^{-1}$ may work.

As the meta-algorithm above suggests, gradient based optimization algorithms are local in the sense that they always move in the direction of steepest descent of the loss function, that is toward a local optimum. Therefore, they will in general not find the global optimum of the loss function. By choosing a *convex* loss function, which has maximally one local, and thus also, global optimum it is possible to avoid getting stuck at local optima.

Convexity is, however, not enough to guarantee convergence to a global optimum. First of all, a global optimum might not exist.[6] Moreover, convergence may be too slow. This can leads to premature termination of the training procedure. This is specifically a problem for first order methods.

Online Estimation The optimization methods discussed up to this point have been so called *batch methods*. The derivatives of the loss function is computed over the entire training data and parameters are updated accordingly. Batch methods can be slow and subsequent training when new training examples become available is computationally intensive. *Online algorithms* are an alternative to batch methods, where the loss is instead computed for a randomly chosen training example and the parameters are then updated accordingly. In practice, online methods can give fast convergence. Moreover, re-training is relatively efficient when new training examples become available.

Stochastic gradient descent is a well known online estimation algorithm. The algorithm processes one random training example at a time. It uses the gradient $\nabla L_{\mathcal{D}[i]}(\theta)$ of the loss for this training example $\mathcal{D}[i]$ to approximate the gradient $\nabla L_{\mathcal{D}}(\theta)$ over the entire training data $\mathcal{D}$. It is identical to the ordinary batch gradient descent except that it is an online estimation algorithm. In practice, SGD converges substantially faster than regular gradient descent (Vishwanathan et al., 2006) because the evaluation of the approximate gradient is very fast compared to evaluation of the gradient over the entire training data.

[6] This can happen if the domain of the loss function is not compact. Unfortunately, it usually is not.

Regularization Due to the problem of over-fitting, a family of heuristic techniques called *regularization* is often employed. They aim to transform the original problem in a way which will penalize both deviance from the gold standard and "complexity" of the solution θ. Regularization can be seen to convey the same idea as Occam's Maxim which states that a simpler explanation for a phenomenon should be preferred when compared to a more complex explanation yielding equivalent results. Of course, this does not explain what is meant by a "complex" parameter vector θ.

To illustrate simple and complex parameter vectors, examine a case of linear regression where the dependent variable y and the predictors x_i have mean 0 and variance 1 in the training data. This may seem restrictive but in fact any linear regression problem can easily be transformed into this form by applying an affine transformation $z \mapsto az - b$. When doing inference, this affine transformation can simply be reversed by applying $z \mapsto a^{-1}(z + b)$. The simplest parameter vector θ is clearly the zero vector $\theta = (0...0)$. It corresponds to the hypothesis that the predictors x_i have no effect on the dependent variable y. According to this hypothesis, the prediction for the house price is identically zero.

The zero solution to a linear regression problem is simple but also completely biased. Because we are assuming that the independent variables x_i explain the dependent variable y, a model that completely disregards them is unlikely to give a good fit to the training data. By introducing a regularization term into the loss function, we can however encourage simple solutions while at the same time also preferring solutions that give a good fit. There are several ways to accomplish this but the most commonly used are so called L_1 and L_2 regularization.[7] These are general regularization methods that are employed in many models in machine learning.

The L_1 regularized loss function for linear regression is given in Equation 3.3. L_1 regularization, also called LASSO regularization Tibshirani (1996), enforces solutions where many of the parameter values are 0 (such parameter vectors are called sparse). It is suitable in the situation where the model is overspecified, that is, many of the predictors might not be necessary for good prediction. The L_1 regularized linear regression loss is given by Equation 3.3.

$$\theta = \underset{\theta' \in \mathbb{R}^n}{\arg\min} \sum_{x_t \in \mathcal{D}} |y^t - y(x^t; \theta')|^2 + \lambda \sum_i |\theta_i| \tag{3.3}$$

The L_2 regularized loss function is given in 3.4. L_2 regularization is also called Tikhonov regularization. In contrast to L_1 regularization, it directly prefers solutions with small norm. A linear regression model with Tikhonov regularization is called a ridge regression model.

$$\theta = \underset{\theta' \in \mathbb{R}^n}{\arg\min} \sum_{x_t \in \mathcal{D}} |y_t - y(x_t; \theta')|^2 + \lambda \|\theta\|^2 = \underset{\theta' \in \mathbb{R}^n}{\arg\min} \sum_{x_t \in \mathcal{D}} |y_t - y(x_t; \theta')|^2 + \lambda \sum_i |\theta[i]|^2 \tag{3.4}$$

The coefficient $\lambda \in \mathbb{R}^+$ is called the *regularizer*. The regularizer determines the degree to which

[7] Another approach to counteracting overfitting is provided by Bayesian statistics where the parameter vector θ is drawn from a prior distribution. In practice, Bayesian methods and regularization are often equivalent.

model fit and simplicity affect the loss. A higher λ will increase the loss for complex models more than a lower one. When λ increases, the optimal parameter vector θ approaches the zero vector and when it decreases θ approaches the parameters that fit the training data as closely as possible. This is called under-fitting.

Hyper-parameters The regularizer is a so called *hyper-parameter* of the regularized liner regression model. It is easy to see that increasing λ will automatically increase the loss. Therefore, there is no direct way to estimate its correct magnitude simply using the training data. Instead *held-out data* can be used. Held-out data is labeled data that is not used directly for estimating model parameters. If the model over-fits the training data, that is generalizes poorly to unseen examples, the held-out data will have a high loss. However, it will also have a high loss if the model under-fits, that is, performs poorly on all data. Held-out data can therefore be used to find an optimal values for the regularizer λ. Often, one tries several potential values and chooses the one that minimizes the loss of the held-out data. Usually, one uses the non-regularized loss function for the held-out data.

3.2 Machine Learning Experiments

In this thesis and in the associated articles, I present several experiments in morphological tagging. The experiments are conduct on labeled data and follow a set pattern.

1. **Data Splits** The labeled data set is divided into three non-overlapping parts: (1) a training set used for estimating model parameters (2) a development set used for setting hyper parameters and performing preliminary experiments during development and (3) a test sets which is used to perform the final evaluation of the model.

2. **Feature Engineering** Using the training set and development set, a number of features are tested and depending on tagging errors in the development data, new features may be added.

3. **Tuning** The model hyper-parameters are set using development data.

4. **Training** When model parameters and hyper-parameters are set, the final model is trained on the combined training and development data. Training time is measured at this point.

5. **Evaluation** The performance of the model is measured on the test data in order to derive an estimate of tagging accuracy and tagging speed.

A crucial component of machine learning experiment is the *baseline*. For example, when investigating the impact of a set of features on tagging accuracy, the baseline will be the model which does not include those features. In Publication **VI**, which investigates the tagging accuracy, tagging speed or training speed of the FinnPos toolkit, other established tagger tool-kits are used as baseline.

When comparing tagging accuracy of two taggers, we compare their accuracies on the test set. However, this is only an estimate of the true tagging accuracies of the systems. When the difference in performance between the systems is small, it is therefore not possible to say with great certainty which system will perform better on new data. In this situation, it is helpful to know about the variance of the accuracy.

The variance is a measure of the stability of the difference in accuracies between tagging systems. It can be estimated using random samples of the test data. If one system consistently performs better than the other one on random samples of the test data, it is more likely to performs better on some unseen sample. In contrast, when the performance of one system is better on some samples and worse on others, it is less certain that it would perform better on unseen data even though it performs better on average in the entire test set.

Using statistical significance testing, the above comparison can be formalized. In the papers included in this thesis, the 2-sided Wilcoxon signed-rank test (Wilcoxon, 1945). In contrast to the often used t-test, the Wilcoxon test does not assume that the measurements are drawn from a Gaussian distribution. A 2-sided test (instead of a 1-sided test) is used because it cannot be known which of the systems actually has the higher accuracy although we know that one of the systems performs better on the test set.[8]

[8]This was suggested by one of the reviewers of Publication **VI**.

Chapter 4

Hidden Markov Models

This chapter introduces hidden Markov models (HMM), which are a widely used model for POS tagging and morphological tagging. Extensions to the HMM are further investigated in the next chapter and Publications **I**, **II** and **III**.

4.1 Example

I will illustrate Hidden Markov Models using an example. Imagine a person called Jill who is hospitalized and occupies a windowless room. The only way for her to know what is happening in the outside world is to observe a nurse who passes her room daily.[1]

Suppose, Jill is interested in weather phenomena and she decides to pass time by guessing if it is raining outside. She bases her guesses on whether or not the nurse is carrying an umbrella. In other words, she predicts an *unobserved variable*, the weather, based on an *observed variable*, the nurse's umbrella.

There are several probabilistic models Jill might use. The simplest useful model assigns probability 1 to the event of rain, if the nurse carries an umbrella, and assign it the probability 0 otherwise. This simplistic model would certainly give the correct prediction most of the time, but Jill believes that she can do better.

Jill knows that people often carry an umbrella when it is raining. She also knows that they rarely carry one when the weather is clear. However, people sometimes do forget their umbrella on rainy days, perhaps because they are in a hurry. Moreover, people sometimes carry an umbrella even when it is not raining. For example the weather might be murky and they might anticipate rain. Therefore, Jill decides to reserve some probability, say 0.2, for the event that the nurse is carrying an umbrella when there is no

[1] To make things simple, imagine the nurse works every day.

rain. She reserves an equal probability for the event that the nurse arrives at work without an umbrella although it is in fact raining.

Without additional information, this more complicated model will give exactly the same MAP predictions as the simplistic one. Knowledge of meteorology, however, also factors in. Let us suppose Jill is a weather enthusiast and she knows that the probability of rain is 0.25 a priori, making the probability of clear weather 0.75. She also knows that the probability of rain increases markedly on days following rainy days at which time it is 0.7. Similarly, the probability of clear weather increases to 0.9 if the weather was clear on the previous day. Figure 4.1 summarizes these probabilities.[2]

ι	
CLEAR	0.75
RAIN	0.25

T	CLEAR	RAIN
CLEAR	0.9	0.1
RAIN	0.3	0.7

E		
CLEAR	0.8	0.2
RAIN	0.2	0.8

Figure 4.1: The probability distributions which define the HMM in the weather forecast example. ι specifies the initial probability of CLEAR and RAIN. T shows the transition distributions, which specify the probability of CLEAR and RAIN given the weather on the previous day. Finally, E shows the emission distributions, which specify the probabilities of seeing an umbrella depending on the weather.

Let us assume that Jill observes the nurse for one week. She sees the nurse carry an umbrella on all days except Tuesday. The MAP prediction given by the simplistic model is that Tuesday is clear and all other days are rainy. The more complex model will, however, give a different MAP prediction: the probability is maximized by assuming that all days are rainy. Under the more complex model, it is simply more likely that the nurse forgot to bring an umbrella on Tuesday.

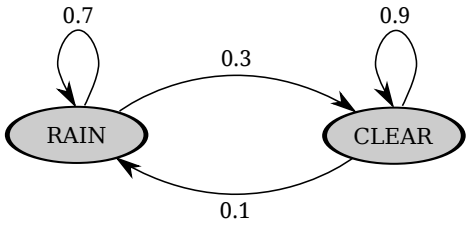


Figure 4.2: A visual representation of the HMM in Figure 4.1.

The model Jill is using for weather prediction is called a Hidden Markov Model. It can be used to make predictions about a series of events based on indirect observations.

The HMM is commonly visualized as a directed graph. Each hidden state, for example RAIN and CLEAR, represents a vertex in the graph. Transitions from one hidden state to another are represented by arrows labeled with probabilities. Figure 4.2 shows a graph representing the transition structure of the

[2]Since the author of this thesis has very little knowledge about meteorology, these probabilities are likely to be nonsense. The overall probability of rain and clear weather is, however, chosen to be the steady state of the Markov chain determined by the probabilities of transitioning between states. Consistency is therefore maintained.

HMM outlined in Figure 4.1.

4.2 Formal Definition

Abstracting from the example above, an HMM is a probabilistic model that generates sequences of state observation pairs. At each step t in the generation process, the model generates an observation by sampling the *emission distribution* ε_{y_t} of the current state y_t. It will then generate a successor state y_{t+1} by sampling the *transition distribution* τ_{y_t} of state y_t. The first hidden state y_1 is sampled from the *initial distribution* ι of the HMM.

Since the succession of days is infinite for all practical purposes, there was no need to consider termination in the example presented in Figure 4.2. Nevertheless, many processes, such as sentences, do have finite duration. Therefore, a special *final state* f is required. When the process arrives at the final state, it stops: no observations or successor states are generated.

Following Rabiner (1990)[3], I formally define a *discrete* HMM as a structure $(Y,\ X,\ i,\ T,\ E,\ F)$ where:

1. Y is the set of hidden states ($Y = \{\mathrm{CLEAR}, \mathrm{RAIN}\}$ in the example in Figure 4.1).
2. X is the set of emissions, also called observations ($X = \{$  ,  $\}$ in the example in Figure 4.1).
3. $\iota : Y \to \mathbb{R}$ is the initial state distribution, that is the probability distribution determining the initial state of an HMM process (array ι in Figure 4.1).
4. T is the collection of transition distributions, $\tau_y : Y \to \mathbb{R}$, that determine the probability of transitioning from a state y to each state $y' \in Y$ (array T in Figure 4.1).
5. E is the collection of emission distributions $\varepsilon_y : X \to \mathbb{R}$, which determine the probability of observing each emission $o \in X$ in state $y \in Y$ (array E in Figure 4.1).
6. $f \in Y$ is the final state. The state f emits no observations and there are no transitions from f.

Figure 4.3 gives a visualization of the HMM in Figure 4.1 with an added final state. Because the progression of days is infinite for all practical purposes, the probability of transitioning to the final state f in example 4.3 is 0 regardless of the current state. Hence, the probability of any single sequence of states and emissions is 0. The probability of an initial segment of a state sequence may, however, be non-zero.[4]

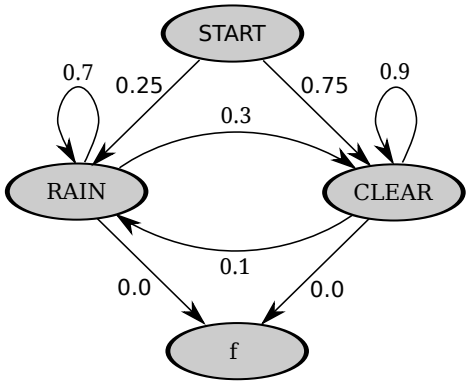
An HMM models a number of useful quantities:

1. The *joint probability* $p(x, y; \theta)$ of a observation sequence x and state sequence y. This is the probability that an HMM with parameters θ will generate the state sequence y and generate the observation x_t in every state y_t.

[3] The definition of HMMs in this thesis differs slightly from Rabiner (1990) since I utilize final states.

[4] The probability of an initial segment up to position t can be computed using the forward algorithm, which is presented in Section 4.3.

Figure 4.3: The HMM presented in Figure 4.1 with an unaccessible final state.



2. The *marginal probability* $p(x;\theta)$ of an observation sequence x. This is the overall probability that the observation sequence generated by an HMM is x.

3. The *conditional probability* $p(y \mid x;\theta)$ of a state sequence y given an observation sequence x. That is, how likely it is that the model passes through the states in y when emitting the observations in x in order.

4. The *marginal probability* $p(z,t,\ x;\theta)$ *of state* z *at position* t when emitting the observation sequence x. That is, the probability of emitting observation sequence x under the single constraint that the state at position t has to be z.

To formally define these probabilities, let $\theta = \{\iota,\ T,\ E\}$ be the parameters of some HMM with observation set X and hidden state set Y, $x \in X^T$ be a sequence of observations and $y \in Y^{T+1}$ a sequence of hidden states. The last state y_{T+1} in y has to be the final state f. Then the joint probability $p(x,y;\theta)$ of x and y given θ is defined by Equation (4.1).

$$p(x,y;\theta) = p(y;\theta) \cdot p(x \mid y;\theta) = \left(\iota(y_1) \cdot \prod_{t=1}^{T} \tau_{y_t}(y_{t+1})\right) \cdot \prod_{t=1}^{T} \varepsilon_{y_t}(x_t) \tag{4.1}$$

Equation (4.1) is a product of two factors: the probability of the hidden state sequence y, determined by the initial and transition probabilities, and the probability of the emissions x_t given hidden states y_t determined by the emission probabilities.

When the HMM model is used as a morphological tagger, the emissions are word forms and the hidden states are morphological labels. This allows for capturing simple grammatical dependencies between adjacent morphological labels. For example, in English, a determiner is often followed by an adjective, participle, noun or adverb, but rarely followed by an active verb form or another determiner. The HMM

can, therefore, be seen as a simple probabilistic model of grammar where the grammar rules only concern co-occurrences of words and labels as well as co-occurrences of adjacent labels. As demonstrated by the success of the HMM model in POS tagging, this simple model can be surprisingly effective.

In the standard HMM, every hidden states in Y has a probability for emitting any given observation (of course, the emission probability for a particular observation can be zero in some states). Therefore, several state sequence $y \in Y^{T+1}$ can be generate the same sequence of observations $x \in X^T$. The marginal probability $p(x;\theta)$ of an observation sequence x can be found by summing over all state sequences that could have generated x. It is defined by Equation (4.2).

$$p(x;\theta) = \sum_{y \in Y^{T+1},\ y_{T+1}=f} p(x,y;\theta) \tag{4.2}$$

Possibly the most important probability associated to the HMM is the conditional probability $p(y \mid x;\theta)$ of state sequence y given observations x. This is an important quantity because maximizing $p(y \mid x;\theta)$ with regard to y will give the MAP assignment of observation sequence x. It is defined by Equation (4.3).

$$p(y \mid x;\theta) = \frac{p(x,y;\theta)}{p(x;\theta)} \tag{4.3}$$

It is noteworthy, that $p(y \mid x;\theta) \propto p(x,y;\theta)$ because the marginal probability $p(x;\theta)$ is independent of y. Therefore, y maximizes $p(y \mid x;\theta)$ if and only if, it maximizes $p(x,y;\theta)$. This facilitates inference because the MAP assignment for the hidden states can be computed without computing the marginal probability $p(x;\theta)$.

Finally, the posterior marginal probability of state z at position t given the observation sequence x is computed by summing, or marginalizing, over all state sequence y, where $y_t = z$. It is defined by Equation (4.4)

$$p(z,t,\ x;\theta) = \sum_{y' \in Y^{T+1},\ y'_t=z,\ y'_{T+1}=f} p(x,y';\theta) \tag{4.4}$$

4.3 Inference

Informally, inference in HMMs refers to finding a maximally probable sequence of hidden states y that might have emitted the observation x. As Rabiner (1990) points out, this statement is not strong enough to suggest an algorithm.

Maximally probable is an ambiguous term when dealing with structured models. It could be taken to mean at least two distinct things. The MAP assignment y_{MAP} of the hidden state sequence is the most

probable joint assignment of states defined by Equation (4.5) and depicted in Figure 4.4a.

$$y_{MAP} = \arg\max_{y \in Y^T} p(y \mid x; \theta) \tag{4.5}$$

Another possible definition would be the *maximum marginal* (MM) assignment. It chooses the most probable hidden state for each word considering all possible assignments of states for the remaining words. The MM assignment y_{MM} is defined by Equation (4.6). Figure 4.4c shows the paths whose probabilities are summed in order to compute the marginal for one position and state.

$$y_{MM} = \arg\max_{y \in Y^T} \prod_{t=1}^{T} p(y_t, t \mid x; \theta) \tag{4.6}$$

As Merialdo (1994) and many others have noted, the MAP and MM assignments maximize different objectives. The MM assignment maximizes the accuracy of correct states per observations whereas the MAP assignment maximizes the number of completely correct state sequences. Both objectives are important from the point of view of POS tagging in a theoretical sense. However, they are often quite correlated and, at least in POS tagging, it does not matter in practice which of the criteria is used (Merialdo, 1994). Most systems, for example Church (1988), Brants (2000), Halácsy et al. (2007), have chosen to use MAP inference, possibly because it is easier to implement and faster in practice.

Although, MM inference is more rarely used with HMMs, computing the marginals is important both in unsupervised estimation of HMMs and discriminative estimation of sequence models. Therefore, an efficient algorithm for MM inference, the *forward-backward algorithm*, is presented below.

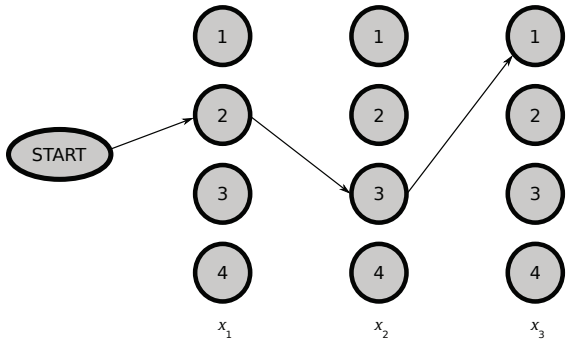
There are a number of strongly related algorithms for both exact MAP and MM inference. The work presented in this thesis, uses the Viterbi algorithm for MAP inference and the forward-backward algorithm for MM inference (Rabiner, 1990). *Belief propagation*, introduced by Pearl (1982), computes the MM assignment and can be modified to compute the MAP assignment as well. For sequence models, such as the HMM where hidden states form a directed sequence, belief propagation is very similar to the forward-backward algorithm. It can, however, be extended to cyclic graphs (Weiss, 2000) unlike the Viterbi algorithm.

Since cyclic models fall beyond the scope of this thesis and both the Viterbi and forward-backward algorithms are amenable to well known optimizations, which are of great practical importance, I will not discuss belief propagation further. Koller and Friedman (2009) gives a nice treatment of belief propagation and graphical models at large.

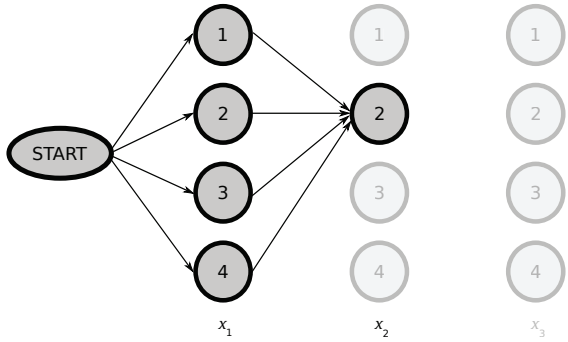
Before introducing the Viterbi and forward-backward algorithm, it is necessary to investigate the forward algorithm, which is used to compute the marginal probability of an observation and also as part of the forward-backward algorithm. The forward algorithm and Viterbi algorithm are closely related.

The Forward Algorithm Equations (4.5) and (4.6) reveal, that both MAP and MM inference require knowledge of the entire observation x. In the weather prediction example, observations are, however,

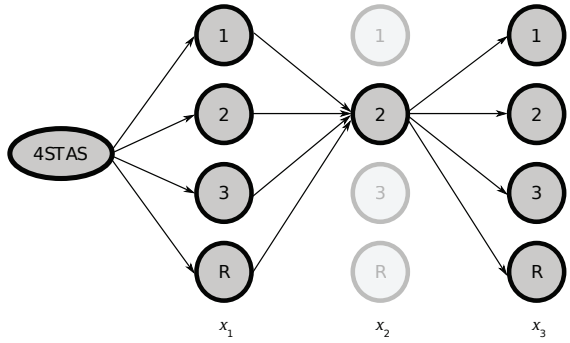
Figure 4.4: A visualization of subsets of paths in a trellis.



(a) Trellis and path.



(b) Forward path prefixes.



(c) Marginal paths.

always infinite. What kind of inference is possible in this case?

Even when we only know a prefix $x[1:t]$ (of length t) of the entire observation x, we can still compute the *belief state* (Boyen and Koller, 1998) of the HMM given the prefix. The belief state is in fact not a single state, but rather a distribution over the set of hidden states Y. It tells us how likely we are to be in state z at time t, when we have emitted the prefix $x[1:t]$.

To compute the belief state at position t, we first need to compute the *forward probabilities* for each state $z \in Y$. The forward probability $\mathrm{fw}_{t,z}(x)$ of state z at position t is the probability of emitting prefix $(x_1, ..., x_t)$ and ending up in state $z \in Y$. For example, given an infinite observation (☂, ☂, 🌂, ...), the forward probability $\mathrm{fw}_{3,\mathrm{RAIN}}$ is the probability that the third day is rainy, when the nurse carried an umbrella on the first and second days, but did not carry one on the third day.

I am going to make a technical but useful definition. The *prefix probability* of observation sequence $x = (x_1, \ ..., \ x_T)$ and state sequence $y = (y_1, \ ..., \ y_t)$ at position, where $t < T + 1$ is given by Equation (4.7).

$$p(x, \ y; \theta) = \left(\iota(y_1) \cdot \left(\prod_{u=1}^{t-1} \tau_{y_u}(y_{u+1}) \right) \cdot \prod_{u=1}^{t} \varepsilon_{y_u}(x_u) \right), \ t \leq T \tag{4.7}$$

When $t = T$, this is almost the same as the joint probability of x and y, but the final transition is missing.

Conceptually, the forward probability is computed by summing over the probabilities of all path prefixes up to position t, where the state at position t is z, see Figure 4.4b. Formally, the forward probability is defined by Equation (4.8).

$$\mathrm{fw}_{t,z} = \sum_{y \in Y^t, \ y_t = z} p(x, \ y; \theta) \tag{4.8}$$

Comparing Equations (4.8) and (4.1) shows that the forward probability in a sense represents the probability of a prefix of observation x.

The belief state and posterior marginal distribution may seem similar. They are, however, distinct distributions because the belief state disregards all information about observation x after position t. In contrast, the marginal distribution encompasses information about the entire observation. For example the marginal probability of RAIN at position 3 is likely to depend strongly on whether or not Jill observes the nurse carry an umbrella on the fourth day. However, this will have no impact on the belief state.

Figure 4.5 demonstrates a naive approach to computing the forward probabilities. Simply list all relevant state sequences, compute the probability of each sequence and sum the probabilities. Unfortunately, the naive approach fails for large t because the number of distinct state sequences depends on the sequence length in an exponential manner.

The complexity of the naive algorithm is $|Y|^t$, which is infeasible. For example, $f_{20,\mathrm{RAIN}}(x)$ requires us to sum approximately 20 million probabilities and $f_{30,\mathrm{RAIN}}(x)$ entails summation of approximately 540 million probabilities. Since observation sequences in domains such as natural language processing frequently reach lengths of 100, a more efficient approach is required.

The belief state can be computed in linear time with regard to t and quadratic time with regard to $|Y|$ using the *forward algorithm* (Rabiner, 1990), which is in fact simply a recursive application of the right

Figure 4.5: A naive approach to computing forward probabilities.

y_1	y_2	y_3	p
CLEAR	CLEAR	CLEAR	$(0.75 \cdot 0.8 \cdot 0.9 \cdot 0.2) \cdot 0.9 \cdot 0.8 \approx 0.078$
RAIN	CLEAR	CLEAR	$(0.25 \cdot 0.2 \cdot 0.3 \cdot 0.2) \cdot 0.9 \cdot 0.8 \approx 0.002$
CLEAR	RAIN	CLEAR	$(0.75 \cdot 0.8 \cdot 0.1 \cdot 0.8) \cdot 0.3 \cdot 0.8 \approx 0.012$
RAIN	RAIN	CLEAR	$(0.25 \cdot 0.2 \cdot 0.7 \cdot 0.8) \cdot 0.3 \cdot 0.8 \approx 0.007$
			
			≈ 0.098

distributive rule of algebra

$$a_1 \cdot b + ... + a_n \cdot b = (a_1 + ... + a_n) \cdot b$$

for real numbers a_1 up to a_n and b.

Instead of computing the probability separately for each path, the forward probabilities for longer paths are computed incrementally using the forward probabilities of shorter paths. Examine Figure 4.5. By grouping rows one and two, as well as three and four into pairs, it is easy to see that

$$\mathrm{fw}_{3,\mathrm{CLEAR}} = (\mathrm{fw}_{2,\mathrm{RAIN}} \cdot \tau_{\mathrm{RAIN}}(\mathrm{CLEAR}) + \mathrm{fw}_{2,\mathrm{CLEAR}} \cdot \tau_{\mathrm{CLEAR}}(\mathrm{CLEAR})) \cdot \varepsilon_{\mathrm{CLEAR}}(\text{  })$$

Generalizing, we get the recursion in Equation (4.9).

$$\mathrm{fw}_{t,z} = \begin{cases} \iota(z) \cdot \varepsilon_z(x_1) & , t = 1 \\ \left(\sum_{z' \in Y} \mathrm{fw}_{t-1,z'} \cdot \tau_{z'}(z) \right) \cdot \varepsilon_z(x_t) & , 1 < t \leq T \\ \sum_{z' \in Y} \mathrm{fw}_{T,z'} \cdot \tau_{z'}(f) & , t = T+1,\ z = f. \end{cases} \tag{4.9}$$

The remaining forward probabilities $\mathrm{fw}_{T+1,z}$, where $z \neq f$ are defined to be 0.

The forward probability $f_{T+1,f} = p(x; \theta)$. In fact one of the principal applications for the forward algorithm is computing the marginal probability of an observation. The other central application is in the forward-backward algorithm, which computes the state marginals.

The forward algorithm is outlined in Algorithm 4.1. Assuming that accessing the data structures `x`, `i_prob`, `e_prob`, `tr_prob` and `trellis` is constant time, the complexity of the algorithm is dominated by the three nested loops on lines 27–37. This shows that the complexity of the forward algorithm is linear with regard to the length of the sequence and quadratic with regard to the size of the hidden state set.

Although, the forward algorithm depends linearly on the observation length, its quadratic dependence

on the size of the hidden state set is problematic from the perspective of morphological disambiguation of morphologically complex languages, where the size of the hidden state set is measured in the hundreds or thousands for regular HMMs. When using second order HMMs presented below, the state set can grow to tens of thousands or millions, which can slow down systems to a degree that makes them infeasible in practice. I will present partial solutions to these problems below.

The Viterbi Algorithm Whereas the forward algorithm incrementally computes the marginal probability of an observation x, the Viterbi algorithm incrementally computes the MAP assignment for observation x.

A naive approach to finding the MAP assignment is to list all the hidden state paths, compute their probabilities and pick the one with the highest probability. Similarly as for the forward algorithm, the complexity of this approach is exponential with regard to the length of observation x.

Just as in the case of forward probabilities, the MAP assignment of hidden states for a prefix of the observation x can be computed incrementally. Formally, the MAP assignment for a prefix $x[1 : t]$ is defined by equation (4.10) utilizing the joint prefix probability of x and a state sequence y of length t. Intuitively, it is the sequence of hidden states $y_{t,z}$ which maximizes the joint probability and ends at state z.

$$y_{t,z} = \underset{y \in Y^t,\ y_t = z}{\arg\max}\ p(x,\ y; \theta) \tag{4.10}$$

Comparing this equation with the definition of the forward probability $f_{t,z}$ in Equation 4.8, we can see that the only difference is that the sum has been changed to $\arg\max$.

I will now show that the MAP prefix $y_{t,z}$ can be computed incrementally in a similar fashion as the forward probability $f_{t,z}$. Suppose that $y_{t+1,z'} = (y_1,\ ...,\ y_t = z,\ y_{t+1} = z')$. I will show that $y_{t+1,z'}[1 : t] = y_{t,z}$. Let y' be the concatenation of $y_{t,z}$ and z'. If $y_{t+1,z'}[1 : t] \neq y_{t,z}$, then

$$\begin{aligned} p(x,\ y_{t+1,z'}; \theta) &= p(x,\ y_{t+1,z'}[1 : t]; \theta) \cdot \tau_z(z') \cdot \varepsilon_{z'}(x_{t+1}) \\ &< p(x,\ y_{t,z}; \theta) \cdot \tau_z(z') \cdot \varepsilon_{z'}(x_{t+1}) \\ &= p(x, y'; \theta) \end{aligned}$$

This contradicts the definition in Equation (4.10).[5]

We now get Equation (4.11), which gives us a recursion. The implementation of the Viterbi algorithm is identical to the implementation of the forward algorithm except that sums are replaced by maximization. Consequently, the time complexity of the algorithm is the same. It is linear with regard to sentence length

[5] As long as we suppose that there is exactly one MAP prefix.

Algorithm 4.1: The forward algorithm in Python 3.

```
def forward(x, i_prob, e_prob, tr_prob):
    """
        x       - The observation as a list.
        i_prob  - Initial state distribution.
        e_prob  - Emission distributions.
        tr_prob - Transition distributions.

        Return the trellis of forward probabilities.
    """

    assert(not x.empty())

    trellis = {}

    # Indexing in python starts at 0.
    x_1 = x[0]
    T   = len(x) + 1

    # Set final state F. States are consecutive integers
    # in the range [0, F].
    F = len(i_prob) - 1

    # Initialize first trellis column.
    for z in range(F):
        trellis[(1,z)] = i_prob[z] * e_prob[z][x_1]

    # Set all except the final column.
    for t in range(2, T):
        trellis[(t, z)] = 0

        x_t = x[t - 1]

        for z in range(F):
            for s in range(F):
                trellis[(t, z)] = trellis[(t - 1, s)] * tr_prob[s][z]

            trellis[(t, z)] *= em_prob[z][x_t]

    # Set the last column.
    for z in range(s_count):
        trellis[(T + 1, z)] = trellis[(T, z)] * tr_prob[z][F]

    return trellis
```

and quadratic with regard to the size of the state set.

$$y_{t+1,z} = \arg\max_{z \in Y} \begin{cases} \iota(z) \cdot \varepsilon_z(x_1) & , t = 1 \\ y_{t-1,z'} \cdot \tau_{z'}(z) \cdot \varepsilon_z(x_t) & , 1 < t \leq T \\ y_{T,z'} \cdot \tau_{z'}(f) & , t = T+1,\ z = f. \end{cases} \tag{4.11}$$

Beam Search As seen in the previous section, the complexity of the Viterbi algorithm depends on the square of the size of the hidden state set. This can be problematic when the set of hidden states is large, for example when the states represent morphological labels in a very large label set or when they represent combinations of labels. When tagging, a morphologically complex language, the state set may easily encompass hundreds or even thousands of states.

Beam search is is a heuristic which prunes the search space explored by the Viterbi algorithm based on the following observation: in many practical applications, the number of hidden states, which emit a given observation with appreciable probability, is small. This is true even when the total number of hidden states is very large. For example, when the states represent morphological labels, a given word such as "dog" can usually only be emitted by a couple of states (maybe Noun and Verb in this case).

When the Viterbi algorithm maximizes (4.11) for $y_{t+1,z}$, a large number of histories $y_{t,z}$ can, therefore, be ignored.

Often a constant number, the *beam width*, of potential histories are considered in the maximization. The complexity of the Viterbi algorithm with beam search is $o(|x||b||\mathcal{Y}|)$, where $|x|$ is the input length, $|b|$ the beam width and $|\mathcal{Y}|$ the size of the state set. [6]

In addition to histories, the possible hidden states for output can also be filtered. The simplest method in to use a so called tag dictionary. These techniques are described in Section 4.6.

The Forward-Backward Algorithm The Viterbi algorithm computes the MAP assignment for the hidden states efficiently. For efficiently computing the marginal probability for a every state and position (see Figure 4.4c), the forward-backward algorithm is required.

Intuitively, the probability that a state sequence y has state $z \in Y$ at position t, that is the probability that $y_t = z$, is the product of the probabilities that the prefix $y[1:t]$ ends up at state z and the probability that the suffix $y[t:T]$ originates at z.

The name forward-backward algorithm stems from the fact, that the algorithm essentially consists of one pass of the forward algorithm, which computes prefix probabilities, and another pass of the forward algorithm starting at the end of the sentence and moving towards the beginning which computes suffix probabilities. Finally, the forward and suffix probabilities are combined to give the marginal probability of all paths where the state at position t is z. These passes are called the forward and backward pass, respectively.

[6] Sequential decoding, an approximate inference algorithm, which was used for decoding before the Viterbi algorithm was in common use (Forney, 2005) is very similar to beam search. Indeed, it could be said that Viterbi invented an exact inference algorithm, which is once more broken by beam search.

Formally, the suffix probabilities computed by the backward pass are defined by equation (4.12).

Since a backward pass of the forward algorithm carries the same complexity as the forward pass, we can see that the complexity of the forward-backward algorithm is the same as the complexity of the forward algorithm, however, there is a constant factor of two compared to the forward algorithm.

$$b_{t,z} = \begin{cases} \left(\sum_{z' \in Y} t_z(z') \cdot b_{t+1,z'} \right) \cdot e_z(x_{t+1}) & , 1 < t < T \\ t_z(f) & , t = T+1,\ z = f. \end{cases} \tag{4.12}$$

4.4 Estimation

HMMs can be trained in different ways depending on the quality of the available data, but also on the task at hand. The classical setting presented by Rabiner (1990) is nearly completely unsupervised: the HMM is trained exclusively from observations. Some supervision is nevertheless usually required to determine the number of hidden states. Additionally priors on the emission and transitions distributions may be required to avoid undesirably even distributions (Cutting et al., 1992, Johnson, 2007).

The unsupervised training setting has two important and interrelated applications:

1. Modeling a complex stochastic process from limited data. Here the HMM can be contrasted to a Markov chain (Manning and Schütze, 1999, 318–320), where each emission can occur in a unique state leading to a higher degree of data sparsity and inability to model under-lying structure.

2. Uncovering structure in data, for example part-of-speech induction (Johnson, 2007).

The classical method for unsupervised Maximum likelihood estimation of HMMs is the *Baum-Welch algorithm* (Rabiner, 1990), which is an instance of the *expectation maximization algorithm* (EM) (Dempster et al., 1977) for HMMs.

In morphological tagging, the supervised training scenario is normally used. Supervised training consists of annotating a text corpus with POS labels and estimating the emission and transition probabilities from the annotated data.

Straight-forward counting is sufficient to get the ML estimates for the transition and emission distributions. For example, one can simply count how often a determiner is followed by a noun, an adjective or some other class. Similarly, one can count how many often a verb label emits “dog” and how often the noun label emits “dog”.

Even in large training corpora, “dog” might very well never receive a verb label.[7] Nevertheless, “dog” can be a verb, for example in the sentence “Fans may dog Muschamp, but one thing’s for certain: he did things the right way off the field.”. To avoid this kind of problems caused by data sparsity, both emission and transition counts need to be smoothed.

[7] There are ten occurrences of “dog” in the Penn Treebank and all of them are analyzed as nouns.

Counting for Supervised ML Estimation When HMMs are used in linguistic labeling tasks, such as part-of-speech tagging, they are usually estimated in a supervised manner.[8] Each label is thought to represent a hidden variable, and the HMM models the transitions from one label type to another and the emission of words from each label type.

Mr.	NNP
Vinken	NNP
is	VBZ
chairman	NN
of	IN
Elsevier	NNP
N.V.	NNP
,	,
the	DT
Dutch	NNP
publishing	VBG
group	NN
.	.

Figure 4.6: Tagged text from the Penn Treebank.

Figure 4.6 shows one sentence from the Penn Treebank (Marcus et al., 1993). The sentence is labeled with POS tags which are taken to be the hidden states of an HMM. When estimating an HMM tagger for the corpus, transitions probabilities, for example $t_{\texttt{NNP},\texttt{VBZ}}$, and emission probabilities, for example $e_{\texttt{NNP}}(\texttt{Dutch})$ can in principle be computed directly from the corpus. For example the transition probability $t_{\texttt{NNP},\texttt{VBZ}}$ and the emission probability $e_{\texttt{NNP}}(\texttt{Dutch})$ in the Penn Treebank are simply:

$$t_{\texttt{NNP},\texttt{VBZ}} = \frac{\text{Count of POS tag pair } \texttt{NNP VBZ} \text{ in the corpus}}{\text{Count of POS tag } \texttt{NNP} \text{ in the corpus}} = \frac{4294}{114053} \approx 0.04$$

$$e_{\texttt{NNP}}(\texttt{Dutch}) = \frac{\text{Number of times } \texttt{Dutch} \text{ was tagged } \texttt{NNP} \text{ in the corpus}}{\text{Count of POS tag } \texttt{NNP} \text{ in the corpus}} = \frac{14}{114053} \approx 1.2 \cdot 10^{-4}$$

Simple computation of co-occurrences is insufficient because of data-sparsity. Words do not occur with all POS tags in the training corpus and all combinations of POS tags are never observed. Sometimes this is not a problem. For example, "Dutch" could never be a preposition. We know that the probability that a preposition state emits "Dutch" is 0. However, there are at least three analyses that are perfectly plausible: noun (the Dutch language), adjective (property of being from The Nederlands) and proper noun (for example in the restaurant name "The Dutch").

Since "Dutch" occurs only 14 times in the Penn Treebank, it is not surprising that all of these analyses

[8] Such taggers are sometimes called *visible* Markov models (Manning and Schütze, 1999).

do not occur. Specifically, the noun analysis is missing. An HMM based on direct counts will therefore never analyze “Dutch” as a noun.

It is tempting to think that missing analyses are a minor problem because they only occur for relatively rare words such as “Dutch”. Unfortunately, a large portion of text consists of rare words. The problem therefore has very real consequences.

The usual approach is to use a family of techniques called *smoothing*. In smoothing, zero counts and all other counts are modified slightly to counter-act sparsity.

Smoothing of emission probabilities and transition probabilities differ slightly. For transition probabilities it is common practice to use counts of both tag pairs and single tags to estimate tag probabilities either in a back-off scheme or using interpolation (Brants, 2000).

Many systems such as the HMM tagger by Brants (2000) do not smooth emission probabilities for words seen in the training corpus. However, words *not* seen in the training corpus, or out-of-vocabulary (OOV) words still require special processing. The simplest method is to estimate combined statistics for words occurring one time in the training corpus and use these statistics for OOV words. However, word forms contain valuable information which this approach disregards. Another approach would be to build models to guess the analysis of OOV words using the longest suffix of the word shared with a word in the training data.

Brants (2000) employs a specialized emission model for OOV words, which combines both approaches. It assigns a probability $p(y|x)$ for any label $y \in \mathcal{Y}$ and an arbitrary word x based on suffixes s_i of the word different lengths. The subscript i indicates suffix length.

The model uses relative frequencies $\hat{p}(y|s_i)$ of label y given each suffix s_i of x that occurs in the training data. The frequencies for different suffix lengths are recursively combined into probability estimates $p(y|s_i)$ using successive interpolations

$$p(y|s_{i+1}) = \frac{\hat{p}(y|s_{i+1}) + \theta \cdot p(y|s_i)}{1 + \theta}.$$

The base case $p(y|s_0)$, for the empty suffix s_0, is given by the overall frequency of label type y in the training data, i.e. $p(y|s_0) = \hat{p}(y)$, and the interpolation coefficient θ is the variance of the frequencies of label types in the training data

$$\theta = \frac{1}{|\mathcal{Y}|-1} \sum_{y \in \mathcal{Y}} (\hat{p} - \hat{p}(y))^2.$$

Here $\hat{p}$ is the average frequency of a label type. Finally, $p(y|x) = p(y|s_I)$, where s_I is the longest suffix of x that occurs in the training data. However, a maximal suffix length is imposed to avoid over-fitting. Brants (2000) uses 10 for English. Moreover, the training data for the emission model is restricted to include only “rare” words, that is words whose frequency does not exceed a given threshold. This is necessary, because the distribution labels for OOV words usually differs significantly from the overall label distribution in the training data.

Brants (2000) does not discuss the choice of θ in great length. It is, however, instructive to consider

the effect of the magnitude of θ on the emission model. When the variance of label type frequencies, that is θ, is great, shorter suffixes and the prior distribution of label types will weigh more than long suffixes. This is sensible as (1) a high θ implies that the distribution of words into label types is eschewed a priori and (2) long suffix statistics are sparse and thus prone to overfitting. When θ is low, the prior distribution of word classes is closer to the even distribution. Therefore, there is no choice but to trust longer suffixes more.

For morphologically complex languages, the smoothing scheme employed by Brants (2000) may be inferior to a longest suffix approach utilized in Publication **II** and Lindén (2009). This may happen because productive compounding. For languages with writing systems that radically differ from English, such as Mandarin Chinese, suffix based methods work poorly. Other methods, such as basing the guess on all symbols in the word, may work better.

The EM algorithm for Unsupervised ML Estimation The Baum-Welch, or Expectation Maximization, algorithm for HMMs is an iterative hill-climbing algorithm, that can be used to find locally optimal parameters for an HMM given a number of unlabeled independent training examples which are drawn from the distribution that is being modeled by the HMM. Here is a short outline of the algorithm:

1. Random initialize the emission and transition parameters.
2. Use the forward-backward algorithm to compute posterior marginals over input positions.
3. Use the posterior marginals as *soft counts* to estimate new parameters.
4. Repeat steps 2 and 3 until the improvement of likelihood of the training data is below a threshold value.

In step 2, the algorithm computes the maximally likely state distribution for each position given the current parameters. In step 3, the state distributions for each position in the input data are used to infer the MAP parameters for the HMM. Therefore, the marginal probability of the training data has to increase on every iteration of steps 2 and 3, or possible remain the same, if the current parameters are optimal.

There are no guarantees that the optimum found by the EM algorithm is global. Therefore, several random restarts are used and parameters giving the best marginal probability for the training data are used.

A more formal treatment of the EM algorithm can be found in Bilmes (1997).

4.5 Model Order

The standard HMM presented above is called a *first order model* because the next hidden state is determined solely based on the current hidden state. This model is easy to estimate and resistant to over-fitting caused by data-sparsity, but it fails to capture some key properties of language. For example, in the Penn Treebank, the probability of seeing a second adverb `RB` following and adverb is approximately, 0.08. If the

first order assumption were valid, the probability of seeing a third adverb following two adverbs should also be 8%, however it is lower, around 5%.

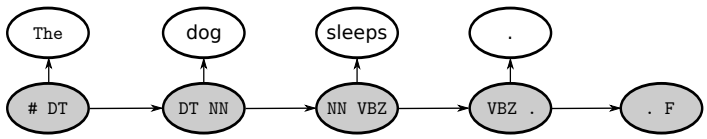


Figure 4.7: A second order HMM for a sentence.

The example with adverbs is poor at best, but it illustrates the kind of effect *second order* information can have. Second order HMMs are models where transitions are conditioned on two preceding hidden states. Equivalently, in POS tagging, the hidden states can be taken to be pairs of POS tags, e.g. `(DT, NN)`. In such a model transitions can only occur to a subset of the hidden state set. For example a transition from `(DT, NN)` to `(NN, VBZ)` is possible, but a transition to `(JJ, NN)` is impossible. Figure 4.7 illustrates a path with legal transitions.

Figure 4.7 implies that emissions in a second order model are conditioned on two labels like the transitions. However, many existing HMM based POS tagging systems such as Brants (2000) condition emissions only on one label, that is use $e_{t_i,t_{i-1}}(w_i) = p(w_i \mid t_i)$ instead of $e_{t_i,t_{i-1}}(w_i) = p(w_i \mid t_{i-1},\, t_i)$. The reason is probably data-sparsity. Therefore, these systems cannot be called HMMs in the strictest sense of the word. They should instead be called trigram taggers.

Halácsy et al. (2007) show that it is possible to maintain the correct HMM formulation, over-come the data sparsity problem and achieve gains over the more commonly used trigram tagger. However, they fail to describe the smoothing scheme used, which is crucial. This defect is partly remedied by the fact that the system is open-source. One of the chief contributions of Publication **II** was to investigate the effect of different ways of estimating the emission parameters in a generative trigram tagger paying attention to smoothing.

Increasing model order unfortunately leads to increased data sparsity because the number of hidden states increases. Therefore, smoothing transition probabilities is even more important than in the first order case.

An alternative to increasing model order, is to use so called latent annotations (Huang et al., 2009) in an otherwise regular first order HMM. Conceptually, each label for example `NN` is split into a number of sub-states `NN1`, `NN2` and so on. Expectation maximization is used to train the model in a partly supervised fashion. Splitting labels, and indeed any increase in order, is probably works better for label sets with quite few labels. Otherwise, it will simply contribute to data sparsity.

4.6 HMM taggers and Morphological Analyzers

The inventory of POS labels that are possible for a given word form tends to be small. For example the English “dog” can get two of the Penn Treebank POS tags singular noun `NN` and `VB` infinitive verb form. The remaining 43 POS tags can never label “dog”. Consequently, in an HMM POS tagger, only the states corresponding to `VB` and `NN` should ever emit the word “dog”.

A `tag dictionary` (Brants, 2000) can be used in combination with the Viterbi algorithm to limit the set of hidden states that could emit a word. The tag dictionary can be constructed from the training corpus. Additionally, an external lexical resource, such as a morphological analyzer, can be used. Such a lexical resource can help to compensate for missing statistics for OOV words. In the frequent setting, where most rare words have quite few analyses, this can have a substantial effect on tagging accuracy.

Chapter 5

Generative Taggers using Finite-State Machines

In this Chapter, I will present an implementation of HMMs using *weighted finite-state machines*. It is further investigated in Publications **I** and **II**. The implementation allows for extensions of the HMM model in the spirit of Halácsy et al. (2007), who utilize label context in the emission model of an HMM. It also allows for applying global grammatical constraints. I will first present a short summary of the most important aspects of finite-state calculus and then present the finite-state implementation of HMMs.

5.1 Weighted Finite-State Machines

Automata Weighted Finite-state automata are a data structure for representing algorithms that solve the decision problem of a regular language. A string can be either accepted or discarded by a an automaton with some weight. Typically, weights bear resemblance to probabilities and if they are interpreted as probabilities, an automaton defines a distribution over the set of strings.

Figure 5.1 presents a finite-state which recognizes a subset of noun phrases in the Penn Treebank. It illustrates the key components of a finite-state automaton M

1. A finite set of states Q_M ($\{0,\ 1,\ 2,\ 3,\ 4\}$ in Figure 5.1).
2. An alphabet Σ_M (the POS labels in Penn Treebank in Figure 5.1).
3. A unique initial state I_M (0 in Figure 5.1).[1]

[1] Some formulations allow for several initial states with initial weights. This does, however, not increase the expressiveness of

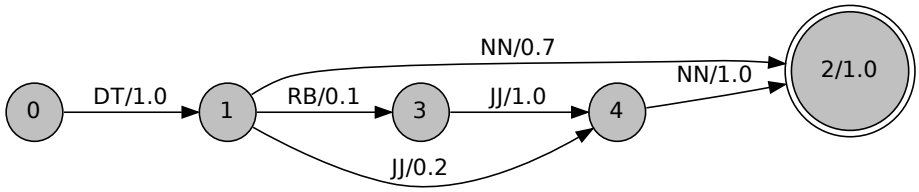


Figure 5.1: A finite-state machine accepting a subset of the singular noun phrases in Penn Treebank.

4. A set of final states $F_M = \{r_1, ..., r_m\} \subset Q_M$ with associated final weights $\mathrm{f}(r_i)$ ($\{2\}$ and 1.0 in Figure 5.1).[2]

5. A transition function τ_M, which specifies a, possibly empty, set

$$\tau_M(q, x) = \{(q_1, w_1),\ ...,\ (q_n, w_n)\}$$

of target states and transition weights for each symbol $x \in \Sigma_M$ in each state $q \in Q_M$ (in Figure, 5.1 the relation is represented by the arrows in the graph).

When $\tau_M(a, q)$ is either empty or a singleton set for all a and q, the automaton M is called *deterministic* and I write $\tau_M(a, q) = (q_1, w_1)$ instead of $\tau_M(a, q) = \{(q_1, w_1)\}$.

Weight Semirings Transition and final weights should form a algebraical structure $\mathbb{K}$ called a semiring which is an abstraction of the structure of positive real numbers under addition and multiplication. Consequently, two operations, addition $\oplus$ and multiplication $\otimes$, are defined in $\mathbb{K}$. The addition operation should be associative, commutative and have an identity element $\mathbb{0}$. The multiplication operation should also be associative and commutative. If it has an identity element, that element is denoted by $\mathbb{1}$. Multiplication should distribute over addition (Allauzen et al., 2007).

As noted above, the prototypical example of a weight semiring is given by the positive real numbers with the regular addition and multiplication of reals. This weight semiring, called the *probability semiring*,[3] can be used to represent a probability distribution over the set of strings. In practice, this weight semiring is however rarely used. Because of numerical stability concerns, a logarithmic transformation $x \mapsto -log(x)$ is often applied to the weights and operations in the probability semiring. This gives the *log semiring*, where weights belong to $\mathbb{R}$, multiplication is given by $x \otimes y = x + y$ and addition by $x \oplus y = -\log(\exp(-x) + \exp(-y))$.

The addition operation of the log semiring $-\log(\exp(-x)+\exp(-y))$ is slow in practice. Moreover, $-\log(\exp(-x) + \exp(-y)) \approx x$ when $x \ll y$. Therefore, the addition operation in the log semiring is

the formalism.

[2] Alternatively, we could also specify a final weight for every state. Then the actual final states would be the ones wit non-zero weight.

[3] Even though the probability semiring is intended for representing probability distributions, weights cannot be restricted to $[0, 1]$ because the semiring has to be closed under addition.

often replaced by $x \oplus y = \min(x, y)$. This gives the *tropical semiring* which is the semiring used in this thesis.

I denote the transition weight of the transition leaving state q_1 with symbol x and ending up in q_2 by $\mathrm{w}(\tau_M(q_1, x), q_2)$. As a notational convenience, $\mathrm{w}(\tau_M(q_1, x), q_2) = \mathbb{0}$, if there is no transition from q_1 to q_2. An automaton M assigns a weight to a path of states $p = (q_0, \ ..., \ q_{n+1})$, where $q_0 = I_M$ and $q_n \in F_M$, and a string $s = s_1...s_n \in \Sigma_M^n$. The weight is $\mathrm{w}_M(s, p)$ is given by Equation 5.1. The total weight $\mathrm{w}_M(s)$ assigned to string s by automaton M is given by Equation 5.2

$$\mathrm{w}_M(s, p) = \mathrm{f}(q_{n+1}) \otimes \bigotimes_{i=0}^{n} \mathrm{w}_M(\tau_M(q_i, s_i), q_{i+1}) \tag{5.1}$$

$$\mathrm{w}_M(s) = \bigoplus_{p \in Q_M^{n+1}} \mathrm{w}_M(s, p) \tag{5.2}$$

Closure Properties It is well known that regular languages are closed under many unary and binary operations: union, negation, concatenation and reversion (Sipser, 1996). Similarly, the class of weighted finite-state machines is closed under these operations and efficient algorithms for computing these operations exist. Table 5.1, summarizes the properties of these algorithms.

Optimization As stated above, a finite state machine where each symbol and state is associated to maximally one transition is called deterministic. It is well known that every finite-state machine corresponds to a deterministic machine and this holds for weighted finite-state machines as well given light assumptions on the weight semiring (Mohri et al., 2002). A determinization algorithm can be applied to any weighted finite-state machine in order to produce a deterministic machine which accepts exactly the same set of weighted strings as the original machine.

N-Best The finite-state implementation of morphological taggers presented in Publications **I** and **II** compiles a weighted finite-state machine which represents a sentence and all of its alternative label sequences as paths. The path weights correspond to the joint probabilities of the sentence and label sequences. An n-best algorithm (Mohri et al., 2002) can then be used to efficiently extract the path that carries the highest probability. The best path can be found in time $O(|Q|\log(|Q|) + |\tau|)$, where Q is the state set of the sentence machine and $|\tau|$ is the number of transitions in the machine.

5.2 Finite-State Implementation of Hidden Markov Models

As seen in Chapter 4, a generative HMM can be decomposed into an emission model $p(x_i|y_i)$ of emissions x_i given states y_i a transition model $p(y_{n+1}|y_1, ..., y_n)$, which models the conditional distribution of a state

Operation	Symbol	Definition
Power	M^n	$\mathrm{w}_{M^n}(s) = \bigoplus_{s_1 \ldots s_n = s} \mathrm{w}_M(s_1) \otimes \ldots \otimes \mathrm{w}_M(s_n)$
Closure	$M^* = \bigoplus_{n=0}^{\infty} M^*$	$\mathrm{w}_{M^*}(s) = \bigoplus_{s_1 \ldots s_k = s} \mathrm{w}_M(s_1) \otimes \ldots \otimes \mathrm{w}_M(s_k),$ where $k \in \mathbb{N}$
Union	$M_1 \oplus M_2$	$\mathrm{w}_{M_1 \oplus M_2}(s) = \mathrm{w}_{M_1}(s) \oplus \mathrm{w}_{M_2}(s)$
Concatenation	$M_1.M_2$	$\mathrm{w}_{M_1.M_2}(s) = \bigoplus_{s_1 s_2 = s} \mathrm{w}_{M_1}(s_1) \otimes \mathrm{w}_{M_2}(s_2)$
Intersection	$M_1 \cap M_2$	$\mathrm{w}_{M_1 \cap M_2}(s) = \mathrm{w}_{M_1}(s) \otimes \mathrm{w}_{M_2}(s)$
Composition	$M_1 \circ M_2$	$\mathrm{w}_{M_1 \circ M_2}(s{:}t) = \bigoplus_r \mathrm{w}_{M_1}(s{:}r) \otimes \mathrm{w}_{M_2}(r{:}t)$

Table 5.1: A selection of operators for weighted finite-state machines given by Allauzen et al. (2007).

y_{n+1} given a state history $y_1, ..., y_n$. As shown in Publications **I** and **II** both of these can be compiled into weighted finite-state machines.

A generative HMM can be represented as a weighted finite-state machine in several ways. The implementation presented in Publication **I**, however, allows for enriching the emission model by conditioning them on neighboring word forms and labels.

The main idea of the implementation discussed in Publication **I** is to represent a labeled sentence as a string of word form label pairs as in Figure 5.2. The emission and transition models are implemented as weighted finite-state machines which assign weights to such labeled sentences. Because of this representation, both the emission model and transition model can access information about the sequence of word forms and their labels. Therefore, the emission and transition models can use more information than in a regular HMM model.

In a normal HMM tagger, extension of the emission and transition model requires changes to inference algorithms used by the tagger. In contrast to a traditional HMM tagger, the finite-state tagger presented in Publications **I** and **II** uses an n-best paths algorithm for inference. This is a general algorithm which can be applied on any model that can be represented as weighted finite-state machine. Therefore, extending the emission and transition models requires no changes to the inference procedure.

In this Section, I will discuss the implementation of a HMM model. In the next Section, I will show how emission and transition models can be extended.

The	DT	dog	NN	sleeps	VBZ	.	.

Figure 5.2: The representation of a labeled sentence as a single sequence.

Emission Model Let $x = (x_1, ..., x_T)$ be a sentence and let $Y_t = \{y_t^1, ..., y_t^n\}$ be the set of possible labels for word x_t. We can construct a very simple finite-state machine X_t which recognizes the word x_t followed by one of its possible labels $y_t^i \in Y_t$ and assigns that combination a log weight corresponding to the probability $p(x_t \mid y_t^i)$. As in the case of a regular HMM tagger, $p(x_t \mid y_t^i)$ can be estimated from the training data. For OOV words, we can use a guesser, for example the one presented in Chapter 4.

As an optimization, only the most probable labels for each word can be included in the emission model. However, it is completely possible to include all labels for each word.

The individual emission machines X_t can be combined into a sentence model using concatenation as shown in Figure 5.3. The paths through the sentence model correspond to the possible label assignments of sentence x.

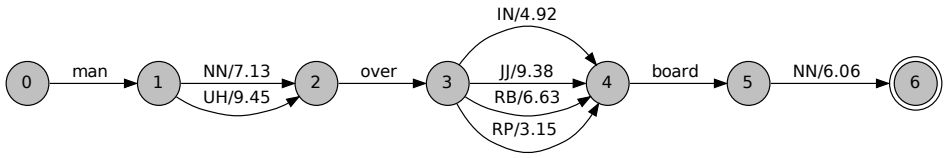


Figure 5.3: An example of a sentence model. The weights on the arcs are negative logarithms of emission probabilities. Only a subset of the possible labels are shown in the picture.

Transition Model As stated above, Publications **I** and **II** represent labeled sentences as a sequence of pairs where each pair consists of a word form and a label. The transitions model assigns weight to such sequences. I will explain the construction of the transition model in three phases:

1. How to construct a model which assigns weight $-\log(p(y_{n+1} \mid y_1, ..., y_n))$ to plain label n-grams $y_1, ..., y_{n+1}$.
2. How to extend the model to assign weight to an n-gram of word form label pairs.
3. How to score an entire labeled sentence.

The construction presented below will result in a number of deterministic finite-state machines whose combined effect (the intersection of the machines) corresponds to the n-gram model in a standard HMM.

Scoring one label n-gram The transition distributions $p(y_i \mid y_{i-1}, ..., y_{i-n})$ in an nth order HMM encode the likelihood of label sequences. I will first consider the problem of constructing a machine which represents transition weights for isolated label n-grams.

To emulate transitions weights in an HMM using finite-state calculus, we can first compile a machine T which accepts any sequence of $n + 1$ labels $y_1, ..., y_{n+1}$. The weight assigned by the machine to one of these paths can be estimated from a training corpus for all sequences that occur in the corpus. Some form of smoothing is required to score label n-grams missing from the training corpus. In Publication **II**, a very simple form of smoothing is used. Each n-gram, not occurring in the training corpus, receives an identical penalty weight $-\log(1/(N + 1))$, where N is the size of the training corpus.

The machine T will be quite large. If it is deterministic and has one path corresponding to each label n-gram $y_1, ..., y_{n+1}$, where $y_i \in \mathcal{Y}$, each non-terminal state in the machine will have $\mathcal{Y}$ transitions. Because

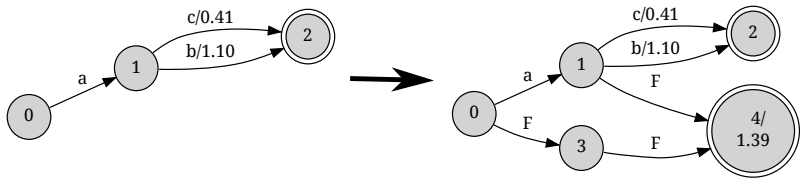


Figure 5.4: Failure transitions (with symbol F) are added to a bigram model. With failure transitions, the model will accept previously missing n-grams, such as aa and bb with penalty weight 1.39.

T encodes the weight $p(y_{n+1} \mid y_1, ..., y_n)$ for each label n-gram, it will have a large amount of states when many label n-grams occur in the training corpus. It is difficult to present a formal analysis of the size of T as a function of the number of distinct label n-grams in the corpus. An example can, however, illustrate the number of states that are typically required.

For the FinnTreeBank corpus, 8801 non-terminal states are required to represent T for $n = 2$. As the corpus has 1399 distinct morphological labels, this translates to approximately 12 million transitions. When using add one smoothing for label n-grams missing in the training corpus, most paths will have the same weight. This fact allows for an optimization which substantially reduces the size of T.

In order to, reduce the size of the model, so called *failure transitions* can be used (Knuth et al., 1977, Mohri, 1997).[4] A failure transition in a state q, will match any symbol which does not have another outgoing transition in q. The failure transitions will go to sink states, which encode the penalty weight for unseen label n-grams. When failure transitions are used to encode label n-grams that are missing from the training corpus, most states will only have a few outgoing transitions. Figure 5.4 illustrates a bigram model with failure transitions.

I will now outline the procedure to compute a machine with failure transitions in the general case. We first need an auxiliary definition. For a state $q \in Q_T$, let $n(q)$ be the length of the shortest symbol string required to reach state q from the initial state q_0. Now, given a machine T that recognizes every label n-gram *occurring in the training corpus*, a corresponding machine T_f with failure transitions can be computed.

1. $n + 1$ new sink states are added: $Q_{T_f} = Q_T \cup \{s_1, ..., s_{n+1}\}$. The state s_{n+1} is final and its final weight is the penalty weight for unseen n-grams.

2. A failure symbol is added: $\Sigma_{T_f} = \Sigma_T \cup \{f\}$, where $f \notin \Sigma_T$.

3. Transitions are copied from T: $\tau_{T_f}(q, a) = \tau_T(q, a)$ for all $q \in Q_T$ and $a \in \Sigma_T$.

4. $\tau_{T_f}(s_1, f) = \{(s_{i+1}, \mathbb{1})\}$ for $i <= n$.

5. Failure transitions are added: $\tau_{T_f}(q, f) = \{(s_{n(q)+1}, \mathbb{1})\}$, for all $q \in Q_T$.

[4]The failure transitions used by Mohri (1997) differ from the ones used in this thesis because they do not consume input.

Adding word forms The current transition model scores label n-grams. However, because we represent labeled sentences as sequences of word form label pairs, we need to include word forms in the model. This can be accomplished by adding a number of new states and failure transitions to the model. When implementing a standard HMM tagger, the added failure transitions will simply skip word forms. Figure 5.5 demonstrates the construction for the transition model in Figure 5.4.

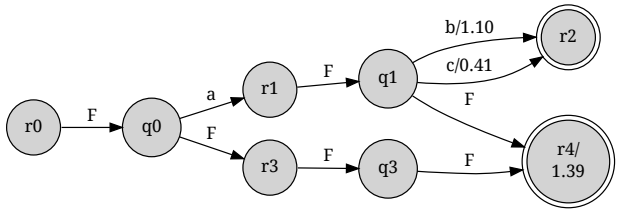


Figure 5.5: The transition model in Figure 5.4 is augmented with additional failure transitions and states in order to be able to handle word forms.

We construct a new machine T_w which accepts n-grams of word form label pairs. Let $Q_{T_f} = \{q_0, ..., q_k\}$, then $Q_{T_w} = Q_{T_f} \cup r_0, ..., r_k$, where $r_i \notin Q_{T_f}$. Let r_0 be the start state of Q_{T_f} and let $F_{T_f} = F_{T_w}$. The transitions function τ_{T_w} is defined in the following way.

1. $\tau_{T_w}(r_i, f) = \{(q_i, \mathbb{1})\}$.
2. $\tau_{T_w}(q_i, x) = \{(r_j, w)\}$ for all $x \in \Sigma_{T_w}$, if $\tau_{T_f}(q_i, x) = \{(q_j, w)\}$.

Consider two states $q_1, q_2 \in Q_M$, in a machine M with failure transitions. Failure transitions in q_1 and q_2 may match a different set of symbols. For example, If q_1 has a transition with symbol $a \in \Sigma_M$ and q_2 does not, then a will match the failure transition in q_2 but it will not match the failure transition in q_1. This is a problem when the determinization algorithm is applied to M because determinization joins states. State joining may change the language accepted by M and the weights assigned to strings.

It is highly desirable that the transition model of the finite-state implementation of an HMM tagger is deterministic because of reduced tagging time. However, as noted above, we cannot use standard determinization with machines which have failure transitions. Therefore, the construction presented below will produce deterministic machines without resorting to determinization.

Scoring a sentence We will now see how the transition model for scoring an isolated label n-gram can be extended for scoring an entire sentence. Given the machine T_w which scores one n-gram, we can form the Kleene closure of T_w. Since M_n is an acyclic machine accepting strings of equal length (that is $2n$: n word forms and n labels), we can easily compute a deterministic Kleene closure T_w^* of T_w by re-directing transitions going to final states into the initial state[5]

1. $\Sigma_{T_w^*} = \Sigma_{T_s}$.

[5]It can easily be seen that this construction fails if the machine accepts strings of unequal lengths.

2. $Q_{T_w^*} = Q_{T_w} - F_{T_w}$.

3. $F_{T_w^*} = \{q_0\}$ and $f(q_0) = \mathbb{0}$.

4. $\tau_{T_w^*}(a, q) = \{(q', w)\}$, if $\tau_{T_w}(a, q) = \{(q', w)\}$ and $q' \notin F_{T_w}$.

5. If $\tau_{T_w}(a, q) = \{(q', w)\}$ and $q' \in F_{T_w}$, then $\tau_{T_w^*}(a, q) = \{(q_0, w + f(q'))\}$.

The machine T_w^* will score entire labeled sentences, however, it only scores some of the trigrams in the sentence, namely, the ones starting at positions divisible by $n+1$. Fortunately we can form $n+1$ machines T_0 ... T_n which will score all remaining n-grams. Let $T_0 = T_w^*$ and let $T_{i+1} = F.F.T_i$. Intuitively each T_i will skip i word form label pairs.

The final scoring of all possible labeled sentences, corresponding to an input sentence x, is accomplished by intersecting the sentence model X and each of the T_i using an intersection algorithm which handles failure symbols correctly. In Publications **I** and **II** a parallel intersection algorithm (Silfverberg and Lindén, 2009), however, this is not actually required. The intersections can be performed sequentially.

Smoothing As observed in Chapter 4, a second order model usually gives the best results in morphological tagging. However, a pure second order model suffers from data sparsity which degrades it performance. This can be avoided using smoothing. In Publications **I** and **II**, smoothing is accomplished by using first and zeroth order transition models in addition to the second order transition model. To get the combined effect of all models, each of them is intersected with the sentence model.

Usually, for example in Brants (2000), the transition probabilities $p(y_i|y_{i-1}, y_{i-2})$ are a linear interpolation of the probability estimates $\hat{p}$ for different orders as shown in equation 5.3. Brants (2000) sets the values for α_i using deleted interpolation and cross validation. When $\sum_i \alpha_i = 1$, it is easily seen that Equation 5.3 defines a probability distribution over y_i.

$$p(y_i|y_{i-1}, y_{i-2}) = \alpha_2 \hat{p}(y_i|y_{i-1}, y_{i-2}) + \alpha_1 \hat{p}(y_i|y_{i-1}) + \alpha_0 \hat{p}(y_i) \tag{5.3}$$

Linear interpolation is not possible when using the finite-state implementation presented in this chapter because intersection of weighted machines corresponds to multiplying probability estimates, not to adding them. Therefore, Publications **I** and **II** define the score $s(y_i|y_{i-1}, y_{i-2})$ of a labeled sentence as the weighted product given in Equation 5.4. Inference corresponds to finding the label sequence which maximizes the score. The optimal values for the exponents α_i in Equation 5.4 are found by optimizing the tagging accuracy on held out data using grid search.

$$s(y_i|y_{i-1}, y_{i-2}) = \hat{p}(y_i|y_{i-1}, y_{i-2})^{\alpha_2} \hat{p}(y_i|y_{i-1})^{\alpha_1} \hat{p}(y_i)^{\alpha_0} \tag{5.4}$$

The weighted product $s(y_i|y_{i-1}, y_{i-2})$ given by 5.4 does not necessarily define a probability distribution over y_i. It can, however, easily be normalized to give one. When the score is normalized, it can be seen as a special case of the family of distributions defined by Equation 5.5. Here the parameter val-

ues $r_2(y_i|y_{i-1}, y_{i-2})$, $r_1(y_i|y_{i-1})$, $r_0(y_i)$ can be arbitrary positive real numbers. Each assignment of the parameter values defines a probability distribution over y_i.

$$p(y_i|y_{i-1}, y_{i-2}) = \frac{r_2(y_i|y_{i-1}, y_{i-2})r_1(y_i|y_{i-1})r_0(y_i)}{\sum_{y\in\mathcal{Y}} r_2(y|y_{i-1}, y_{i-2})r_1(y|y_{i-1})sr0(y)} \tag{5.5}$$

The interpretation of the weighted product in Equation 5.4 given by Equation 5.5 reveals a problem. There is no guarantee that the parameter values $r(y_i|y_{i-1}, y_{i-2}) = \hat{p}(y_i|y_{i-1}, y_{i-2})^{\alpha_2}$, $r_1(y_i|y_{i-1}) = \hat{p}(y_i|y_{i-1})$ and $r_0(y_i) = \hat{p}(y_i)^{\alpha_0}$ result in a model which fits the training data maximally well in the sense that is discussed in Chapter 3. This may have contributed to the inferior tagging accuracy of the system when compared to Brants (2000) which is seen in the experiments in Publication **II**. This problem lead the author to consider conditional random fields presented in Chapter 6, which naturally support a product formulation.

5.3 Beyond the Standard HMM

The real strength of the system presented in this chapter lies in its capability of easily incorporating information not usually present in a generative HMM tagger. Halácsy et al. (2007) show that enriching the emission model of an HMM tagger by including label context can improve tagging results. Instead of the usual emission model $p(x_t \mid y_t)$ which conditions each word on its morphological label, Halácsy et al. (2007) instead use a model $p(x_t \mid y_{t-1}, y_t)$, where the emission is conditioned on preceding label context. As stated in Chapter 4, this is in fact not an extension to the standard second order HMM. Instead, it is the faithful implementation of the second order HMM model. The definition $p(x_t \mid y_t)$, used by for example Brants (2000), is incorrect in a second order HMM. It is probably used because of data sparsity. Nevertheless, Halácsy et al. (2007) show that the correct formulation can result in improved tagging accuracy.

Richer local structure The model presented by Halácsy et al. (2007) can easily be implemented as a finite-state machine by a slight modification to the compilation of the sentence model as described in Publication **I** and it can be extended to model $p(x_t \mid y_{t-1}, y_t, y_{y+1})$. Moreover, it is possible to condition transitions on word forms as shown in Publication **I**. Both of these modifications are shown to give statistically significant improvements over the standard baseline model. The problem with including such information in the emission and transition models is that it violates the conditional independence assumptions in the generative HMM model. This is yet another reason to consider alternative models such as the conditional random field or averaged perceptron.

Global Constraints In addition to local changes to the emission and transition models, it would also be possible to include global probabilistic constraints to the model. These are constraints that apply on the entire input sentence. A simple example of such a constraint is the existence or frequency of finite verb

forms in the sentence. Another family of interesting global constraints is given by syntactic and semantic valency of words (Baker et al., 1998). Such information could be represented as a weighted finite-state machine. Similarly to the enriched locally emission and transition models, global constraints also violate the independence assumption of the generative HMM model.

5.4 Summary of Publications I, II and III

Publication **I** presents the finite-state implementation of HMMs introduced in this Chapter and Publication **II** presents experiments using the model on the Penn Treebank and a Finnish data set. The taggers presented in Publications **II** are used in Publication **III** to implement a language model for a context-sensitive finite-state spelling corrector.

Publication I The main contribution of this publication is to present the finite-state implementation of HMMs. The publication presents experiments on morphological tagging of Finnish, English and Swedish but the experiments presented in the publication are nearly void of value because they were conducted on machine labeled data and the amount of training data was unrealistic (1 million sentences for each language). Both factors contribute to extremely good, and quite unrealistic, tagging accuracy for all languages on test data. Still, the extreme size of the training set does demonstrate that the method can use large amounts of training data.

For Finnish, experiments on machine labeled data were the only option because, at the time, there was no freely available hand-labeled morphologically tagged corpus available. For Swedish and English, established data sets should have been used.

The formulation of the probabilistic model in Publication **I** differs from the formulation in this Chapter in two respects. Instead of the usual transitions probabilities $p(y_t \mid y_{t-n}, ..., y_{t-1})$, Publication **I** uses the joint probability $p(y_{t-n}, ..., y_t)$. The model can therefore not be seen as an actual HMM. Additionally, the publication uses lexicalized transition probabilities. The final probability is thus $p(l_{t-n}, y_{t-n}, ..., l_t, y_t)$, where the l refer to lemmas. This is possible because of the extremely large training set.

Although the experiments in Publication **I** are flawed, the paper is included in the thesis because it describes the finite-state implementation for HMM taggers and can be seen as a natural starting point for Publications **II** and **III**.

Publication II The main contribution of this publication is to present experiments on a standard data set for POS tagging of English, the Penn Treebank 2. However, because of insufficient knowledge at the time, the experiments were performed on a non-standard version of Penn Treebank 2. Publication **I** uses the data splits introduced by Collins (2002) but the data from the `tagged` sub-directory in the Penn Treebank 2 distribution. As Toutanova et al. (2003) explain, it is conventional to extract the POS tagged sentences from the parse trees in the `parsed` sub-directory in the distribution. Unfortunately, this makes the reported accuracies approximately 0.3 %-points higher than they would be if the experiments were

performed on the correct data set. For Finnish, experiments were performed on machine labeled data by necessity, similarly as in the case of Publication **I**.

Publication **II** presents models for Finnish and English which use enriched emission models described in Section 5.3, which are inspired by the HunPos system (Halácsy et al., 2007). The taggers are evaluated against a standard HMM baseline. The results are also compared against tagging accuracies reported in Brants (2000) and Halácsy et al. (2007). Because of the unfortunate mix-up with the Penn Treebank data set, the results for English are not comparable between the different tagging systems. However, Publication **II** does show that the enriched emission models yield clear improvements over baseline. Moreover, the final system outperforms HunPos by 0.4%-points on the Finnish data set. Because the data set is machine labeled, this result may of course not be convincing.

Publication III This publication applies the taggers presented in Publications **I** and **II** to the task of context-sensitive spelling correction. Many spelling correction systems determine the best spelling correction for a misspelled word based solely on the misspelled word form itself. Typically, correction candidates that have a small edit distance to the misspelled word form are ranked higher than more remote candidates. Context-sensitive spelling correction systems additionally utilize surrounding words to rank correction candidates.

For English, plain word context can improve the accuracy of spelling correction (Brill and Moore, 2000). For example "cat" is much more likely in the context "the _ miaowed" than "car" is. As Publication **II** shows, word context does improve results for Finnish as well. However, Publication **III** also shows that a morphological tagger can yield greater improvements in accuracy for both English and Finnish when using comparable amounts of training data for the tagger and word context model. Of course, the word context model can be trained on unlabeled data. Therefore, there is in principle no obstacle to using arbitrarily much training data.

The system presented in Publication **III** first generates a set of correction candidates for the misspelled word using a finite-state spelling correction system based on edit distance. It then uses a generative morphological tagger for selecting the best candidate. The misspelled form is replaced with each correction candidate c_1, ..., c_n in turn producing n sentences x_1, ..., x_n. The sentences are then tagged which results in n tag sequences y_1, ..., y_n. The candidates c_i are finally ranked according to the joint probability of the sentence and label sequence $p(y_i, x_i)$.[6]

The spelling correction system using the morphological tagger probably yields better results, especially for English, because it is less susceptible to data sparsity than the word context model. As noted above, the amount of training data for the word context model could, however, be increased. This is likely to gradually improve accuracy. At the same time it, however, increases the size of the model. The spelling correction system that uses the morphological tagger can be more compact and thus more practicable while delivering comparable accuracy.

[6] In fact only a sub-sequence of the sentence is tagged as an optimization, however, the basic idea is the same.

Chapter 6

Discriminative Models

As seen in Chapter 4, a first order HMM POS tagger can be viewed as a process which alternates between sampling a word from a label specific observation distribution and sampling the next morphological label from a label specific transition distribution. The emitted word and the next morphological label are conditioned *solely* on the current morphological label. These independence assumptions are harsh. For example, collocations cannot be adequately modeled because the model does not include direct information about word sequences.

Although information about collocations and orthography is quite useful in morphological tagging, it is often difficult to incorporate such information in a generative model. As Sutton and McCallum (2012) note, two principal approaches could be attempted:

1. Extending the emission model presented in Chapter 4 to incorporate additional sources of information.

2. Replacing the usual emission model with a Naive Bayes' model which in theory can handle arbitrary features.

Approach 1 is difficult in a fully generative setting because the emission model needs to account for the complex dependencies that exist between sentence level context and orthography. There simply does not seem to exist a straightforward way of modeling the dependencies.[1]

Approach 2, that is making the naive Bayes assumption, corresponds to disregarding the dependencies that exist between orthography, word collocations and other sources of contextual information. In the domain of named entity extraction, which is closely related to morphological tagging, Ruokolainen and Silfverberg (2013) show that approach 2 also fails. In fact the experiments in the paper indicate that adding richer context modeling such as adjacent words may worsen the performance of a tagger with a

[1] Although recent development in deep learning might make this approach viable.

Naive Bayes emission model. One reason for this may be that the richer sources of information are often correlated and this violates the independence assumption of the Naive Bayes model. This can cause it to give overly confident probability estimates (Sutton and McCallum, 2012). When the emission probabilities are over confident, and thus biased, combining them with the transition model can be problematic.

In contrast to generative sequence models, discriminative sequence models such as Maximum Entropy Markov Models (Ratnaparkhi, 1998) and Conditional Random Fields (Lafferty et al., 2001) can incorporate overlapping sources of information. They model the conditional distribution of label sequences $p(y \mid x)$ directly instead of modeling the joint distribution $p(x,\ y)$. Therefore, they do not need to model dependencies between words and orthographic features.

Discriminative models assign probabilities $p(y \mid x)$ for label sequences $y = (\mathrm{DT},\ \mathrm{NN},\ \mathrm{VBZ},\ .)$ and word sequences $x = (\mathrm{The},\ \mathrm{dog},\ \mathrm{eats},\ .)$ by extracting *features* from the input sentence and label sequence. Examples of features include **the current word is "dog" and its label is NN** and **the previous label is DT and the current label is NN**. Each feature is associated with a parameter value and the parameter values are combined to give the conditional likelihood of the entire label sequence. Naturally, the label sequence which maximizes the conditional likelihood given sentence x is the label sequence returned by the discriminative POS tagger.

In generative models, emissions and transitions are independent. Both are determined exclusively based on the current label. In contrast, there are no emissions or transitions in a discriminative model. Instead, it is customary to speak about unstructured features which relate the label in one position to the input sentence, and structured features, which incorporate information about the label sequence. Simplifying a bit, discriminative models make no independence assumptions among features relating to a single position in the sentence. This allows for improved fit to training data but parameter estimation becomes more complex as we shall soon see. Moreover, discriminative models are more prone to over-fitting.[2]

6.1 Basics

In this section, I will describe a CRF POS tagger from a practical point-of-view. The tagging procedure encompasses two stages: feature extraction and inference using an exact or approximate inference algorithm. Inference in CRFs is very similar to inference in HMMs. We did not discuss feature extraction in association to HMMs. The discussion was omitted because HMM taggers use a fixed set of features (the current word and preceding labels). In contrast, CRF taggers can incorporate a variety of user defined features.

As seen above, features are true logical propositions that apply to a position t in a labeled sentence. They connect aspects of the input sentence to the label at position t.[3] They consist of two components: a

[2] This is of course an example of the famous bias-variance trade-off (Geman et al., 1992).

[3] Although the original first order CRF formulation by Lafferty et al. (2001) allows for features that refer to both unstructured and structured information at the same time, the author has found that such features do not improve model performance significantly. They, however, do increase model size substantially. Therefore, the models used in the thesis extract purely unstructured features, which relate one label to the sentence, and structured features, which only relate adjacent labels to each other.

feature template, for example **The current word is "dog"** or **the previous label is DT**, and a label **the current label is NN**. The set of features recognized by a CRF POS tagger consists of all conjunctions $f\&y$ of a feature template f and label y. For example, the feature **The current word is "dog" and the current label is DT** can be formed although it is unlikely that this feature would ever be observed in actual training data.

Feature template
The current token is $w \in W$
The current token has prefix $p \in P$
The current token has suffix $s \in S$
The current token contains a digit
The current token contains a hyphen
The current token contains an upper case letter
The previous token is $w \in W$
The next token is $w \in W$
The token before the previous token is w
The token after the next token is w

Table 6.1: The set of unstructured feature templates introduced by Ratnaparkhi (1996)

Ratnaparkhi (1996) introduced a rather rudimentary feature set and variations of this feature set are commonly used in the literature (for example Collins (2002), Lafferty et al. (2001), and Publications **V** and **VI**). Let W be the set of word forms in the training data. Additionally let P and S be the sets of prefixes and suffixes of maximal length 4 of all words $w \in W$. Then, the Ratnaparkhi feature set contains the unstructured feature templates in Table 6.1 and the structured feature templates in Table 6.2.

Feature template
The label of the previous word is y
The label of the previous two words are y' and y

Table 6.2: The set of structured feature templates introduced by Ratnaparkhi (1996)

As in the case of an HMM, the order of a CRF can be increased. This corresponds to including more label context in structured features. It is instructive to estimate the number of features when using a realistic training set. It is $|\mathcal{Y}|^3+|\mathcal{F}||\mathcal{Y}|$, where $\mathcal{Y}$ is the set of morphological labels and $\mathcal{F}$ is the set of unstructured feature templates.

For small label sets and large training data, the bulk of all features consists of unstructured features. However, for large label sets in the order of $1,000$ labels, there will be a significant number of structured features (one billion in this case). This necessitates either dropping second order structured features or using sparse feature representations. All structured features simply cannot be represented in memory. We will see techniques to circumvent these problems. Especially the averaged perceptron is essential.

It is common to represent the CRF using linear algebra. Each position t in the sentence is represented as a vector ϕ_t whose dimensions correspond to the entire space of possible features. The selection of features is finite because it is limited by the training data: there are only finitely many word forms, suffixes, morphological labels and so on in the training data. The elements of each vector ϕ_t represent activations of features. In the present, work all elements are either 0 or 1 mirroring the truth value of a feature at position t in the labeled sentence. Other activations in $\mathbb{R}$ can also be used when appropriate.

In order to represent sentence positions as vectors, we need an injective index function I which maps features onto consecutive integers starting at 1. For each feature f, $\mathrm{I}(f)$ will correspond to one dimension in ϕ_t. In a concrete implementation of a CRF tagger, features can be represented as strings and the index function I can be implemented as a hash function.

Given a sentence x and label sequence y, we can extract the set of features $F_t(x)$ for each position t in x. Let $\phi_t \in \mathbb{R}^N$ be a vector defined by

$$\phi_t(i) = 1, \text{ if } i \leq N \text{ and } I(f) = i \text{ for some } f \in F_t$$

all other entries in ϕ_t are 0.

Given a parameter vector $\theta \in \mathbb{R}^N$, the probability $p(y|x)$ is

$$p(y|x) \propto \prod_{t=1}^{T} \exp(\theta^\top \phi_t)$$

Specifically, the same parameter vector θ is shared by all sentence positions and the probability $p(y|x)$ is a log linear combination of parameter values in θ.

6.2 Logistic Regression

The *Logistic Regression Model* can be said to be the simplest instance of the conditional random field. It is an unstructured probabilistic discriminative model. In this section, I will present a formal treatment of the logistic regression model because it aids in understanding more general CRFs.

Regular linear regression models a *real valued quantity* y as a function of an input $x = (x_1, ..., x_n)$. In contrast, the logistic regression model models *the probability* that an observation x belongs to *a class* $y \in |\mathcal{Y}|$, where $\mathcal{Y}$ is a finite set of classes. For example, a logistic classifier can be used to model the probability of a tumor belonging to the class MALIGNANT or BENIGN. The probability is based on quantifiable information about the tumor such as its size, shape. These quantifiable sources of information are the *feature templates* of the logistic classifier and combined with class labels they constitute the features of the model.

The material at hand deals with linguistic data where most information sources are binary, for example

whether a word ends in suffix "-s" and whether a word is preceded by the word "an". In other domains such as medical diagnostics, more general features can be used. These can be real valued numerical measurements such as the diameter of a tumor. This treatment of logistic classifiers will assume binary feature activations. When using binary features, we can equate the example x with the set of feature templates $F_x \subset \mathcal{F}$ that it *activates*, that is **Tumor diameter** $\geq$ **5 cm**, **The previous word is "an"** and so on. Examples that activate the exactly same feature templates will be indistinguishable from the point of view of the Logistic Regression model.

The logistic classifier associates each combination of a feature template and class with a unique feature and a corresponding real valued parameter. Intuitively, the logistic classifier models correlations of feature templates and classes by changing the parameter values of the associated features. For example, it might associate the feature template **Tumor diameter** $\geq$ **5 cm** more strongly with the class MALIGNANT than the class BENIGN if large tumors are cancerous more often than smaller ones. This could be reflected in the parameter values of the model that correspond to the features $f =$ **Tumor diameter** $\geq$ **5 cm and class is MALIGNANT** and $f' =$ **Tumor diameter** $\geq$ **5 cm and class is BENIGN** so that the parameter value for f is greater than the parameter value for f'. In general parameter values, however, also depend on other features and feature correlations in the model. Therefore, we can say that the parameter value of f will be guaranteed to be greater than the parameter value of f' when **Tumor diameter** $\geq$ **5 cm** is the sole feature template and the model accurately reflects the original distribution of class labels among examples. In the general case, where there are several feature templates, this might fail to hold.

Formalizing the notation used in Section 6.1, let $\mathcal{F}$ be a finite set of feature templates and $\mathcal{Y}$ a finite set of classes. Each combination of feature template $f \in \mathcal{F}$ and class $y \in \mathcal{Y}$ corresponds to a unique feature. Therefore, the model will have $|\mathcal{F} \times \mathcal{Y}|$ features in total. Let $\theta \in \mathbb{R}^{|\mathcal{F} \times \mathcal{Y}|}$ be a real valued parameter vector and let I be a 1-to-1 index function which maps each feature onto and index of θ, that is $1 \leq \mathrm{I}(f, y) \leq |\mathcal{F} \times \mathcal{Y}|$.

For each example x, let $F_x \subset \mathcal{F}$ be the set of feature templates that x activates and let $y \in \mathcal{Y}$ be a class. Then the feature vector associated with x and y is $\phi(x, y) = \{0, 1\}^{|\mathcal{F} \times \mathcal{Y}|}$ defined by

$$\phi(x, y)[i] = \begin{cases} 1 & \text{iff } i = \mathrm{I}(f, y) \text{ for some } f \in F_x, \\ 0 & \text{otherwise.} \end{cases}$$

Now the conditional probability $p(y \mid x)$ defining the Logistic classifier is given by Equation (6.1). The equation defines a probability distribution over the set of classes $\mathcal{Y}$ because each quantity $p(y \mid x; \theta)$ is a positive real and the quantities sum to 1.

$$p(y \mid x; \theta) = \frac{\exp(\theta^\top \phi(x, y))}{\sum_{z \in Y} \exp(\theta^\top \phi(x, z))} \tag{6.1}$$

Inference Inference for a Logistic Regression Model corresponds to performing the maximization in Equation 6.2. As the equation demonstrates, the full computation of the probability is not required when

classifying. The maximization can be performed without normalization.

$$y_{max} = \arg\max_{y \in Y} p(y \mid x; \theta) = \arg\max_{y \in Y} \frac{\exp(\theta^\top \phi(x, y))}{\mathrm{Z}(x; \theta)} = \arg\max_{y \in Y} \exp(\theta^\top \phi(x, y)) \tag{6.2}$$

To avoid underflow when using finite precision real numbers (such as floating-point numbers), the maximization is usually rephrased as the minimization of a loss function in Equation 6.3 by applying a logarithmic transformation $x \mapsto -\log(x)$.

$$y_{min} = \arg\min_{y \in Y} -\theta^\top \phi(x, y) \tag{6.3}$$

From a practical implementation perspective, the minimization in Equation 6.3 boils down to computing one inner product $\theta^\top \phi(x, y)$ for each label $y \in Y$ and finding the minimum. Using a suitable sparse approach, each of the inner products can be computed in $\mathrm{O}(|F_x|)$ time, where F_x is the set of feature templates activated by example x. Therefore, the worst-case complexity of classification is dependent on the size of the label set Y and the number of feature templates $f \in F$, that is the complexity is $\mathrm{O}(|Y||F|)$.

Estimation The Logistic Regression Model is log-linear as demonstrated by Equation 6.4, which represents the model using a loss function $\mathcal{L}$.

$$\mathcal{L}(\theta; \mathcal{D}) = -\log p(y \mid x; \theta) = log(\mathrm{Z}(x; \theta)) - \theta^\top F_y(x) \tag{6.4}$$

Here $Z(x; \theta) = \sum_{z \in Y} \exp(\theta^\top F_z(x))$ is the partition function for example x.

Given labeled training data $\mathcal{D} = \{(x_1,\ y_1),\ ...,\ (x_n,\ y_n)\}$, there exist several options for estimating the parameters θ. The most commonly used is *maximum likelihood estimation*, which finds a parameter vector that minimizes the loss $\mathcal{L}$ on the training data $\mathcal{D}$ as shown in Equation (6.5).

$$\theta = \arg\min_{\theta'} \mathcal{L}(\theta; \mathcal{D}) = \arg\min_{\theta'} \sum_{(x,\ y) \in \mathcal{D}} \mathrm{Z}(x; \theta') - {\theta'}^\top F_y(x) \tag{6.5}$$

The probability $p(y \mid x; \theta)$ has exponential form, which means that the probability is proportional to a product of factors of the form e^{ap}, where a is an activation (0 or 1) and p is a parameter. This has three important consequences:

1. The function $\theta \mapsto p(y \mid x; \theta)$ is smooth.
2. The function $\theta \mapsto p(y \mid x; \theta)$ is convex.
3. There exists a *unique* θ maximizing the likelihood of the training data $\mathcal{D}$.[4]

Smoothness follows from the fact that each factor $a \mapsto e^{ap}$ is smooth and products and sums of smooth functions are smooth. Convexity of the likelihood follows by a straightforward application of the Hölder

[4]Technically this requires that the possible values of θ are limited into a compact subset of the parameter space.

inequality. Property 3 is a consequence of properties 1 and 2.

Although the maximization in Equation 6.1 cannot be solved exactly in general, the convexity and smoothness of $p(y \mid x; \theta)$ mean that efficient numerical methods can be used for approximating the maximum.

Gradient based methods such as SGD (leading to online estimation) and L-BFGS (leading to batch estimation) require information about the partial derivatives of the loss function. Therefore the partial derivatives $\partial\mathcal{L}(\theta; \mathcal{D})/\partial$i need to be computed. Examining Equation 6.5, we can see that the loss consists of two terms $f(\theta; \mathcal{D}) = \log(\mathrm{Z}(\mathcal{D}; \theta))$ and $g(\theta; \mathcal{D}) = \sum_{(x,y)\in\mathcal{D}} \theta^\top F_y(x)$. The partial derivative of g w.r.t. parameter i can be computed in the following way

$$\frac{\partial g}{\partial i} = \sum_{(x,y)\in\mathcal{D}} F_y(x)[i]$$

This quantity represents the total activation of feature i in the training data and is called *the observed count* of feature i.

Using the chain rule of derivatives, we get the partial derivative of f w.r.t. to param i

$$\frac{\partial f}{\partial i} = \sum_{(x,y)\in\mathcal{D}} \frac{\sum_{y'\in\mathcal{Y}} F_{y'}(x)[i] \exp(\theta^\top F_{y'}(x))}{\mathrm{Z}(x; \theta)} = \sum_{(x,y)\in\mathcal{D}} \sum_{y'\in\mathcal{Y}} F_{y'}(x)[i] p(y'|x; \theta)$$

This is the *expected count* of feature i which is the activation of feature i in the data set $x_1, ..., x_n$ predicted by the model given all possible label assignments.

Using the partial derivatives of the functions f and g, we see that the gradient of the loss function $\mathcal{L}$ is defined by

$$\nabla\mathcal{L}[i] = \sum_{(x,y)\in\mathcal{D}} \Big(\sum_{y'\in\mathcal{Y}} F_{y'}(x)[i] p(y'|x; \theta) \Big) - F_y(x)[i] \tag{6.6}$$

Equation 6.6 shows that the loss is zero when the expected and observed counts for each feature agree.[5] The properties for the logistic regression model discussed above guarantee that this there is at most one θ_{ML} like this and, when it exists, θ_{ML} is the maximum likelihood estimate for the parameters.

Regularization methods such as L_1 and L_2 introduced in Chapter 3 can also be applied to the model. This naturally changes the gradient and also the properties of the model. Analysis of the regularized model falls outside of the scope of this thesis.

[5] As each feature i is label specific in the current work, the sum $\sum_{y'\in\mathcal{Y}} F_{y'}(x)[i] p(y'|x; \theta)$ reduces to $F_y(x)[i] p(y|x; \theta)$. It is thus easy to see that the loss vanishes if $p(y|x; \theta) = 1$ for each $(x, y) \in \mathcal{D}$.

6.3 The Perceptron Classifier

The perceptron algorithm (Rosenblatt, 1958) is an alternative to maximum likelihood estimation for learning the weights of a discriminative classifier. As seen above, the logistic classifier optimizes the conditional probability of gold standard classes given training inputs. In contrast, the perceptron rule directly optimizes the classification performance of the discriminative classifier.

Intuitively, the multiclass perceptron algorithm works by labeling each training example in order using a current estimate of the parameter vector θ and adjusting the parameter vector whenever training examples are incorrectly labeled. Consequently, the perceptron algorithm is an online learning algorithm.

Inference Similarly as in the case of any linear classifier, the perceptron classifier scores each example x and class y by $\theta^\top \phi(x, y)$. Example x is labeled by the $y \in \mathcal{Y}$ which maximizes the score.

Estimation The perceptron algorithm is an error-driven online learning algorithm (see Algorithm 6.1 for an implementation in Python 3). When a classification error is encountered during estimation, that is

$$\theta^\top \phi(x, y) > \theta^\top \phi(x, y_{gold})$$

for some $y \neq y_{gold}$, the parameter vector θ is adjusted for relevant features. For every feature template f which is activated by the example x, the weights $\theta[I(f, y_{gold})]$ and $\theta[I(f, y)]$ are adjusted. Here $I(f, y_{gold})$ and $I(f, y)$ are the features corresponding to the template f and classes y_{gold} and y, respectively. The perceptron rule for weight adjustment is the following:

$$\theta[I(f, y_{gold})] = \theta[I(f, y_{gold})] + 1 \text{ and } \theta[I(f, y)] = \theta[I(f, y)] - 1$$

The perceptron adjustment does not guarantee that example x is correctly classified. However, it does guarantee that the score difference between the gold class and erroneous class decreases.[6] Given training data consisting of just one example, it is easy to see that the perceptron algorithm will eventually classify the example correctly. If there are more examples, it may however happen that a correct parameter vector is never found.

The perceptron algorithm *converges* when no example in the training data causes a change in the parameter vector θ. Equivalently, the perceptron algorithm correctly classifies every example in the training data. It can be showed that the perceptron algorithm converges whenever there exists a parameter vector that correctly classifies the training data (Freund and Schapire, 1999). Such a data set is called linearly separable. The term originates from a geometrical interpretation of the 2-class perceptron algorithm, where

[6] There are refinements of the perceptron algorithm, such as the passive-aggressive learning algorithm, which aim to make fewer updates by updating more aggressively when the difference in scores between the erroneous class and gold class is large (Crammer et al., 2006)

Algorithm 6.1: One pass of the perceptron algorithm in Python 3.

```
def infer(x, fextractor, theta, label_set)
    """
        x          - An observation.
        fextractor - A vector valued function.
                     len(fextractor(x,y)) == len(theta).
        theta      - A parameter vector.
        label_set  - Set of potential labels.
    """
    sys_label = None
    max_score = -float('inf')

    for y in label_set:
        score = dot_product(theta, fextractor(x,y))
        if score > max_score:
            max_score = score
            sys_label = label

    assert(sys_label != None)
    return sys_label

def perceptron(data, fextractor, theta, label_set):
    """
        data       - data[i][0] is an observation, data[i][1] a label.
        fextractor - A vector valued function.
                     len(fextractor(x,y)) == len(theta)
        theta      - The parameter vector.
        label_set  - Set of potential labels.

        Run one pass of the perceptron algorithm.
    """

    for x, y_gold in data:
         y_system = infer(x, fextractor, theta, label_set)

         if y_system != y_gold:
             for f in fextractor(x, y_system):
                 theta[f] -= 1
             for f in fextractor(x, y_gold):
                 theta[f] += 1
```

the parameter vector defines a hyper plane in the feature space which divides the space into two halves. A data set is called separable if there is a hyper space which separates the examples in each of the classes into their own half space.

When a data set is linearly separable, there are typically infinitely many parameter vectors that that classify the data set correctly. The perceptron algorithm will give one of these. Other algorithms exist which attempt to find an optimal parameter vector in some sense (e.g. Support Vector Machines (Cortes and Vapnik, 1995)). These, however, fall beyond the scope of the present work.

Even when the training set is not linearly separable, the perceptron algorithm will have good performance in practice. In the non-separable case, a held out development set is used. Training is stopped when the performance of the classifier on the development set no longer improves.

Voting and Averaging Because the perceptron algorithm makes fixed updates of size 1, the parameter vector tends to change too rapidly at the end of the training procedure. To avoid this, it is customary to use the average of all parameter vectors from the training procedure instead of the final parameter vector. This will give better performance during test time. Parameter averaging is an approximation of so called *voting perceptron*. In voting, each parameter vector is considered a separate classifier and the classification is performed by taking a majority vote of all of the classification results. This is impractical, because there are thousands of classifiers. Therefore, averaging is used in order to achieve approximately the same effect.

6.4 CRF – A Structured Logistic Classifier

This Section presents Linear Chain Conditional Random Fields (CRF).[7] Just as the HMM is a structured equivalent of the NB classifier, the CRF is the structured equivalent of the logistic regression model. Consequently, many of the algorithms required to run a CRF tagger are similar to the algorithms required to run an HMM tagger. Estimation of model parameters is, however, different because of the discriminative nature of the model.

Another major difference between an HMM classifier and a CRF classifier is that the CRF classifier typically employs a much larger set of features. This increases the size of the model. It also makes the discriminative tagger slow in comparison to the generative tagger. The slowdown is demonstrated by the experiments in Publication **VI**. However, the accuracy of the discriminative model in morphological tagging is superior to the generative HMM tagger.

Intuitively, the CRF model resembles a sequence of logistic regression classifiers with shared parameters. Given a sentence $x = (x_1, \ ..., \ x_T)$, label sequence $y = (y_1, \ ..., \ y_T)$ and parameters θ for the logistic regression model, a score $s(x, y_{t-2}, y_{t-1}, y_t, t; \theta)$ for label y_i in position t can be computed. Here the logistic regression model utilizes the input sentence x as well as labels y_{t-2}, y_{t-1} and y_t to extract

[7] More general CRF models can be formulated but these mostly fall beyond the scope of this thesis. See (Sutton and McCallum, 2012) for further details.

unstructured and structured features from the sentence and label sequence. The score s takes on a familiar form

$$s(x, y_{t-2}, y_{t-1}, y_t, t; \theta) = \exp(\theta^\top F_y(x, t, y_{t-2}, y_{t-1}))$$

where F_y is a vector valued feature extraction function. Each feature associated to the label y corresponds to one element of the vector $F_y(x, t, y_{t-2}, y_{t-1})$. Since F_y refers to a context spanning three labels, the model is a second order model. All discriminative taggers discussed in this Chapter are assumed to be second order models. As in the case of the logistic regression model, each entry of the vector can be an arbitrary real number but in this thesis they will always be either 0 or 1.

The probability of label sequence $y \in \mathcal{Y}^T$ given a sentence x of length T is

$$p(y|x; \theta) \propto \prod_{t=1}^{T} s(x, y_{t-2}, y_{t-1}, y_t, t; \theta) \tag{6.7}$$

In Equation 6.7, the labels y_{-1} and y_0 are special stop labels which do not belong to the label set $\mathcal{Y}$. The partition function of the sentence x is given by

$$\sum_{y \in \mathcal{Y}^T} \prod_{t=1}^{T} s(x, y_{t-2}, y_{t-1}, y_t, t; \theta) \tag{6.8}$$

It is noteworthy, that the probability in Equation 6.7 is normalized for *the entire sentence*, not for each position in isolation. A similar model, where normalization happens in each position is called the Maximum Entropy Markov Model (MEMM). It has been shown to give inferior performance in POS tagging of English (Lafferty et al., 2001).[8]

Inference Tagging of a sentence using the CRF model is very similar to HMM tagging. The major difference is that there are far more features in a CRF model which slows down inference compared to a typical HMM tagger. As in the case of an HMM, the Viterbi algorithm has to be used to find the MAP assignment of the label sequence because of the structured nature of the model. The forward-backward algorithm can be used to compute the marginal probabilities of labels.

Estimation Estimation of the CRF model parameters is more involved than the straightforward counting which is sufficient for HMM training. Estimation is instead very similar to estimation of the logistic regression model parameters. However, the structured nature of the CRF model complicates matters slightly.

[8] The inferior performance of the MEMM has been thought to be a result of the so called label bias problem (Lafferty et al., 2001) although *observation* bias may be more influential in POS tagging (Klein and Manning, 2002).

Let (x, y) be a labeled sentence. The observed count of feature i in position t is

$$F_y(x, t, y_{t-2}, y_{t-1}, y_t)[i]$$

and its expected count is

$$\sum_{y' \in \mathcal{Y}^T} F_y(x, t, y'_{t-2}, y'_{t-1}, y'_t)[i] p(y' \mid x; \theta).$$

The partial derivative w.r.t. to the loss $\mathcal{L}$ is the difference between the expected and observed counts as in the case of logistic regression.

$$\frac{\partial \mathcal{L}}{\partial i} = \sum_{t=1}^{T} \Big(\sum_{y' \in \mathcal{Y}^T} F_y(x, t, y'_{t-2}, y'_{t-1}, y'_t)[i] p(y' \mid x; \theta) \Big) - F_y(x, t, y_{t-2}, y_{t-1}, y_t)[i]$$

The quantities $\sum_{y' \in \mathcal{Y}^T} F_y(x_l, t, y'_{t-2}, y'_{t-1}, y'_t)[i] p(y' \mid x; \theta)$ have to be computed using the forward backward algorithm because a naive algorithm has too high computational cost. The need of the forward backward algorithm is the most important difference between logistic regression and CRF estimation. Commonly, the SGD algorithm and L-BFGS are used for the optimization of θ (Vishwanathan et al., 2006). As in the case of logistic regression model, the CRF model can also be regularized using for example L1 or L2 regularization (Sutton and McCallum, 2012).

Held out development data is commonly used to set the number of training epochs, model order and regularization hyper-parameters.

Alternative estimators In addition to the ML estimator, a variety of estimators are available for discriminative taggers. These alternative estimators are predominantly used because ML estimation can be quite resource intensive. A commonly used substitute for the ML estimator is the structured variant of the averaged perceptron algorithm described in Section 6.5 and taggers trained using the averaged perceptron algorithm are often called perceptron taggers (see for example Collins (2002)). Other examples mentioned by Sutton and McCallum (2012) are the pseudolikelihood Besag (1975) and piecewise pseudolikelihood estimators Sutton and McCallum (2007). Publication **IV** investigates a pseudolikelihood inspired variant of the perceptron estimator.

Given a training example $(x, y) \in \mathcal{D}$ where $x = x_1, ..., x_T$ and $y = y_1, ..., y_T$, the pseudolikelihood of y given x is given by equation 6.9. The notation $y_{\neg t}$ in Equation 6.9 refers to all the labels in y except label y_t.

$$\prod_{t=1}^{T} p(y_t | x, i, y_{\neg t}; \theta) \tag{6.9}$$

The complexity of optimizing the parameters θ with regard to pseudolikelihood is linear linear with regard to sentence length. In contrast, ML estimation which requires the forward-backward algorithm

is has quadratic complexity. This means that the pseudolikelihood estimator is substantially more efficient than ML estimation. However, it may result in poor accuracy as indicated by the experiments in Publication **IV**.

6.5 The Perceptron Tagger

Whereas the unstructured perceptron classifier presented in Section 6.3 is a discriminative classifier similar to the logistic regression model except that it uses perceptron estimation, a perceptron tagger (Collins, 2002) is a sequence labeling model similar to the CRF except that it uses perceptron estimation.

The model is formulated very similarly as the CRF model. It also uses a real valued parameter vector θ but the definition of the score of a label sequence y given sentence x is different

$$s(y|x;\theta) = \sum_{i=1}^{T} s(x, y_{i-2}, y_{i-1}, y_i, i; \theta) \tag{6.10}$$

The definition of the score s in an individual position t in Equation 6.10 is defined as $s(x, y_{i-2}, y_{i-1}, y_i, i; \theta) = \theta^\top F_y(x, i, y_{i-2}, y_{i-1})$, where F_y is the vector valued feature extraction function for label y. The definition of the perceptron model is simpler than the definition of the CRF model. The reason is that the perceptron model is not intended for defining a probability distribution among label sequence. It can only be used for determining the best label sequence with regard to a sentence x.

Inference The Perceptron tagger uses the Viterbi algorithm for exact inference. Beam search can be used for faster approximate inference together with a label dictionary as shown in Publication **VI**. Chapter 8 presents experiments on varying beam widths.

Estimation Whereas, CRF estimation requires the forward-backward algorithm, perceptron estimation only requires the Viterbi algorithm. Exactly as in the case of the unstructured perceptron algorithm, each training example (i.e. sentence) is labeled and unstructured and structured features are updated accordingly. The number of training epochs is determined using held-out data. As in the case of the unstructured perceptron algorithm, parameter averaging is useful for improving the accuracy of the percetron tagger.

Besides the standard perceptron algorithm, there are many approximative perceptron variants. Publication **IV** introduces the pseudopeceptron and piece-wise pseudoperceptron estimators which are inspired by the pseudolikelihood and piecewise pseudolikelihood estimators for the CRF model. In the spirit of pseudolikelihood, the pseudoperceptron maximizes the score of each label in isolation as shown by equa-

tion 6.11. The remaining labels in the sentence are fixed to their gold standard values.

$$s(y|x, y_{gold}; \theta) = \sum_{t=1}^{T} s(y|x, y_{gold,\neg t}; \theta). \tag{6.11}$$

Beam search for estimation A high model order and large label set can result in a prohibitive runtime for the Viterbi algorithm. It has a time complexity that is dependent on the $n+1$st power of the label set size for an nth order model. This problem can be avoided using beam search during estimation instead of the Viterbi algorithm. This reduces the complexity to $O(T|\mathcal{Y}|b$, where T is the size of the training data, $\mathcal{Y}$ the label set and b is the beam width.

Because beam search is an approximative inference algorithm, it may however not give the correct MAP assignment for a sentence. It may happen that beam search returns a label sequence y_{sys} for training sentence x whose score with regard to the current model parameters is lower than the score of the gold label sequence y_{gold}. This leads to perceptron updates which are not necessary because the model would already correctly label sentence x, if only exact inference were used. In some domains such as syntactic parsing, this leads to significant reduction in classification performance (Huang et al., 2012)

Violation fixing To avoid superfluous perceptron updates, a technique called violation fixing can be used (Huang et al., 2012) (this is an extension of the early update technique suggested for incremental parsing in Collins and Roark (2004)). Huang et al. (2012) suggest several related violation fixing methods. The essence of the methods is to compute the score for each prefix of the label sequence y_{sys} returned by beam search and each prefix of the gold standard label sequence y_{gold}. One prefix length i is then selected so that the score of $y_{sys}[1:i]$ is higher than $y_{gold}[1:i]$.[9] Updates are then performed for $y_{sys}[1:i]$ and $y_{gold}[1:i]$. The choice of i depends on the violation fixing method. For example, the maximum violation criterion corresponds to choosing an i which maximizes the score difference between $y_{sys}[1:i]$ and $y_{gold}[1:i]$.

In the experiments presented in Publication **VI**, violation fixing did not result in consistent statistically significant improvements. It may be that violation fixing is more influential in parsing than POS tagging or morphological tagging.

- Hierarchical CRF Müller et al. (2013), Weiss and Taskar (2010) and Charniak and Johnson (2005).

Cascaded Model Architecture Beam search can be used to speed up estimation for the perceptron tagger. For large label sets, even beam search may not lead to sufficient run-time optimization because the complexity of beam search is dependent on the label set size. Publication **VI** introduces a method to optimize estimation even further by only considering a subset of $\mathcal{Y}$ in every position in the training data.

The approach is an application of the idea of structured prediction cascaded presented by Weiss and Taskar (2010). They propose to use a cascade of increasingly complex models. Each model prunes the

[9] If such an i does not exist, then the score for y_{sys} is in fact higher than the score of y_{gold}.

label candidates available at a position in the input data. Subsequent models only consider those labels that were not pruned by an earlier model in the cascade. This leads to accelerated inference and estimation. Müller et al. (2013) applied the idea of structured prediction cascades to CRF models. Their results suggest that this can lead to both accelerated estimation and improved tagging results.

As in the case of the CRF model, a cascaded model structure can be used to shorten training time of a perceptron tagger as shown in Publication **VI**. Instead of a cascade of discriminative classifiers, used by Müller et al. (2013), the system presented in Publication **VI** uses a combination of a generative label guesser of the type presented in Section 4.4 and a structured perceptron tagger. The number of guesses can be determined either based on a probability mass threshold or using a fixed number of guesses per word. Setting the threshold too low will result in a training task that is to easy. Consequently the model will over-fit the training data. Higher thresholds will approximate the original training task more closely but will also lead to longer training times.

Like beam search, label guessing modifies the training task. It does, however, not require violation fixing because it does not influence the relative difference in scores of label sequences as long as the gold standard label sequence is never pruned out. Therefore, the gold standard label should always be added to the set of guesses given by the label guesser.

The combination of beam search and model cascading results in a fast training with a tolerable decrease in tagging accuracy even for large label sets and for second order models as shown by the experiments in Chapter 8.

An early approach that resembles the structured cascade approach is tiered tagging used by Tufis (1999) and Ceausu (2006). In tiered tagging, the label set is first projected onto a smaller set of coarse labels. A tagger is first used for labeling text with coarse labels. Subsequently another tagger is used to convert coarse labels into full morphological labels. The structured prediction cascades seem to deliver superior results as indicated in Publication **V**, however, direct comparison is difficult because Ceausu (2006) uses a MEMM tagger and Publication **V** uses a perceptron tagger. However, this seems plausible because both tagging with coarse labels and subsequent recovery of fine-grained labels represent potential error sources.

6.6 Enriching the Structured Model

As seen above, the feature set of a discriminative classifier with $|\mathcal{Y}|$ classes and $|F|$ feature templates has on the order of $|\mathcal{Y} \times F|+|\mathcal{Y}|^n$ features and parameters, where n is the model order. When $\mathcal{Y}$ is large, this gives rise to data sparsity because each feature template is seen with quite few labels $y \in \mathcal{Y}$. Large morphological label sets, however, are typically internally structured. For example, the noun label noun+sg+nom and adjective label adj+sg+nom share the same number sg and case nom.

Often data sparsity is combated by introducing more abstract features that will be activate more often than the existing features. In the case of large structured feature sets, a natural choice is to extract features for the components of morphological labels as well as for the entire labels. I will call such features *sub-*

label features.

For the label noun+sg+nom, the features which are extracted is by a standard CRF are given by

$$F_{\text{noun+sg+nom}}(x, t, y_{t-2}, y_{t-1})$$

When using sub-label features, we additionally extract features for the components of noun+sg+nom

$$\sum_{y \in \{\text{noun+sg+nom, noun, sg, nom}\}} F_y(x, t, y_{t-2}, y_{t-1})$$

The features extraction function also utilizes the internal structure of the argument labels y_{t-1} and y_{t-2}. Examples of sub-label features include **a word dog is singular** and **a singular word follows another singular word**. It is, however, best to do this in a restricted manner. The experiments in Chapter 8 demonstrate that in a second order CRF tagger, structured sub-label features of order one result in consistent improvement in accuracy but higher order sub-label features give no improvement.

Sub-label features can help combat data sparsity. Additionally, they are useful for modeling linguistic phenomena such as congruence. For example, two adjacent singular words will activate the set of features for the singular sub-label regardless of the main POS of the words.

In restricted form, sub-label features have been utilized by for example Müller et al. (2013). They use sub-labels exclusively for unstructured features. While sub-labels seem to be most beneficial in combination with unstructured features in a morphological tagging setting where a morphological analyzer is not utilized, this is not the case in a morphological disambiguation setting as the experiments in Chapter 8 indicate. When using a morphological analyzer, sub-labels result in the greatest improvement when combined with structured features.

Smith et al. (2005) also utilize structured sub-labels, however, only in a restricted way. They build separate structured chains modeling the sequence of main POS classes, cases, numbers, genders and lemmas of neighboring words. Due to the lack of cross-dependencies between different grammatical categories, is doubtful that their system could model phenomena like the dependence between a verb and the case of its object.

Spoustová et al. (2009) also use a linguistically motivated selection of unstructured and structured sub-labels (at least for main part-of-speech and case of nominals) for tagging Czech, however, it is difficult to establish exactly what kind of features they use because this is not documented in a detailed manner in Spoustová et al. (2009).

Publication **V** extends this approach to fully take into account structured features. As the experiments in Chapter 8, structured sub-labels have a substantial impact on tagging accuracy both in morphological tagging setting without a morphological analyzer and in morphological disambiguation setting. For Finnish, the impact of sub-label features on tagging accuracy seems to be on par with the impact of higher model order.

6.7 Model Pruning

Discriminative models for morhpological tagging can often grow quite large in terms of parameter count. For example, the model learned from the FTB corpus by the FinnPos tagger has more than 4 million parameters.

A large number of parameters is problematic because it causes over-fitting of the model to the training data. Moreover, large models can be problematic when memory foot-print is an issue: e.g. on mobile devices.

Different methods have been proposed for pruning of perceptron models. Goldberg and Elhadad (2011) prune the models based on update count. Parameters that receive less than a fixed amount of updates during training will be omitted from the final model. Another approach is to prune by feature count. For example Hulden et al. (2013) prune out features for words occurring less than a fixed amount of times in the training data. More generally, features that are activated less than a fixed amount of times may be pruned out.

Some regularization techniques can also be used to learn sparse perceptron models. L1-regularization yields sparse models similarly as for logistic regression. Zhang et al. (2014) investigate L1-regularization for structured perceptron. They gain accuracy but do not report results on model size.

I have explored two different pruning strategies

- Pruning based on update counts (Goldberg and Elhadad, 2011).
- Pruning based on parameter value.

Goldberg and Elhadad (2011) show that update count based pruning beats feature count based pruning in dependency parsing and POS tagging. Therefore, I decided not to compare those approaches. Instead, I compare update count based pruning to pruning based on final parameter value.

Update Count Pruning When using this strategy, each parameter which did not receive at least n updates during training, is omitted from the final model. Here n is a hyper-parameter which is set using held-out data. In practice, this pruning strategy requires that one maintains a update count vector where each element corresponds to one model parameter. Whenever a parameter is updated during training, the update count is increased.

As stated before, the perceptron algorithm labels a training example and then performs updates on the model parameters. When labeling during training, only those parameters that already received at least n updates are used. However, updates are performed on all parameters. When the update count of a parameter exceeds n, the parameter value will therefore already be of similar magnitude with the rest of the parameter values in the model. This speeds up estimation as Goldberg and Elhadad (2011) note.

Goldberg and Elhadad (2011) do not explore pruning in an early stopping scenario. My preliminary experiments showed that it is best to first set the number of training passes without parameter pruning and then set the pruning threshold n separately using development data. If the number of passes and the update

count threshold are set at the same time, the model parameters converge quite slowly resulting in many training epochs and consequently many parameter updates. This has an adverse effect on the number of parameters that can be pruned from the final model without resulting in improved accuracy.

Value Based Pruning A very simple strategy for parameter pruning is to prune based on the parameter value. The model is trained in the regular manner. After training, all parameters whose absolute value does not exceed a threshold κ are omitted from the model. Remaining parameter values remain unchanged. The hyper-parameter κ is determined using a development set.

In the experiment chapter, I show that value based pruning outperforms update count based pruning on the data-sets that I have used. In some settings, the difference is substantial.

6.8 Summary of Publications IV, V and VI

Publication IV A central problem training perceptron and CRF taggers for morphologically complex languages is that the time complexity of the Viterbi algorithm is dependent on the $n+1$st order of the label set size, when training an nth order tagger. When the label set is large, this results in inconvenient training times even in the case of first order taggers. Publication **IV** presents two novel variants of the perceptron algorithm which are inspired by the pseudo-likelihood and piecewise pseudo-likelihood criteria presented in Section 6.4.

The new estimators, pseudo-perceptron and piecewise pseudo-perceptron are shown to be competitive with greedy decoding and passive aggressive training with regard to accuracy. Moreover, it delivers substantially shorter training times in presence of large label sets. The training time is, however, still influenced by the label set size because the time complexity of pseudo-perceptron and piecewise pseudo-perceptron is linear with regard to label set size.

Publication V This publication investigates sub-label dependencies in morphological tagging of five languages: English, Romanian, Estonian, Czech and Finnish. The experiments show that sub-label dependencies yield statistically significant improvements for Estonian, Czech and Finnish. Moreover, the experiments indicate that addition of sub-label dependencies to a first order model results in greater improvement in tagging accuracy than going from a first order to a second order model.

Publication VI This paper describes FinnPos, an open-source morphological tagging and lemmatization toolkit for Finnish. The morphological tagging model is based on the averaged structured perceptron classifier. Given training data, new taggers are estimated in a computationally efficient manner using a combination of beam search and model cascade. The lemmatization is performed employing a combination of a rule-based morphological analyzer, OMorFi, and a data-driven lemmatization model.

The toolkit is readily applicable for tagging and lemmatization of running text with models learned from the recently published Finnish Turku Dependency Treebank and FinnTreeBank. Empirical evalua-

tion on these corpora shows that FinnPos performs favorably compared to reference systems in terms of tagging and lemmatization accuracy. In addition, we demonstrate that our system is highly competitive with regard to computational efficiency of learning new models and assigning analyses to novel sentences.

Chapter 7

Data-Driven Lemmatization

In this section, I will present the task of data driven lemmatization. I will examine different approaches to data driven lemmatization and present the lemmatizer used in the FinnPos toolkit presented in Publication **VI**.

A lemmatizer is a system that takes text as input and returns the lemmas of words in the text. Because dictionaries and other lexical resources often list lemmas but omit other word forms, lemmatizers are useful for example for information extraction. They are particularly useful for morphologically complex languages where a substantial part of words occurring in text undergo various inflection.

Lexical resources such as dictionaries or morphological analyzers are very helpful for the lemmatization task. In fact, lemmatization is often seen as one of the sub tasks of morphological analysis. Another task which is closely related to lemmatization is *morphological paradigm induction* (Ahlberg et al., 2014). Here the task is to generate all, or a selection, of the inflectional forms of a word form. Therefore, lemmatization can also be seen as a sub-task of morphological paradigm induction.

Word	Label	Translation	Lemma
kissa	noun+sg+nom	a/the cat	kissa
sanoessa	verb+act+inf+ine	while saying (something)	sanoa
talossa	noun+sg+ine	in a/the house	talo

Table 7.1: Lemmatization of three Finnish word forms ending “-ssa”.

I will treat lemmatization as a follow-up task of morphological labeling. Therefore, the lemmatizer has access to the morphological labels of the words in the text. The morphological label provides very useful information for lemmatization because it can help to disambiguate between candidate lemmas. As an example, consider the three Finnish word form ending “-ssa” in Table 7.1. The different morphological

analyses correspond to different ways of forming the lemma. For example in the case of a singular inessive of a noun (“talossa”), the lemma (“talo”) is formed by removing the suffix “-ssa”. If the word form is instead a nominative, the lemma is identical to the word form. As the example shows, the morphological label can help to rule out incorrect lemmas.

A morphological analyzer can be used for lemmatization of a morphologically tagged text. First, analyze each word using the morphological analyzer. This produces a set of morphological labels and associated lemmas. Then simply pick the lemma which is associated with the correct morphological label. As long as the morphological label assigned to each word is also known by the morphological analyzer, this works perfectly. Problems arise when word forms are not recognized by the morphological analyzer or when words are assigned morphological labels not recognized by the analyzer. There are several approaches to solving these problems. One approach is to utilize the morphological analyzer (for example a finite-state analyzer) to produce a guess for a lemma even though the word form is not recognized. The guess is based on orthographically similar words which are recognized and can be lemmatized by the morphological analyzer. As an example of this approach, see Lindén (2009) who use finite-state algebra to transform a morphological analyzer into a morphological guesser that can generate the lemma for words that are not recognized by the original analyzer.

The main advantage of basing a data driven lemmatizer on an existing morphological analyzer is that large coverage morphological analyzers model most, if not all, morphotactic and the morphophonological phenomena that occur in a language. Therefore, it is likely that the analyzer recognizes a number of similar words in the inflectional paradigm of an unknown word even though it would not recognize that specific word form which can be utilized in lemmatization.

Most existing work on analyzer based lemmatizers has used rather simple statistical models. For example, Lindén (2009) uses plain suffix frequencies.

7.1 Framing Lemmatization As Classification

In contrast to lemmatizers based on morphological analyzers, classifier based lemmatizers Chrupala et al. (2008) are learned from data without an existing model. The general approach is based on the observation that word forms can be transformed into lemmas using an *edit script*. For example, the English noun “dogs” has the lemma “dog”. To convert “dogs” into “dog” one needs to remove the suffix “s”. This is a very simple example of an edit script which I will denote $[-s \rightarrow \varepsilon]$. Classifier based lemmatizers frame the lemmatization task as a as classification task. As labels, the classifier uses edit scripts. Subsequently to labeling a word form with an edit script class, the lemmatizer will apply the edit script thus constructing a lemma.

The main advantage of using a classifier based lemmatizer is that the classifier can use a feature based discriminative model. In contrast to analyzer based lemmatizers, classifier based lemmatizers can therefore use richer information sources such as prefixes and word shapes expressed as regular expressions

[1] – not exclusively information about word suffixes.

Although it would be very interesting to combine these approaches, it falls beyond the scope of this thesis. Therefore, I have used classifier based lemmatizers. I decided upon classifier based lemmatizers partly because the work of Lindén (2009) already investigates analyzer based lemmatization for Finnish. When performing morphological disambiguation based on the output of a morphological analyzer, the current system does use the morphological analyzer for lemmatization of all word forms which are recognized by the analyzer. For all remaining words, the data driven lemmatizer is used.

In the field of morphological paradigm generation, there exists work which in a sense combines the analyzer and classifier based approaches, for example (Hulden et al., 2014). However, their starting point is not a morphological analyzer. Instead a list of morphological paradigms is used. It would be interesting to explore this but it falls beyond the scope of the current work. Another interesting direction for future research is joint tagging and lemmatization which has yielded improvements both in tagging and lemmatization accuracy (Müller et al., 2015). However, this also falls beyond the scope of the current work.

7.2 Lemmatization in FinnPos

The classification based lemmatizer in the FinnPos toolkit reads an input word, identifies the set of edit scripts that can be applied to the input and scores the candidate scripts using the input form, and its morphological label. The score is given by a feature based classifier. Finally, the edit script that receives the highest score is applied on the input form and the lemma is recovered.

Extracting Edit Scripts Given a word form such as “dogs” and its lemma “dog”, several edit scripts can be extracted. For example, $[-s \rightarrow \varepsilon]$, $[-gs \rightarrow -g]$, $[-ogs \rightarrow -og]$. The current system extracts the shortest script which adequately recovers the lemma.

The FinnPos system only extracts edit scripts which delete a suffix and appends another suffix such as the script $[-s \rightarrow \varepsilon]$. This is sufficient for Finnish where all words except numerals exclusively exhibit inflection at the end of words. Naturally, this would not be sufficient in general. More general edit scripts are therefore applied with other languages, for example by Chrupala et al. (2008).

For morphologically complex languages, a large number of edit scripts may be extracted from training data. For example, the Finnpos system extracts 4835 different edit scripts for the 145953 tokens in the training and development data of FinnTreeBank. Therefore, many of the classes occur only a few times in the training data. This leads to data sparsity. However, increasing the amount of the training data would probably alleviate the problem significantly because the inventory of inflectional paradigms, and consequently edit scripts, is finite.

[1] An example of a word shape expressions in POSIX syntax is `[A-Z][a-z]+` which matches capitalized English words.

Features for Lemmatization For a word $w = (w_1 ... w_n)$ and a morphological label y, the lemmatizer in the FinnPos system currently uses the following feature templates:

- The word form w.
- The morphological label y.
- Suffixes (w_n), (w_{n-1}, w_n), ... Up to length 10.
- Prefixes (w_1), (w_1, w_2), ... Up to length 10.
- Infixes (w_{n-2}, w_{n-1}), (w_{n-3}, w_{n-2}) and (w_{n-4}, w_{n-3}).

For each feature template f (except the morphological label template y), FinnPos additionally uses a combination template (f, y) which captures correlations between morphological labels and the orthographical representation of the word form.

The infix templates are useful because they model the environment where an inflectional suffix (such as "-ssa") is removed and a lemma suffix is added. They aim at preventing phonotactically impossible combinations.

Estimation The lemmatizer can be implemented using any discriminative classifier. For example as an averaged perceptron classifier or a logistic classifier. In the FinnPos system, the lemmatizer is an averaged perceptron classifier.

The estimation of the lemmatizer model differs slightly from standard averaged perceptron estimation presented in Chapter 3. Even though the number of edit scripts can be very large (in the order of thousands), the subset of edit scripts applicable for any given word form is much smaller. Moreover, it is always known in advance because it is completely determined by the suffixes of the word form. Therefore, the classifier is only trained to disambiguate between the possible edit scripts associated to each word form. This speeds up estimation considerably.

Inference In the FinnPos system, words which were seen during training time, are lemmatized based on a lemma dictionary which associates each pair of word form and morphological label with a lemma. For words which were not seen during training or which received a label not seen during training, are lemmatized using the data driven lemmatizer. Additionally, a morphological analyzer can be used to assign lemmas to those words which it recognizes.

For word forms which cannot be lemmatized using the lemma dictionary or morphological analyzer, the data driven lemmatizer is used. For each word form, the set of applicable edit scripts is formed and scored. The highest scoring edit script is subsequently applied to the word form to produce a lemma.

Chapter 8

Experiments

Publication **VI** presents FinnPos, which is a discriminative morphological tagger based on the averaged perceptron classifier. It is specifically geared toward morphologically complex languages. To this end, it implements beam search and a cascaded model architecture for speeding up estimation and tagging speed. To improve accuracy, it includes an option to use a morphological analyzer during tagging. Moreover, it extracts sub-label dependencies for unstructured and structured features in order to improve tagging accuracy in presence of large structured label sets.

This chapter presents experiments on morphological tagging in Finnish using FinnPos. These experiments are intended to augment the experiments in Publication **VI**. I will present experiments on the following themes

- **Cascaded Model Architecture** What is the effect of different settings for the label guesser on tagging accuracy, estimation speed and tagging speed?
- **Beam Search** What is the effect of beam width on tagging accuracy, estimation speed and tagging speed?
- **Model Order** What is the effect of model order on tagging accuracy, estimation speed and tagging speed? Experiments are presented both when tagging with and without a morphological analyzer.
- **Sub-Label Order** What is the effect of the order of sub-label dependencies on tagging accuracy, estimation speed and tagging speed? Experiments are presented both when tagging with and without a morphological analyzer.
- **Model Pruning** Which is the better model pruning strategy: update count based or value based pruning?

8.1 Data Sets

All experiments are conducted on both the Turku Dependecy Treebank (Haverinen et al., 2014) (TDT) and FinnTreeBank (Voutilainen, 2011) (FTB). Two distinct data sets, containing text from different genres, are used in order to justify general claims about the performance of the system in Finnish morphological tagging. Table 8.1 gives a numerical overview of both data sets.

	TDT	FTB
Sentence Count	13,572	19,121
Token Count	183,118	162,028
Label Count	2,014	1,399
OMorFi Coverage	94.2%	99.0%

Table 8.1: Summary of Turku Dependency Treebank (TDT) (Haverinen et al., 2014) and FinnTreeBank (FTB) (Voutilainen, 2011). The OMorFi coverage refers to coverage per token.

FinnTreeBank FTB is a morphologically tagged and dependency parsed collection of example sentences from Iso Suomen Kielioppi, a descriptive grammar of the Finnish language (Hakulinen et al., 2004). The examples have been harvested from newspapers, novels and blogs. Additionally, some examples represent spoken language.

Each sentence in the FTB corpus has been selected to illustrate some grammatical phenomenon. Therefore, it is safe to say that FTB does not represent a random sample of Finnish text. It probably contains a high number of rare grammatical phenomena. This can be seen as a weakness because results on the FTB corpus may not carry over to other data sets. However, it also makes the corpus interesting and challenging from the point of view of morphological tagging because the data is expected to be complex.

Both the morphological tagging and dependency structures of FTB have been manually prepared. The morphological analyses of word tokens are post-processed outputs of OMorFi, an open-source morphological analyzer for Finnish (Pirinen, 2011).

Turku Dependency Treebank TDT contains text from ten different domains, for example Wikipedia articles, blog entries, and financial news. The annotation has been prepared by manually correcting the output of an automatic annotation process. Similarly to FTB, the morphological analyses of word tokens in FTB are post-processed outputs of OMorFi (Pirinen, 2011). However, the treebanks are based on different versions of OMorFi. Moreover, the post-processing steps applied in TDT and FTB differ. This results in somewhat different annotation schemes. The TDT annotation for each word token consists of word lemma (base form), part-of-speech (POS), and a list of detailed morphological information, including case, number, tense, and so forth.

Data Splits Widely used data sets usually have established splits into training, development and test data. For example, most work on English POS tagging reports results on the Penn Treebank corpus (Marcus et al., 1993) using the data splits introduced by Collins (2002) (Sections 1-18 for training, 19-21 for development and 22-24 for testing). This is sound because it guarantees that results reported in different papers are comparable. For the data sets used in this thesis, FTB and TDT, there are no established splits. Therefore, the experiments use 80% of the data for training, 10% as development data and 10% as final test data. The data is split in the following way: For each consecutive ten sentences, the first eight are assigned to the training set, the ninth one to the development set and the tenth one to the test set.

8.2 Setup

Exhaustive experiments on the effect and interactions of the different hyper-parameters used by the FinnPos tagger would require hundreds of experiments because FinnPos incorporates a variety of optimizations to accuracy and speed governed by different hyper-parameters.[1] I did not deem this feasible in practice. Instead I have chosen the settings presented in Publication **VI** as vantage point and examined the impact of changing one hyper-parameter at a time. These settings were chosen because they give state-of-the-art accuracy as presented in the paper.

The basic setting for all experiments is

- A second order model.
- First order sub-label dependencies.
- 99.9% mass for the generative label guesser.
- 99.9% mass for the adaptive beam search.

This setting is varied with respect to the hyper-parameter that is being investigated. All experiments are run on a desktop computer (Intel Core i5-4300U with 1.90 GHz and 16 GB of memory).

In all experiments, hyper parameters are first set using development data. Then the development data and training data are combined and this data set is used to train the final model, which is then evaluated using the test set. Training times refer to training the final model. Thus they do not contain the cost of development. This setup was used because it is also used in Publications **IV**, **V** and **VI**.

A morphological analyzer is used when specifically indicated. A label dictionary is used in all experiments both to speed up decoding and improve accuracy. The label dictionary is also used when a morphological analyzer is used in tagging. The reason for this is that a liberal compounding and derivation mechanisms (such as the ones implemented in the OMorFi morphological analyzer) can result in a

[1] Experiments should vary beam width, the settings of the label guesser, model order and order of sub-label dependencies. Moreover, experiments should be conducted using a morphological analyzer and without one. Assuming 10 different settings for beam, and guesser and three settings for model order and sub-label order, this gives $10^2 \cdot 3^2 \cdot 2 = 1800$ distinct experiments.

number of unlikely analyses for longer word forms. Analyses that have been attested in the training corpus should, therefore, be preferred when possible.

The feature set used in the experiments follows Publication **VI**. Let $x = (x_1 ... x_T)$ be a sentence, $y = (y_1 ... y_T)$ a label sequence and t and index. Then the unstructured features templates for the morphological tagger are the familiar Ratnaparkhi features (Ratnaparkhi, 1998) augmented with a few additional features. For all words, FinnPos uses the following feature templates

- The word form x_t and the lower cased version of x_t.
- The length $|x_t|$ of word form x_t.
- Each word form in a four word window around t: x_{t-2}, x_{t-1}, x_{t+1} and x_{t+2}.
- The word form bigrams (x_{t-1}, x_t) and (x_t, x_{t+1}).

For rare words[2], it additionally extracts the following features

- `DIGIT` when x_t contains a digit.
- `UC` when x_t contains an upper case letter.
- Prefixes and suffixes of x_t up to length 10.

When a morphological analyzer is used, each morphological label given to word x_t is also used as a feature template.

Some baseline runs are impossible to run. FinnPos uses a second order model. With a label set size of 1,000, inference using the plain Viterbi algorithm is prohibitively slow because the time complexity of the algorithm is dependent on the third order of the label set size. Therefore, it was not feasible to run experiments without label pruning during training.

Finally, the results of the experiments presented here differ minutely from the results presented in Publication **VI** due to added features (lower cased word form and word length) and a few bug fixes that have improved results.

8.3 Using a Cascaded Model

This section presents experiments on using different settings for the generative label guesser included as a pre-pruning step during training as explained in Section 6.5.

[2]A rare word is one that is not common. The list of common words is user defined but in these experiments I have defined common words to be words having frequency 10 or higher in the training corpus

ADAPTIVE GUESS COUNT

Guess Mass	Tagging Accuracy (%)	Training time	Dec. Speed (KTok/s)
0.9	93.21 (OOV: 77.68)	3 min, 3 epochs	7
0.99	93.11 (OOV: 77.14)	3 min, 3 epochs	7
0.999	93.23 (OOV: 78.49)	4 min, 4 epochs	8
0.9999	93.41 (OOV: 78.55)	2 min, 2 epochs	7

FIXED GUESS COUNT

Guess Count	Tagging Accuracy (%)	Training time (min)	Dec. Speed (KTok/s)
1	91.48 (OOV: 69.81)	1 min, 3 epochs	8
10	93.23 (OOV: 77.56)	2 min, 2 epochs	7
20	93.18 (OOV: 77.89)	3 min, 3 epochs	7
30	93.22 (OOV: 77.62)	4 min, 3 epochs	6
40	93.43 (OOV: 78.49)	4 min, 2 epochs	5

Table 8.2: Different label guesser settings for FinnTreeBank

ADAPTIVE GUESS COUNT

Guess Mass	Tagging Accuracy (%)	Training time (min)	Dec. Speed (KTok/s)
0.9	92.69 (OOV: 74.10)	8 min, 8 epochs	5
0.99	92.66 (OOV: 74.13)	9 min, 8 epochs	5
0.999	92.76 (OOV: 74.65)	8 min, 7 epochs	5
0.9999	92.75 (OOV: 74.30)	6 min, 5 epochs	5

FIXED GUESS COUNT

Guess Count	Tagging Accuracy (%)	Training time (min)	Dec. Speed (KTok/s)
1	89.85 (OOV: 61.33)	2 min, 5 epochs	6
10	92.35 (OOV: 72.55)	5 min, 7 epochs	6
20	92.61 (OOV: 73.88)	5 min, 4 epochs	6
30	92.81 (OOV: 74.87)	12 min, 9 epochs	5
40	92.91 (OOV: 75.35)	14 min, 9 epochs	5

Table 8.3: Different label guesser settings for Turku Dependency Treebank

Setup For both TDT and FTB, I present results for using a fixed amount of label guesses and for choosing a varying amount of guesses per word based on probability mass. Because of the prohibitive runtime and memory requirements, it was not possible to run experiments without any form of label pruning during training. The most important point of these experiments is that using some kind of label guesser during training is almost necessary if one wants to train a second order model for label sets of several hundreds

or thousands of labels.

Results As Table 8.3 demonstrates, using a larger amount of label guesses improves accuracy in general. When using a fixed number of guesses for every word, the accuracy levels off at 40 guesses per word for both FTB and TDT.

When pruning label candidates based on probability mass, the results differ between the treebanks. For FTB, the mass 0.9999 yields approximately the same accuracy than as 40 guesses. For TDT, however, 40 label guesses result in 0.2%-points better accuracy than using the mass 0.9999. The difference is substantial.

The training time per epoch is clearly related to the amount of label guesses, however, the number of epochs seems to fluctuate somewhat from two to four for FTB and from five to eleven for TDT. Therefore, it is difficult to see a clear trend for the total training time. It is also not possible to say that mass based pruning always leads to a faster training time when compared to a fixed number of guesses which yield similar accuracy.

The amount of label guesses influences decoding speed to some degree because the same setting is used for the label guesser during decoding. Because the label guesser is only used for OOV words, the exact setting of the guesser does however only have a moderate impact.

Discussion Whereas training a second order tagger for a label set exceeding a thousand labels requires a prohibitive amount of computational resources when estimation and inference utilize the standard Viterbi algorithm or even a beam search, the experiments in this section demonstrate that a cascaded model architecture allows for training second order models rather fast. The memory requirement was moderate as it did not exceed the 16 GB available on the author's computer.

It is not clear whether pruning based on probability mass is superior to pruning based on a fixed amount of guesses. It is concerning that the results on the TDT data set for the mass 0.9999 were clearly inferior to the result when using 40 guesses for each word. Increasing the the probability mass also did not seem to improve the results further. A reason for this may be that the TDT corpus is larger than the FTB corpus. The generative guesser may suffer from over-confident probability estimates when the amount of training data is increased and it may be quite difficult to set the threshold for the probability mass. A discriminative guesser could be used instead, but its training time would exceed that of the generative guesser. This would reduce overall performance. An alternative might be to decreases the amount of training data for the guesser, but this requires further experiments.

8.4 Beam Search

This subsection presents experiments using different beam settings during training and decoding. Experiments are presented both for fixed beam widths and adaptive beam, which is described in Section 6.5.

Setup I compare fixed beam width to an adaptive beam presented in Section 6.5. Additionally, I include training and decoding results when no beam is used. The same beam width is used during in training and when tagging the test set.

ADAPTIVE BEAM

Beam Width	Tagging Accuracy (%)	Training time (min)	Dec. Speed (KTok/s)
0.9	93.08 (OOV: 76.99)	3 min, 3 epochs	6
0.99	93.14 (OOV: 77.44)	3 min, 3 epochs	8
0.999	93.28 (OOV: 80.49)	2 min, 2 epochs	8

FIXED BEAM

Beam Width	Tagging Accuracy (%)	Training time	Dec. Speed (KTok/s)
1	92.32 (OOV: 75.07)	2 min, 2 epochs	7
10	93.33 (OOV: 78.28)	2 min, 2 epochs	7
20	93.11 (OOV: 77.29)	4 min, 3 epochs	7
∞	93.31 (OOV: 78.19)	20 min, 2 epochs	6

Table 8.4: Different beam settings for FinnTreeBank.

ADAPTIVE BEAM

Beam Width	Tagging Accuracy (%)	Training time (min)	Dec. Speed (KTok/s)
0.9	92.55 (OOV: 73.73)	5 min, 5 epochs	6
0.99	92.87 (OOV: 74.88)	7 min, 7 epochs	6
0.999	92.76 (OOV: 74.65)	8 min, 7 epochs	6

FIXED BEAM

Beam Width	Tagging Accuracy (%)	Training time (min)	Dec. Speed (KTok/s)
1	91.80 (OOV: 71.26)	6 min, 9 epochs	6
10	92.83 (OOV: 74.85)	7 min, 6 epochs	6
20	92.60 (OOV: 73.98)	10 min, 7 epochs	6
∞	91.58 (OOV: 69.46)	199 min, 8 epochs	6

Table 8.5: Different beam settings for Turku Dependency Treebank.

Results It is difficult to see a clear relation between the beam width and tagging accuracy. The fixed beam of width one is clearly the worst for both TDT and FTB. Larger beams give improved results when compared to beam width one, however, all larger beams seem to give results in the same range. Moreover, increasing the beam from 10 to 20 results in a 0.1%-point drop in accuracy for FTB. Additionally, the

system without beam search performs worse than systems with adaptive or fixed beam width for TDT.

A small beam width results in a faster training time than a larger beam. When using an infinite beam, the training time for TDT is surprisingly large. The reason for this is that there are sequences of words in the training and development data which receive a large number of label guesses. When no beam is used, this results in very long tagging times for the sequences because a second order model is used.

Discussion It seems that the effect of beam width on tagging accuracy is not easy to analyze. Beam width one gives inferior results when compared to other beam settings but all other beam settings seem to deliver accuracy in the same range. The effect on training time is, however, clear. Larger beam width slows down training.

The effect of large beam widths on training time can be dramatic as exemplified by the results for TDT using infinite beam width. Analysis of the problem revealed that there are sentences in the treebank that contain words which consist of characters that only occur in those words. Such words will receive a very large amount of guesses when label guesses are pruned based probability mass. This happens because the suffix based label guesser uses Laplace smoothing. The distribution $p(y|x)$ of labels y given a word form x becomes almost flat when the suffixes of x only occur a single time in the training data. For example, the three Greek words in "Larnakan lentoasema (kreik. Διεθνές Αεροδρόμιο Λάρνακας) on Kyproksen kansainvälinen lentoasema" receive 1287 label guesses each, that is every label in the TDT label set, when using mass 0.999. When no beam is employed, sequences of consecutive words with extremely many labels candidates result in extreme tagging times for the sentence.[3]

In contrast to training time, the effect of beam width on decoding time is minimal. This happens because decoding uses the label dictionary which means that for most words the tagger will choose the label from a very restricted set of candidates, typically one or two analyses.

Contrary to what the literature indicates (Huang et al., 2012, Collins and Roark, 2004), violation fixing gave no significant improvements in accuracy in preliminary experiments. As it only slows down training, it was not included in FinnPos.

In conclusion, it seems that the effect of beam width on tagging accuracy is erratic but the effect on training time is clear. Therefore, it is probably recommendable to use a beam. The mass based beam resulted in similar training times and tagging accuracies as the fixed beam.

I think it is interesting that infinite beam width results in inferior results for TDT when compared with other beam settings. This effect was also noted by Huang et al. (2012) when performing experiments in POS tagging. I have no explanation for this at the current time.

WITHOUT A MORPHOLOGICAL ANALYZER

Model Order	Tagging Accuracy (%)	Training time	Dec. Speed (KTok/s)
0	91.91	3 min, 3 epochs	8
1	92.49	3 min, 4 epochs	8
2	92.49	4 min, 5 epochs	7

USING A MORPHOLOGICAL ANALYZER

Model Order	Tagging Accuracy (%)	Training time	Dec. Speed (KTok/s)
0	95.35	5 min, 9 epochs	25
1	95.98	5 min, 10 epochs	23
2	95.96	5 min, 8 epochs	24

Table 8.6: Different Model Orders for FinnTreeBank

WITHOUT A MORPHOLOGICAL ANALYZER

Model Order	Tagging Accuracy (%)	Training time (min)	Dec. Speed (KTok/s)
0	91.15	6 min, 4 epochs	6
1	91.17	8 min, 8 epochs	5
2	91.83	5 min, 3 epochs	5

USING A MORPHOLOGICAL ANALYZER

Model Order	Tagging Accuracy (%)	Training time (min)	Dec. Speed (KTok/s)
0	95.53	5 min, 4 epochs	21
1	96.05	5 min, 7 epochs	22
2	96.13	5 min, 5 epochs	20

Table 8.7: Different Model Orders for Turku Dependency Treebank

8.5 Model Order

In this section I examine the impact of model order on tagging accuracy, training time and decoding time. I examine the effect of model order both when tagging without an analyzer and wen using an analyzer.

Setup The experiments in this section do not use sub-label features in order to clearly reveal the impact of model order in isolation of other factors.

[3]FinnPos has now been fixed to employ a user defined ceiling on the amount of guesses used during training. This setting will, however, degrade the accuracy of the tagger to some extent. Therefore, the setting has not been used in these experiments.

Results The accuracy on both FTB and TDT increases when going from order zero to a first order model. Further increasing model order only gives an improvement for TDT.

The increase in accuracy when going from a unstructured (order zero) model to a second order model is approximately 0.5%-points for both FTB and TDT. This applies both when using a morphological analyzer and when not using it. It is noteworthy that the increase in accuracy resulting from the morphological analyzer is substantially larger for both data sets.

Discussion The fact that a second order model improves results only for TDT could be a result of the fact that FTB is smaller, however, the difference in corpus size is quite small (only about 11% or about 20,000 words). A more likely explanation relates to the fact that the average sentence length in TDT is 13 words, whereas sentences in FTB only have 8 words on average. Higher sentence length translates to longer average distance between syntactically dependent words. Therefore, a second order model, which can model longer dependencies, may be more helpful when tagging TDT.

Overall, it seems like the improvement from using a second order model over a zeroth order model is quite small. Partly, this is probably a result of the fact that both of the data sets are quite small. Moreover, the unstructured word context features included in the feature set partly overlap with structured features and thus diminish their effect.

Finally, it is interesting to see that the improvement resulting from model order is about equal when using a morphological analyzer and when not using one even though the tagging accuracies for the zeroth order models when using an analyzer and without one are about 4 %-points apart. This shows that using the analyzer does not nullify the improvement from other improvements to tagging accuracy. Ultimately, the impact of the analyzer on tagging accuracy is however of a higher magnitude than the impact of model order.

8.6 Sub-Label Dependencies

In this section, I examine the impact of sub-label order on tagging accuracy, training time and decoding time both using a morphological analyzer and without a morphological analyzer. The results in this section differ slightly from Publication **VI** because of minor bug fixes and improvements to the feature set of the tagger.

Setup The different model configurations, which are investigated and shown in Tables 8.8 and 8.9, are (1) no sub-label dependencies, (2) unstructured sub-label dependencies, (3) unstructured and first order sub-label dependencies and (4) unstructured, first and second order sub-label dependencies.

Results When a morphological analyzer is not used as part of the tagger, total improvement in accuracy stemming from sub-label dependencies is approximately 0.8%-points for both FTB and TDT. This is

WITHOUT A MORPHOLOGICAL ANALYZER

Sub-Label Order	Tagging Accuracy (%)	Training time	Dec. Speed (KTok/s)
None	92.49 (OOV: 74.68)	3 min, 5 epochs	6
0	93.05 (OOV: 77.74)	1 min, 2 epochs	5
1	93.29 (OOV: 78.40)	1 min, 2 epochs	4
2	92.68 (OOV: 75.22)	5 min, 4 epochs	6

USING A MORPHOLOGICAL ANALYZER

Sub-Label Order	Tagging Accuracy (%)	Training time	Dec. Speed (KTok/s)
None	95.98 (OOV: 91.41)	3 min, 8 epochs	25
0	96.08 (OOV: 91.98)	1 min, 3 epochs	22
1	96.24 (OOV: 92.28)	1 min, 2 epochs	21
2	96.31 (OOV: 92.58)	4 min, 3 epochs	19

Table 8.8: Different Sub-Label Orders for FinnTreeBank

WITHOUT A MORPHOLOGICAL ANALYZER

Sub-Label Order	Tagging Accuracy (%)	Training time	Dec. Speed (KTok/s)
None	91.89 (OOV: 70.63)	2 min, 3 epochs	5
0	92.59 (OOV: 73.98)	2 min, 4 epochs	5
1	92.69 (OOV: 74.35)	5 min, 7 epochs	3
2	92.31 (OOV: 72.31)	13 min, 8 epochs	5

USING A MORPHOLOGICAL ANALYZER

Sub-Label Order	Tagging Accuracy (%)	Training time	Dec. Speed (KTok/s)
None	96.12 (OOV: 91.12)	3 min, 5 epochs	19
0	96.17 (OOV: 91.39)	2 min, 5 epochs	18
1	96.39 (OOV: 91.84)	3 min, 5 epochs	16
2	96.29 (OOV: 91.69)	12 min, 8 epochs	16

Table 8.9: Different Sub-Label Orders for Turku Dependency Treebank

higher than the improvement stemming from increasing model order (from model order zero to model order two).

When a morphological analyzer is used as part of the tagger, total improvement in accuracy provided by sub-label dependencies is approximately the same as transitioning from model order zero to one.

Only for FTB, do second order sub-label features give added accuracy compared to first order features and only when using the morphological analyzer. However, the improvement is not statistically significant. In other cases, second order sub-label dependencies degrade performance compared to first order

sub-label dependencies.

As can be expected, training time increases and decoding speed decreases with increasing sub-label order. However, sub-label dependencies seems to decrease the amount of training epochs needed to converge to the final model parameters.

When a morphological analyzer is not used, unstructured sub-label features are more influential for accuracy than structured sub-label features for both FTB and TDT. When the analyzer is used, the opposite is true. Structured sub-label dependencies improve performance substantially more. In fact, unstructured sub-label dependencies alone do not provide a statistically significant improvement, when the analyzer is used, whereas a combination of unstructured and first order structured sub-label dependencies do. Moreover, the improvement given by first order sub-label dependencies with regard to unstructured dependencies is greater when the analyzer is used than when it is not used.

Discussion As stated in Section 6.5, sub-label dependencies can improve accuracy in two ways

1. they can counteract data sparsity and

2. capture congruence and other phenomena that transcend individual word classes.

Experimentally, it is difficult to discern these two effects (probably the experiments in this section cannot do this). Perhaps they are not even distinct effects. After all, in the presence of a sufficient amount of training data, all combinations of full labels are observed and there is no need for modeling congruence and other similar phenomena using sub-label dependencies. In practice the data is, however, always sparse. And I think that in the case of insufficient data, a stronger structured model can yield better results because it utilizes the training data in a richer manner.

Unstructured sub-label dependencies probably mainly act to reduce data sparsity. This is a credible explanation for the improvement in accuracy because their effect is almost completely nullified by the morphological analyzer whose main purpose is similarly to counteract data sparsity which arises because of the large variety of inflections in Finnish text. In contrast, the effect of structured sub-label dependencies is not nullified by the morphological analyzer. In fact the improvement is greater when the analyzer is used, when compared to the setting where only unstructured sub-label dependencies are used.

It is possible that the effect of structured sub-label dependencies still mainly stems from reduced data sparsity in the structured model but a part of the reduction of data sparsity is that significant grammatical relations can be learned from the data.

The experiments show that sub-label dependencies deliver at least as great improvements as increasing model order which is a standard trick in sequence labeling. Still, however, the impact from using a morphological analyzer dwarfs both of these effects.

8.7 Model Pruning

In this section, I examine two model pruning strategies: pruning by update count and pruning by parameter mass. The strategies are presented in 6.7.

Setup The value for the pruning parameter was set using development data. The range of parameter values was chosen so as to show the difference in pruning efficiency. The specific parameter values are not very important. The important aspect is the relation of model size and accuracy. These experiments only investigate pruning for tagger parameters. Pruning could, however, also be applied to the data-driven lemmatizer.

	UPDATE COUNT THRESHOLD				
MA	None	< 2	< 3	< 4	< 5
no	93.2%, 4.8M	93.2%, 3.9M	93.2%, 3.6M	93.2%, 3.0M	93.1%, 1.0M
yes	96.3%, 4.3M	96.2%, 3.9M	96.2%, 3.7M	96.2%, 3.3M	96.1%, 1.0M

	PARAMETER MASS THRESHOLD				
MA	None	< 2.0	< 2.5	< 3	< 3.5
no	93.2%, 4.8M	93.3%, 1.8M	93.2%, 1.4M	93.2%, 1.2M	93.2%, 0.9M
yes	96.3%, 4.3M	96.3%, 0.9M	96.3%, 0.7M	96.2%, 0.3M	96.1%, 0.1M

Table 8.10: Result of applying different pruning strategies on FinnTreeBank models.

	UPDATE COUNT THRESHOLD				
MA	None	< 2	< 3	< 4	< 5
no	92.8%, 6.4M	92.7%, 5.2M	92.7%, 5.0M	92.8%, 4.9M	92.6%, 4.9M
yes	96.3%, 5.5M	96.4%, 5.0M	96.3%, 4.9M	96.4%, 4.9M	96.3%, 4.9M

	PARAMETER MASS THRESHOLD				
MA	None	< 4.0	< 5.0	< 6.0	< 7.0
no	92.8%, 6.4M	92.8%, 2.9M	92.7%, 2.8M	92.7%, 2.6M	92.6%, 2.1M
yes	96.3%, 5.5M	96.3%, 1.2M	96.3%, 0.8M	96.2%, 0.2M	96.2%, 0.2M

Table 8.11: Result of applying different pruning strategies on Turku Dependency Treebank models.

Results The results for FTB and TDT are shown in Tables 8.10 and 8.11, respectively. The results are visualized in Figure 8.1.

Clearly, mass based pruning is more effective than update count based pruning. For FTB, without a morphological analyzer, the full accuracy of 93.2% can be maintained even when pruning out 81% of model parameters. When using update count as pruning criterion, full accuracy cannot be maintained when pruning out more than 38% of model parameters. For TDT, the corresponding figures are 55% for mass based pruning and 23% for update based pruning.

When using a morphological analyzer, even further feature pruning is possible. For FTB, 84% of model parameters can be pruned while maintaining full accuracy when using mass based pruning. When using update count based pruning, however, no parameters can be pruned without losing accuracy. For TDT, update count based pruning can prune out 72% of the features when using a morphological analyzer but mass based pruning can prune out even more – 81%.

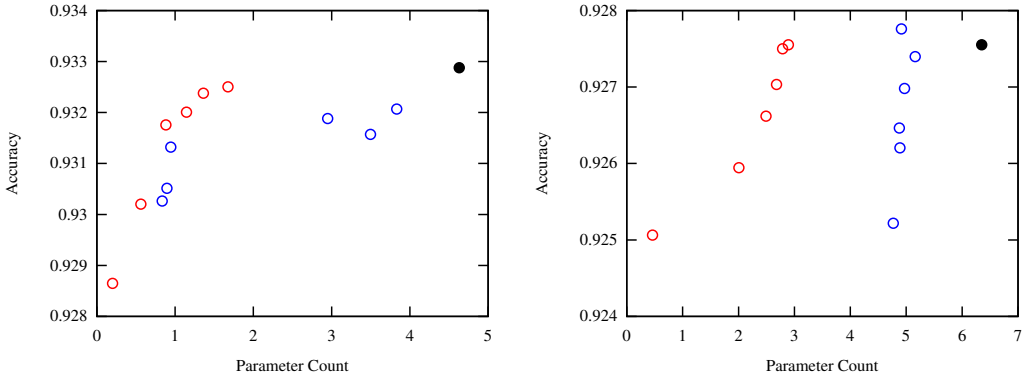
Discussion The guiding principle for pruning based on update count is that parameters which receive few updates activate rarely. Thus they are not very influential for tagging accuracy. Whereas this may give a sufficient criterion for determining that a parameter is non-influential, it does not give a necessary condition. There are features that activate often but do not help in tagging. For example, features sharing the template **The word begins with "a"** activate often in any realistic data set for Finnish morphological tagging. However, they are almost completely uninformative. Therefore, provided sufficient data, their update count will be high but the absolute value of the features will close to zero because the updates cancel out as the features activate approximately equally often for all labels. Value based pruning will prune out both features that activate rarely and features that activate often but do not provide additional information for the tagging task. It is, therefore, not surprising that the experiments quite clearly show that value based pruning is superior on the FTB and TDT data sets.

Models can be pruned more heavily when the morphological analyzer is used. This probably reflects the fact that the tagging task is easier when the tagger can rely on the analyzer. For example, a substantial part of word forms only receive one label from the morphological analyzer. Therefore, the disambiguation task becomes trivial for these words.

Value based pruning may not be able to filter out features that are highly correlated with other features. This case should be handled using for example L1 regularization for perceptron taggers Zhang et al. (2014).

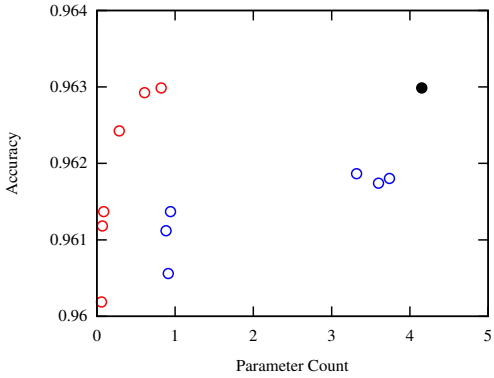
Training and development times are not included in Tables 8.10 and 8.11. However, model development is substantially more time consuming when update count based pruning is used. This happens because a new model has to be trained for every threshold, as the threshold influences the model during training. In contrast, mass based pruning can be performed on a trained model using several thresholds.

Finally, it is interesting to note that some pruned models yields better results for TDT than the original models. It may be that pruning can help to regularize the model in some cases, however, the differences in these experiments are not statistically significant.

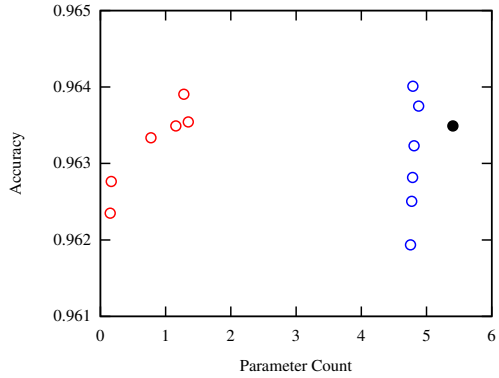


(a) FTB, no analyzer

(b) TDT, no analyzer



(c) FTB, analyzer



(d) TDT, analyzer

Figure 8.1: This figure visualizes the trade-off between model accuracy and model size which can be achieved using value based pruning (the red data points) and update count based pruning (the blue data points). The black data point in each graph represents the original model without pruning. Data points that lie close to the upper left corner of the graph represent models that are pruned efficiently while maintaining a lot of the original accuracy. In contrast, data points closer to the lower right corner represent models where pruning is unable to reduce model size effectively but the accuracy of the model still degrades. The general tendency is that red data points are closer to the upper left corner than blue ones, which means that value based pruning is more effective than update count based pruning.

Chapter 9

Conclusions and Future Work

This thesis has presented work on data driven morphological tagging for Finnish using both generative and discriminative models.

Generative Taggers The finite-state implementation of generative taggers which is presented in Publications **I** and **II** allows for flexible model formulation. Publication **II** shows that it compares favorably to the widely used HunPos tagger when tagging Finnish text. The implementation does, however, not solve the principal problem of HMM taggers: the independence assumptions in the model are too harsh. Therefore, complex unstructured features such as word context cannot be used. This is probably the greatest pitfall of generative models because surrounding words are quite useful in morphological tagging.

There are other reasons to prefer discriminative models above the generative tagging paradigm. All generative models require some manner of smoothing. It is difficult to know what the optimal choice of smoothing method. This may even be language specific to some extent. For example, Publication **II** indicates that the guesser presented in Brants (2000) may not be optimally suited for morphologically complex languages such as Finnish. A guesser based on the longest common suffix with words in the training data may give better results.

Discriminative Taggers The discriminative model presented in this thesis is based on the averaged perceptron model. It incorporates sub-label dependencies to improve accuracy in presence of large structured label sets and a cascaded model structure and beam search in order to speed up estimation. Moreover, I investigated different pruning strategies for models and showed that model size can be reduced by up to 80% with negligible reduction in accuracy. The FinnPos toolkit implements these optimizations and is freely available as an open-source utility.

The experiments in Chapter 8 show that the morphological analyzer is clearly the most influential factor for the accuracy of the model. It results in much larger gains in accuracy than increasing model

order or using sub-label dependencies. Sub-label dependencies however are equally or more influential than increased model order. This is not entirely surprising because a second order model will be very sparse when used in presence of label sets exceeding a thousand labels and training data on the order of 200,000 tokens. In contrast, sub label dependencies are by definition less sparse.

Future Work It would be interesting to try self-training as presented by Søgaard (2011). Other semi-supervised training methods such as distributional similarity could also be interesting. In the context of morphologically complex languages, distributional similarity of word forms might require a large amount of training data. Therefore, it could also be interesting to explore using lemmatized training material.

All experiments in this thesis have used a full-fledged morphological analyzer. It would be interesting to try out a morphological segmentation application such as Morfessor (Creutz and Lagus, 2002). The segments could be used as features in a similar manner as the morphological labels are used in the current system.

Further feature engineering could probably be useful. For example verb valency could be useful. It would also be interesting to combine the finite-state implementation presented in Publications **V** and **VI** with the discriminative estimation in the FinnPos toolkit. Especially, it would be interesting to explore global tagger constraints implemented as features in a discriminative tagger. An simple example of such a feature is **the sentence has a finite verb form**. This would be possible using the finite-state implementation which is not constrained by a fixed model order unlike standard inference using the Viterbi algorithm. However, efficient estimation would probably be a challenge.

As further work on model pruning, it would be interesting to compare value based pruning and L1 regularization for perceptron taggers and investigate the combination of these methods.

References

Ahlberg, M., Forsberg, M., and Hulden, M. (2014). Semi-supervised learning of morphological paradigms and lexicons. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, EACL2014, pages 569–578. Association for Computational Linguistics.

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. (2007). OpenFst: A general and efficient weighted finite-state transducer library. In *Implementation and Application of Automata: 12th International Conference, Revised Selected Papers*, pages 11–23. Springer Berlin Heidelberg.

Baker, C. F., Fillmore, C. J., and Lowe, J. B. (1998). The berkeley FrameNet project. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1*, ACL1998, pages 86–90. Association for Computational Linguistics.

Beesley, K. R. and Karttunen, L. (2003). *Finite state morphology*. CSLI Publications.

Berg-Kirkpatrick, T., Bouchard-côté, A., Denero, J., and Klein, D. (2010). Painless unsupervised learning with features. In *Proceedings of the 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL2010. Association for Computational Linguistics.

Besag, J. (1975). Statistical analysis of non-lattice data. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 24(3):179–195.

Bilmes, J. (1997). A gentle tutorial on the EM algorithm and its application to parameter estimation for gaussian mixture and hidden Markov models. Technical Report ICSI-TR-97-021, University of Berkeley.

Bohnet, B., Nivre, J., Boguslavsky, I., Farkas, R., Ginter, F., and Hajic, J. (2013). Joint morphological and syntactic analysis for richly inflected languages. *Transactions of the Association for Computational Linguistics*, 1:415–428.

Boyen, X. and Koller, D. (1998). Tractable inference for complex stochastic processes. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, UAI1998, pages 33–42. Morgan Kaufmann Publishers Inc.

Brants, T. (2000). TnT - a statistical part-of-speech tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing (ANLP2000)*. Association for Computational Linguistics.

Brill, E. (1992). A simple rule-based part of speech tagger. In *Proceedings of the third conference on Applied natural language processing*, ANLC1992, pages 152–155. Association for Computational Linguistics.

Brill, E. and Moore, R. C. (2000). An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, ACL2000, pages 286–293. Association for Computational Linguistics.

Bybee, J. L. (1985). *Morphology: A study of the relation between meaning and form*. Benjamins.

Carr, P. (1993). *Phonology*. The Macmillan Press LTD.

Ceausu, A. (2006). Maximum entropy tiered tagging. In *The 11th ESSLI Student session*, pages 173–179.

Charniak, E. and Johnson, M. (2005). Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL2005, pages 173–180. Association for Computational Linguistics.

Chrupala, G., Dinu, G., and van Genabith, J. (2008). Learning morphology with Morfette. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC2008)*. European Language Resources Association (ELRA).

Church, K. (2005). The DDI approach to morphology. In *Inquiries into Words, Constraints and Contexts: Festschrift for Kimmo Koskenniemi on his 60th Birthday*, pages 25–34. CSLI Publications.

Church, K. W. (1988). A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the second conference on Applied natural language processing*, ANLC1988, pages 136–143. Association for Computational Linguistics.

Collins, M. (2002). Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical methods in natural language processing*, EMNLP2002, pages 1–8. Association for Computational Linguistics.

Collins, M. and Roark, B. (2004). Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL2004. Association for Computational Linguistics.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7:551–585.

Creutz, M. and Lagus, K. (2002). Unsupervised discovery of morphemes. In *Proceedings of the ACL2002 Workshop on Morphological and Phonological Learning*, MPL2002, pages 21–30. Association for Computational Linguistics.

Cutting, D., Kupiec, J., Pedersen, J., and Sibun, P. (1992). A practical part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, ANLC1992, pages 133–140. Association for Computational Linguistics.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Jornal of the Royal Statistical Society, Series B*, 39(1):1–38.

DeRose, S. J. (1988). Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14(1):31–39.

Forney, G. D. (2005). The Viterbi algorithm: A personal history. *arXiv*. http://arxiv.org/abs/cs/0504020.

Francis, W. N. (1964). *A standard sample of present-day English for use with digital computers*. Department of Linguistics, Brown University.

Freund, Y. and Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computing*, 4(1):1–58.

Goldberg, Y. and Elhadad, M. (2011). Learning sparser perceptron models. Technical report. Ben Gurion University of the Negev.

Hakulinen, A., Korhonen, R., Vilkuna, M., and Koivisto, V. (2004). *Iso suomen kielioppi*. Suomalaisen kirjallisuuden seura.

Halácsy, P., Kornai, A., and Oravecz, C. (2007). HunPos: An open source trigram tagger. In *Proceedings of the 45th Annual Meeting of the ACL*, ACL2007, pages 209–212. Association for Computational Linguistics.

Haverinen, K., Nyblom, J., Viljanen, T., Laippala, V., Kohonen, S., Missilä, A., Ojala, S., Salakoski, T., and Ginter, F. (2014). Building the essential resources for Finnish: The Turku Dependency Treebank. *Language Resources and Evaluation*, 48(3):493–531.

Horsmann, T., Erbs, N., and Zesch, T. (2015). Fast or accurate? – A comparative evaluation of PoS tagging models. In *Proceedings of the International Conference of the German Society for Computational Linguistics and Language Technology*, GSCL2015, pages 22–30. German Society for Computational Linguistics and Language Technology.

Huang, F. and Yates, A. (2009). Distributional representations for handling sparsity in supervised sequence labeling. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, ACL2009, pages 495–503. Association of Computational Linguistics.

Huang, L., Fayong, S., and Guo, Y. (2012). Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL HLT2012, pages 142–151. Association for Computational Linguistics.

Huang, Z., Eidelman, V., and Harper, M. (2009). Improving a simple bigram HMM part-of-speech tagger by latent annotation and self-training. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, NAACL2009, pages 213–216. Association for Computational Linguistics.

Hulden, M., Forsberg, M., and Ahlberg, M. (2014). Semi-supervised learning of morphological paradigms and lexicons. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 569–578. Association for Computational Linguistics.

Hulden, M. and Francom, J. (2012). Boosting statistical tagger accuracy with simple rule-based grammars. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC2012)*. European Language Resources Association.

Hulden, M., Silfverberg, M., and Francom, J. (2013). Finite state applications with javascript. In *Proceedings of the 19th Nordic Conference of Computational Linguistics*, NODALIDA 2013, pages 441–446. LiU Electronic Press.

Johnson, M. (2007). Why Doesn't EM Find Good HMM POS-Taggers? In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 296–305. Association for Computational Linguistics.

Kaplan, R. M. and Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

Karlsson, F., Voutilainen, A., Heikkilä, J., and Anttila, A. (1995). *Constraint Grammar: A Language-Independent Framework for Parsing Unrestricted Text.* Mouton de Gruyter.

Kettunen, K., Kunttu, T., and Järvelin, K. (2005). To stem or lemmatize a highly inflectional language in a probabilistic IR environment? *Journal of Documentation*, 61(4):476–496.

Klein, D. and Manning, C. D. (2002). Conditional structure versus conditional estimation in nlp models. In *Proceedings of the Conference on Empirical methods in natural language processing*, EMNLP2002, pages 9–16. Association for Computational Linguistics.

Knuth, D. E., Morris, J. H., and Pratt, V. R. (1977). Fast Pattern Matching in Strings. *SIAM Journal on Computing*, 6(2):323–350.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press.

Koskenniemi, K. (1984). A general computational model for word-form recognition and production. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd Annual Meeting of the Association for Computational Linguistics*, pages 178–181. Association for Computational Linguistics.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML2001, pages 282–289. Morgan Kaufmann Publishers Inc.

Lindén, K. (2009). Entry generation by analogy – Encoding new words for morphological lexicons. *Northern European Journal of Language Technology*, 1:1–25.

Lindén, K., Silfverberg, M., and Pirinen, T. (2009). HFST tools for morphology – an efficient open-source package for construction of morphological analyzers. In *State of the Art in Computational Morphology: Workshop on Systems and Frameworks for Computational Morphology. Proccedings*, SFCM2009, pages 28–47. Springer Berlin Heidelberg.

Liu, D. C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(3):503–528.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Merialdo, B. (1994). Tagging English text with a probabilistic model. *Comput. Linguist.*, 20(2):155–171.

Mohri, M. (1997). String matching with automata. *Nordic Journal of Computing*.

Mohri, M., Pereira, F., and Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69 – 88.

Müller, T., Cotterell, R., Fraser, A. M., and Schütze, H. (2015). Joint lemmatization and morphological tagging with Lemming. In *Proceedings of the 2015 Joint Conference on Empirical Methods in Natural Language Processing*, EMNLP2015, pages 2268–2274. Association for Computational Linguistics.

Müller, T., Schmid, H., and Schütze, H. (2013). Efficient higher-order CRFs for morphological tagging. In *Proceedings of 2013 Empirical Methods in Natural Language Processing*, EMNLP2013, pages 322–332.

Nguyen, N. and Guo, Y. (2007). Comparisons of sequence labeling algorithms and extensions. In *Proceedings of the 24th International Conference on Machine Learning*, ICML2007, pages 681–688. Association for Computing Machinery.

Orosz, G. and Novák, A. (2013). PurePos 2.0: a hybrid tool for morphological disambiguation. In *Proceedings of the Ninth International Conference on Recent Advances in Natural Language Processing*, RANLP2013, pages 539–545. RANLP 2013 Organising Committee / Association for Computational Linguistics.

Östling, R. (2013). Stagger: an open-source part of speech tagger for swedish. *Northern European Journal of Language Technology*, 3:1–18.

Pearl, J. (1982). Reverend bayes on inference engines: a distributed hierarchical approach. In *Proceedings of the National Conference on Artificial Intelligence*, pages 133–136. American Association of Artificial Intelligence.

Pirinen, T., Silfverberg, M., and Lindén, K. (2012). Improving finite-state spell-checker suggestions with part of speech n-grams. In *Proceedings of the 13th International Conference on Intelligent Text Processing and Computational Linguistics*, CICLing2012. Springer Berlin Heidelberg.

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 the 18th Conference of Computational Linguistics*, NODALIDA2011, pages 299—302. Northern European Association for Language Technology.

Porter, M. F. (1997). An algorithm for suffix stripping. In Sparck Jones, K. and Willett, P., editors, *Readings in Information Retrieval*, pages 313–316. Morgan Kaufmann Publishers Inc.

Rabiner, L. R. (1990). A tutorial on hidden markov models and selected applications in speech recognition. In Waibel, A. and Lee, K.-F., editors, *Readings in Speech Recognition*, pages 267–296. Morgan Kaufmann Publishers Inc.

Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP1996. Association for Computational Linguistics.

Ratnaparkhi, A. (1997). A simple introduction to maximum entropy models for natural language processing. Technical report, Institute for Research in Cognitive Science, University of Pennsylvania.

Ratnaparkhi, A. (1998). *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, University of Pennsylvania.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Ruokolainen, T. and Silfverberg, M. (2013). Modeling oov words with letter n-grams in statistical taggers: Preliminary work in biomedical entity recognition. In *Proceedings of the 19th Nordic Conference of Computational Linguistics (NODALIDA 2013)*, pages 181–193. LiU Electronic Press.

Ruokolainen, T., Silfverberg, M., Kurimo, M., and Linden, K. (2014). Accelerated estimation of conditional random fields using a pseudo-likelihood-inspired perceptron variant. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, EACL2014, pages 74–78. Association for Computational Linguistics.

Samuelsson, C. and Voutilainen, A. (1997). Comparing a linguistic and a stochastic tagger. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, ACL1998, pages 246–253. Association for Computational Linguistics.

Silfverberg, M. and Lindén, K. (2009). Conflict resolution using weighted rules in HFST-TWOLC. In *Proceedings of the 17th Nordic Conference of Computational Linguistics*, NODALIDA2009, pages 174–181. Northern European Association for Language Technology.

Silfverberg, M. and Lindén, K. (2011). Combining statistical models for POS tagging using finite-state calculus. In *Proceedings of the 18th Conference of Computational Linguistics*, NODALIDA2011, pages 183–190. Northern European Association for Language Technology.

Silfverberg, M. and Lindén, K. (2010). Part-of-speech tagging using parallel weighted finite-state transducers. In *Proceedings of the 7th International Conference on NLP*, IceTAL2010, pages 369–380. Springer Berlin Heidelberg.

Silfverberg, M., Ruokolainen, T., Lindén, K., and Kurimo, M. (2014). Part-of-speech tagging using conditional random fields: Exploiting sub-label dependencies for improved accuracy. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, ACL2014, pages 259–264. Association for Computational Linguistics.

Silfverberg, M., Ruokolainen, T., Lindén, K., and Kurimo, M. (2015). FinnPos: an open-source morphological tagging and lemmatization toolkit for Finnish. *Language Resources and Evaluation*, pages 1–16. Online first article not yet assigned to an issue.

Sipser, M. (1996). *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition.

Smith, N. A., Smith, D. A., and Tromble, R. W. (2005). Context-based morphological disambiguation with random fields. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT2005, pages 475–482. Association for Computational Linguistics.

Søgaard, A. (2011). Semisupervised condensed nearest neighbor for part-of-speech tagging. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, HLT2011, pages 48–52. Association for Computational Linguistics.

Spoustová, D. j., Hajič, J., Raab, J., and Spousta, M. (2009). Semi-supervised training for the averaged perceptron POS tagger. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL2009, pages 763–771. Association for Computational Linguistics.

Spoustová, D. j., Hajič, J., Votrubec, J., Krbec, P., and Květoň, P. (2007). The best of two worlds: Cooperation of statistical and rule-based taggers for Czech. In *Proceedings of the Workshop on Balto-Slavonic Natural Language Processing: Information Extraction and Enabling Technologies*, ACL2007, pages 67–74. Association for Computational Linguistics.

Sutton, C. and McCallum, A. (2007). Piecewise pseudolikelihood for efficient training of conditional random fields. In *Proceedings of the 24th International Conference on Machine Learning*, ICML2007, pages 863–870. Association for Computing Machinery.

Sutton, C. and McCallum, A. (2012). An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4):267–373.

Tapanainen, P. and Järvinen, T. (1997). A non-projective dependency parser. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, ANLC1997, pages 64–71. Association for Computational Linguistics.

Taskar, B., Guestrin, C., and Koller, D. (2004). Max-margin markov networks. In *Advances in Neural Information Processing Systems*, NIPS2003. The MIT Press.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)*, 58:267–288.

Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL2003, pages 173–180. Association for Computational Linguistics.

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484.

Tufis, D. (1999). Tiered tagging and combined language models classifiers. In *Proceedings of the Second International Workshop on Text, Speech and Dialogue*, TSD1999, pages 28–33. Springer Berlin Heidelberg.

Vishwanathan, S. V. N., Schraudolph, N. N., Schmidt, M. W., and Murphy, K. P. (2006). Accelerated training of conditional random fields with stochastic gradient methods. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML2006, pages 969–976. Association for Computing Machinery.

Voutilainen, A. (1995). A syntax-based part-of-speech analyser. In *Proceedings of the Seventh Conference on European Chapter of the Association for Computational Linguistics*, EACL1995, pages 157–164. Morgan Kaufmann Publishers Inc.

Voutilainen, A. (2011). FinnTreeBank: Creating a research resource and service for language researchers with Constraint Grammar. In *Proceedings of the NODALIDA 2011 Workshop on Constraint Grammar Applications*, pages 41–49. Northern European Association for Language Technology.

Weiss, D. J. and Taskar, B. (2010). Structured prediction cascades. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, AISTATS2010, pages 916–923. JMLR.

Weiss, Y. (2000). Correctness of local probability propagation in graphical models with loops. *Neural Computing*, 12(1):1–41.

Wilcoxon, F. (1945). Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83.

Zhang, K., Su, J., and Zhou, C. (2014). Regularized structured perceptron: A case study on chinese word segmentation, POS tagging and parsing. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 164–173. Association for Computational Linguistics.