Department of Modern Languages
2015

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 auditorium FOO (in lecture room QUUX), on the XY^{th} of February at 12 o'clock.

University of Helsinki
Finland

Supervisor
Krister Lindén, Helsingin yliopisto, Finland, Anssi Yli-Jyrä, Helsingin yliopisto, Finland

Pre-examiners
XYZ, FOO, BAR
XYZ, FOO, BAR

Opponent
XYZ, FOO, BAR

Custos
XYZ, FOO, BAR

Contact information

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

Email address: postmaster@helsinki.fi
URL: http://www.helsinki.fi/
Telephone: +358 9 1911, telefax: +358 9 191 51120

ISBN XXX-XXX-XX-XXXX-X (printed)
ISBN XXX-XXX-XX-XXXX-X (PDF)
Computing Reviews (1998) Classification: F.1.1, I.2.7, I.7.1
Helsinki 2015
FOO

Nobody is going to read your PhD thesis

Folklore

Contents

List of Publications

I Silfverberg, M. and Lindén, K. (2010). Part-of-speech tagging using parallel weighted finite-state transducers. In Loftsson, H., Rögnvaldsson, E., and Helgadóttir, S., editors, *Proceedings of the 7th International Conference on NLP, IceTAL 2010*, 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*, NEALT Proceedings Series. 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 *13th International Conference on Intelligent Text Processing and Computational Linguistics CICLing 2014*

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, volume 2: Short Papers*, pages 74–78, Gothenburg, Sweden. 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 *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short*

Papers), pages 259–264, Baltimore, Maryland. 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

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. The present author 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. The present author 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. The present author performed 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. The present author 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 between the components of structured morphological labels for improving tagging accuracy. The first author 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. The first author 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
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.
$\mathbb{R}^m_n$	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.

Acknowledgements

Chapter 1

Introduction

A morphological tagger is a piece of computer software that provides complete morphological descriptions of sentences. An example 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 rich 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.

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 almost 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 rich 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 rich 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 techniques for learning tagger models.

1.1 Motivation

Data-driven methods have dominated the field of natural language processing (NLP) since the nineties. 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 due to 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 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 all modern languages, Finnish has borrowed a lot of words from 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.[1]

The recently published FinnTreeBank (Voutilainen, 2011) and Turku Dependence Treebank (Haverinen et al., 2014) represent the first freely available broad coverage Finnish hand labeled 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 rich 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 worse on so called out-of-vocabulary (OOV) words, that is words which are missing from the training corpus, than on words seen during training. In English, this is usually not detrimental to the performance of the tagger, because the amount of OOV words is typically rather low. In contrast, this becomes a substantial problem for purely data driven systems processing morphologically rich languages because productive compounding and extensive inflection lead to a large amount of OOV words even within the same 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 rich 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 (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 rich languages, this approach fails because of data sparsity. Instead, a generative morphological tagger can be used score suggestions based on morphological context as shown in Publication **III**.

[1] 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).

- **Feature extraction specifically aimed at morphologically rich 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 rich 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 rich languages using stadard tagging techniques. Although the speed of tagging systems is not always seen as a major concern, I believe that both estimation and tagging speed is important. 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. I show that these lead to substantial reduction in training times and faster tagging times 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 rich 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 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 the contributions of the author 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 being a *word stem* and the second one being 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 examplified 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. These 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. Others, such as English, employ a mix of concatenation and non-concatenative phenomena. Still, concatenative morphology is probably found to some degree in all languages. It is especially prevalent in languages with rich morphology, such as Finnish or Turkish. From the point of view of language technology for morphologically rich 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 rules 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 given a 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 usually belong to particular word classes, however, several word classes may 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 systems and syntactic parsers, 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 translation of “a 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 rich 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 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 rich morphology. The Morfessor system splits word forms into a sequence of morphemelike 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 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

[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

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 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 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 words, typically the 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[3] 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.[4] 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.

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".

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

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

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 only refers to the POS label (the prediction) of the word “dog”. The information that verbs are much more likely to be followed by pronouns 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 limits. 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 others morphological labels, which makes estimation of statistics from data quite difficult.

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-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.[5]

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 $x = (x_1, \ldots, x_T)$ and its label sequence

[5] 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.

$y = (y_1, \ldots, y_T)$ the model is trained to maximize the probability $p(y_t|x,\ y_1,\ \ldots,\ y_{t-1})$ 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.

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.[6] 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[7], the averaged perceptron tagger is 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 taggers has been investigated by Chrupala et al. (2008)

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

[7] For example experiments performed by Nguyen and Guo (2007) indicate that the classification performance of the averaged perceptron algorithm can in fact be better than the performance of the Conditional Random field.

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 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 hybird systems that combine data-driven and linguist dirven 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 Chzech. 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 Natural Language Processing. 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. I will then elaborate on the different kinds of classifiers applied in NLP; both unstructured and structured.

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

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.

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.[1] 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. Amount of 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 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.

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

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 $x = (x_1, ..., x_n, 1)^\top \in \mathbb{R}^n$, 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.[2]

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

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 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 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.

[2] 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.

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.

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

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.[3] 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 *over-fitting*. Fortunately, the loss function can be modified to counteract over-fitting. 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 analytical methods to try to find 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.[4] 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 $\mathcal{D}$.

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] The solution is given by $\theta = X^+Y$ where $X^+ = (X^\top X)^{-1}X^\top$ is the More-Pennrose pseudo-inverse of X.

[4] At least, differentiability almost everywhere.

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.[5]

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 the 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. In practice, it converges faster than regular gradient descent **?** but is identical in all other respects except that it is an online estimation algorithm instead of a batch algorithm. The algorithm processes one random training example at a time. It uses the gradient $\nabla L_{\mathcal{D}[i]}(\theta)$ of the loss over the single training example $\mathcal{D}[i]$ to approximate the gradient $\nabla L_\mathcal{D}(\theta)$ for the entire training data $\mathcal{D}$.

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

[5] In general, stepsize $(\alpha_t)_{t \in \mathbb{N}}$ that resemble the harmonic sequence, that is $\sum \alpha_t^2 < \infty$ and $\sum \alpha_t = \infty$, guarantee convergence of gradient descent to an minimum of the loss function (if it exists) for a wide variety of functions **?**

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

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 $\theta = (0...0)^\top$. 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 over specified, 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 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.

[7] Another approach to counteracting over-fitting 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.

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 overfits 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 unregularized 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 model is trained. 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. When investigating the tagging accuracy, tagging speed or training speed of the FinnPos toolkit, the baseline will be another established tagger toolkit.

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

AN INTRO

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 the weather is clear. 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

[1] To make things simple, imagine the nurse never gets a day off.

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

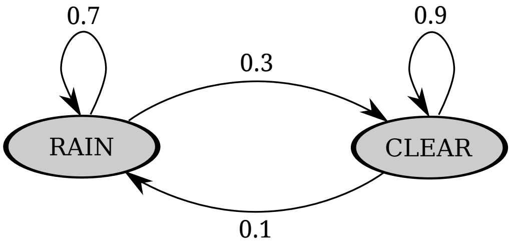
T	CLEAR	RAIN
CLEAR	0.9	0.1
RAIN	0.3	0.7

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: Foo



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 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

[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.

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 (1989)[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 4.2 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]

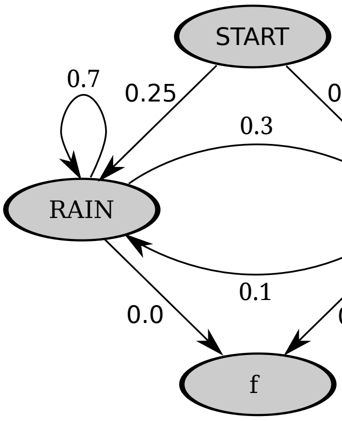
An HMM models a number of useful quantities:

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.
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.

[3] The definition of HMMs in this thesis differs slightly from Rabiner (1989) 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: Foo



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 means finding a maximally probable sequence of hidden states y that might have emitted the observation x. As Rabiner (1989) 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 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. 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, 1989). *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.

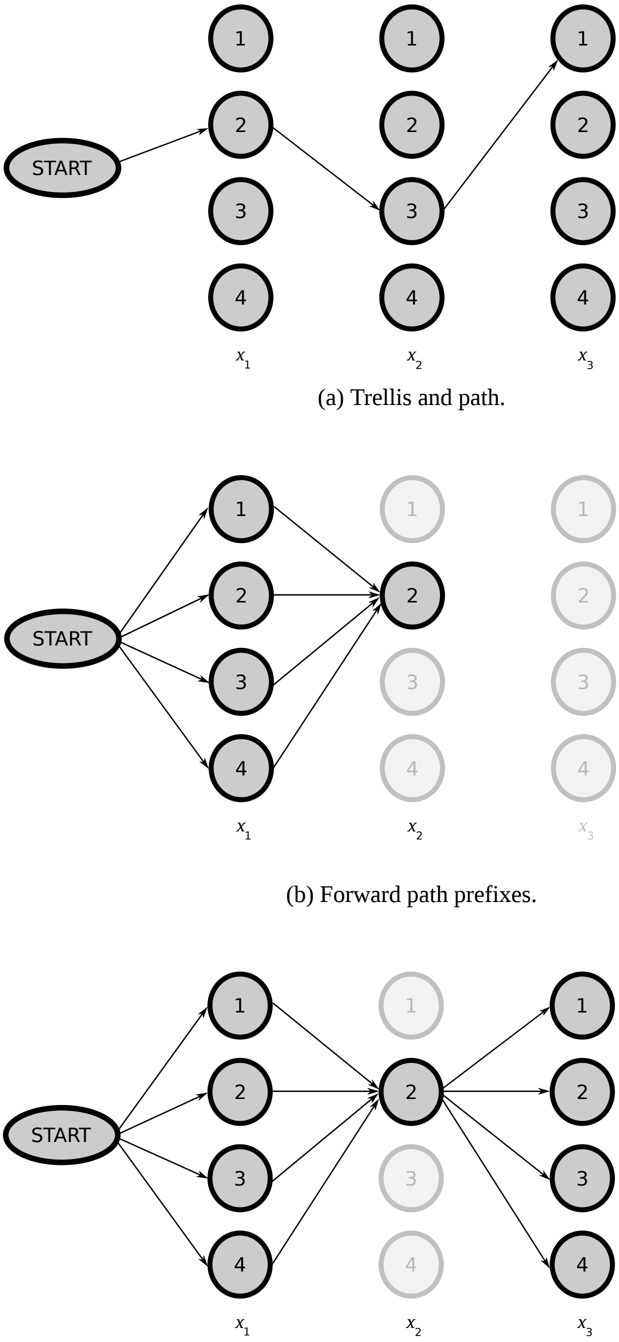
The Forward Algorithm Equations (4.5) and (4.6) reveal, that both MAP and MM inference require knowledge of the entire observation x. In the weather prediction example, observations are, however, always infinite. What kind of inference is possible in this case?

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

Figure 4.4: foo.



(a) Trellis and path.

(b) Forward path prefixes.

(c) Marginal paths.

Figure 4.5: foo

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

(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, 1989), which is in fact simply a recursive application of the right 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

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

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 6.1. 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

Algorithm 4.1: The forward algorithm in Python 3.

```python
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 consequtive 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
```

z.

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

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.

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

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 POS tags in a very large label set or when they represent combinations of POS tags. 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 POS 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 $o(|Y| \cdot \log|Y|)$. The log factor stems from the fact that the histories need to be sorted before maximization.[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.

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

[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.

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.

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.

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

Sparse Forward-Backward Algorithms FIXME (Pal et al., 2006).

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 (1989) is nearly completely unsupervised: the HMM is trained exclusively from observations. Some supervision is nevertheless usually required to determine the number of hidden states.[7] 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, 1989), which is an instance of the *expectation maximization algorithm* (EM) (Dempster et al., 1977) for HMMs.

[7] Although methods for determining the number of states from the data exist (**?**).

In POS tagging and morphological disambiguation, 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.[8] 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.

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. 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: foo

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:

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

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

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

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 compbinations 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 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 is made up from rare words. The problem therefore has very real consequences.

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

Smoothing of emission proabilities 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 sophisticated interpolation schemes can be used for example Kneser-Ney (**?**).

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 all words occurring one time in the training corpus and use these statistics for OOV words (**?**). Lidstone smoothing is another similar approach (**?**). 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. Huang et al. (2007) smooth symbol emission probabilities using geometric mean.

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%.

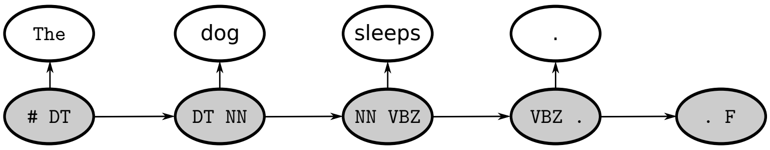


Figure 4.7: foo

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 are instead called trigram taggers.

Halácsy et al. (2007), show that it is possible to maintain the correct HMM formulation over-come the data sparseness 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. Even using smoothing, third and higher order models tend to generalize to unseen data more poorly than lower order models because of over-fitting (**?**).

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 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 some 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]

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]

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

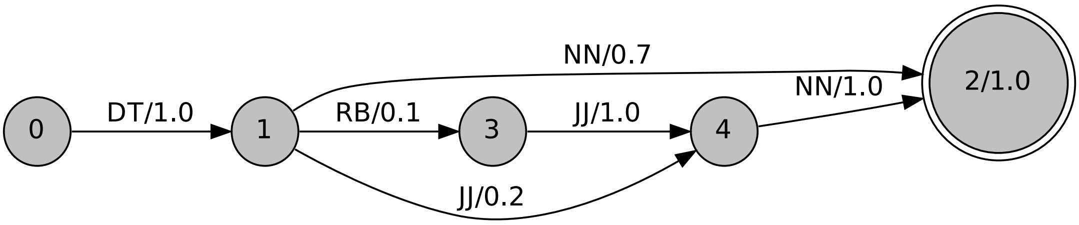


Figure 5.1: A finite-state machine accepting a subset of the singular noun phrases in Penn Treebank.

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 $\mathbb{1}$. The multiplication should distribute over the addition (Allauzen et al., 2007).

As noted above, the prototypical example of a weight semiring is given by the positive real numbers together 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, $-\log(\exp(-x) + \exp(-y)) \approx y$ when $x \gg y$. Therefore, the addition operation in the log semiring is 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

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

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	$\bigoplus_{n=0}^{\infty} M^n$	
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).

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.

N-Best

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 (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.

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 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. The sentence, emission and different models are combined using the operations of finite-state algebra.

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. Then we can construct a very simple finite-state machine X_t which recognizes the word x_t and one of its possible labels $y_t \in Y_t$ 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, $p(x_t \mid y_t)$ 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 by concatenation as shown in Figure 5.3. The paths through the sentence model correspond to the possible label assignments of sentence x.

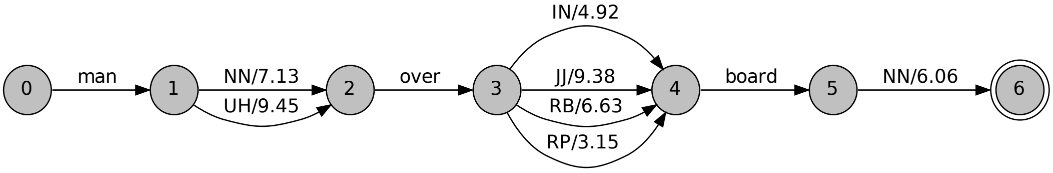


Figure 5.3: An example of a sentence model. The weights on the arcs are negative logarithms of emission probabilities. Only a subset of the possible labels are shown in the picture.

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 label n-gram $y_1, ..., y_{n+1}$.

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 n-th HMM encodes the likelihood of label sequences. I will first consider the problem of constructing a machine which represents transitions 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 the sequences occurring 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 will receive an identical penalty weight $-\log(1/(N + 1))$, where N is the size of the training corpus. Additionally, Publication **II** uses interpolation of probability estimates of n-grams of different orders. More sophisticated smoothing schemes could also be applied at the cost of larger model sizes.

[WRITE ABOUT DETERMINIZATION AND MINIMIZATION]

If the machine T is deterministic and has one path corresponding to each label n-gram $y_1, ..., y_{n+1}$, where $y \in \mathcal{Y}$, each non-terminal state in the machine will have $\mathcal{Y}$ transitions. Because 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, most label n-grams will have the same weight.

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

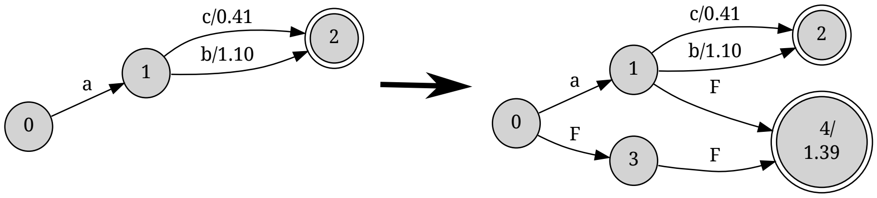


Figure 5.4: Failure transitions (with symbol F) are added to a bigram model.

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, most label n-grams will have the same weight.

In order to, reduce the size of the model, so called *failure transitions* can be used (**?**). 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. Using failure transitions, most states will not have $|\mathcal{Y}|$ transitions. Because of data sparsity, most state will in fact have only 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. $\tau_{T_f}(q, a) = \tau_T(q, a)$ 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: $\tau_{T_f}(q, f) = \{(s_{n(q)}, \mathbb{1})\}$.

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.

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.

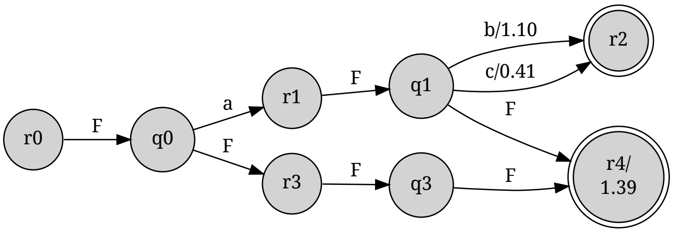


Figure 5.5: The transition model in Figure 5.4 is augmented with failure transitions and states in order to be able to handle word forms.

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[4]

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

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

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 values $r_2(y_i|y_{i-1}, y_{i-2})$, $r_1(y_i|y_{i-1})$, $r_0(y_i)$ 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 usual definition $p(x_t \mid y_t)$ 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 **?** 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 $p(y_i, x_i)$.[5]

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.

[5] 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 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.

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

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 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.

[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.

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	Example
The current word is $w \in W$	The current word is "dog"
The current word has prefix $p \in P$	The current word has prefix "d-"
The current word has suffix $s \in S$	The current word has suffix "-og"
The current word contains a digit	
The current word contains a hyphen	
The current word contains an upper case letter	
The previous word is $w \in W$	The previous word is "The".
The next word is $w \in W$	The next word is "eats".
The word before the previous word is w	
The word after the next word is w	

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

Feature template	Example
The label of the previous word is y	The label of the previous word is "NN".
The label of the previous two words are y' and y	The labels of the two previous words are "DT" and "NN".

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, $I(f)$ 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 $\omega \in \mathbb{R}^N$, the probability $p(y|x)$ is

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

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

6.2 Logistic Regression

The simplest Conditional Random Field is the *Logistic Regression Model*. 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 f 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 $p(y \,|\, x)$ 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 $p(y \,|\, x; \theta)$ is a positive real and the quantities sum to 1.

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

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.

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

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 $x \mapsto -\log(x)$.

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

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 $\mathcal{L}$.

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

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 *empirical risk minimization*, 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 $p(y \,|\, x; \theta)$ 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 $\theta \mapsto p(y \,|\, x; \theta)$ is smooth.

2. The function $\theta \mapsto p(y \,|\, x; \theta)$ is convex.

3. There exists a *unique* θ maximizing the likelihood of the training data $\mathcal{D}$.[4]

4. The model $p(y \,|\, x; \theta)$ is maximally unbiased.

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 inequality . Property 3 is a consequence of properties 1 and 2. Property 4 is discussed in **?**.

Although the maximization in Equation 6.1 cannot be solved exactly in general, the convexity and smoothness of $p(y \,|\, x; \theta)$ 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 estimatiom) require information about the partial derivatives of the loss function. Therefore the partial derivatives $\partial \mathcal{L}(\theta; \mathcal{D}) / \partial \mathrm{i}$ 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.

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

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.

6.3 The Perceptron Classifier

The perceptron algorithm (Rosenblatt, 1958) is an alternative to empirical risk minimization 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 multi-class 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 incorectly 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.

[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 $(x, y) \in \mathcal{D}$.

Estimation The perceptron algorithm is an error-driven online learning algorithm. 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 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] 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 (**?**)

Algorithm 6.1: The 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
```

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 for the label y_i in position t $s(x, y_{t-2}, y_{t-1}, y_t, t; \theta)$ 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 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 model. 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

[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.

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.

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 pseudlikelihood Besag (1975) and piecewise pseudolikelihood

[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).

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 equation 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

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

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 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 neighoring 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 propsed 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 techiques 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 develpment 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 rich 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 criterions 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 picewise 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 evaluation 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 rich 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.

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

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, $[-s \to \varepsilon]$, $[-gs \to -g]$, $[-ogs \to -og]$. 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 $[-s \to \varepsilon]$. 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.

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 rich 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
size	13,572 sent. (183,118 tok.)	19,121 sent. (162,028 tok.)
# labels	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 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.

[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.

- 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.

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,

[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

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 leas 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.

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.

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.

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.

8.5 Model Order

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

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.

[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.

Model Order	Tagging Accuracy (%)	Training time (min)	Dec. Speed (KTok/s)
Without a Morphological Analyzer			
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			
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

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.

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.

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

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 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 con-

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

verge 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.

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.

	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.

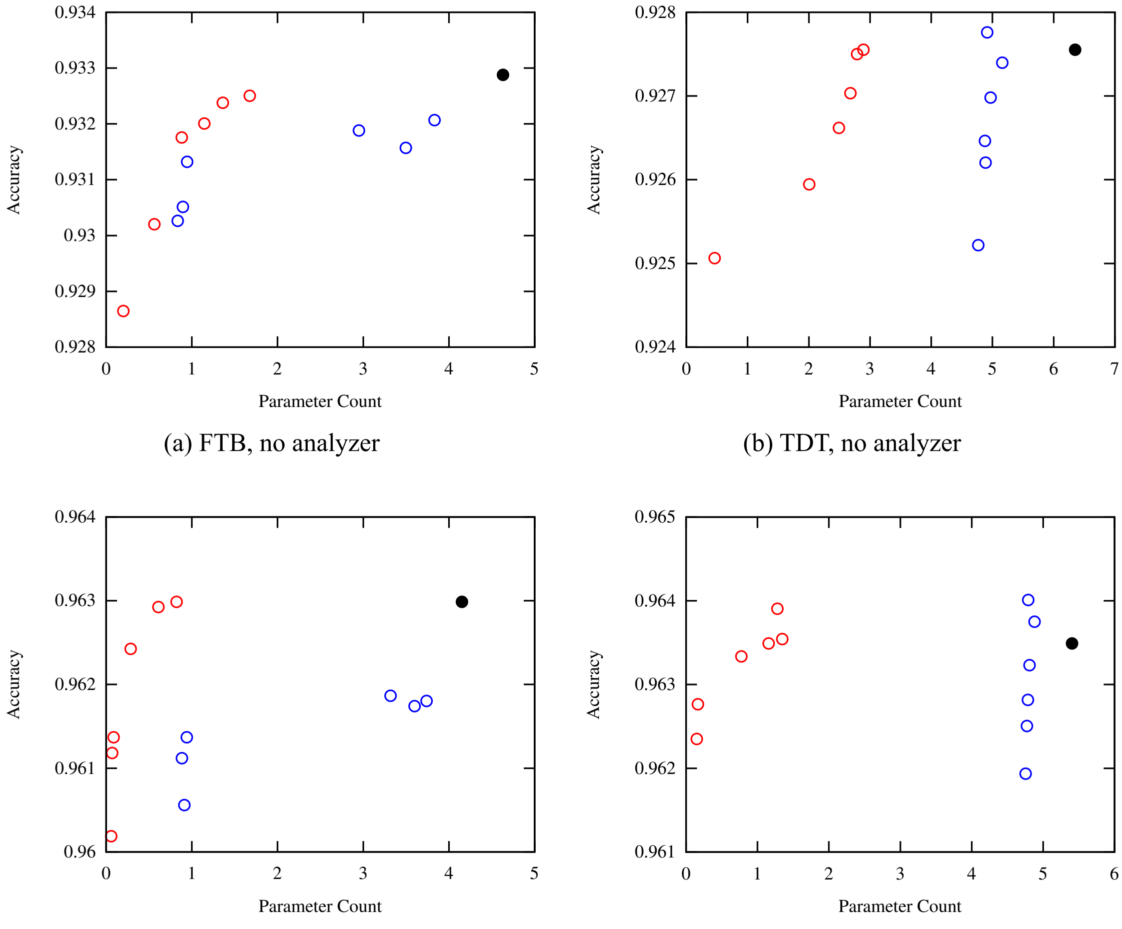
When using a morphological analyzer, even further feature pruning is possible. For FTB, 84% of model parameters can be pruned while maintaining full accuracy when using mass based pruning. When using update count based pruning, however, no parameters can be pruned without losing accuracy. For TDT, update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



(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.

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.

Chapter 9

Conclusions and Future Work

Future Work:

- Including disambiguation rules for Finnish.
- Further feature engineering especially in the form of linguistically relevant gazzetteers.
- Semi-supervised approach using self-training.
- Other possible semi-supervised approaches.
- Using morfessor instead of a morphological analyzer.
- Using a data-driven morphological analyzer instead of a regular one.

For future work, 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*, pages 569–578.

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. (2007). Openfst: A general and efficient weighted finite-state transducer library.

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*, ACL ’98, pages 86–90, Stroudsburg, PA, USA. Association for Computational Linguistics.

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

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

Bohnet, B., Nivre, J., Boguslavsky, I., Farkas, R., Ginter, F., and Hajič, 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 Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI’98, pages 33–42, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Brants, T. (2000). Tnt - a statistical part-of-speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing (ANLP-2000)*, Seattle, WA.

Brill, E. (1992). A simple rule-based part of speech tagger. In *Proceedings of the third conference on Applied natural language processing*, ANLC ’92, pages 152–155, Stroudsburg, PA, USA. 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*, ACL ’00, pages 286–293, Stroudsburg, PA, USA. Association for Computational Linguistics.

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

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

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*, ACL ’05, pages 173–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

Chrupala, G., Dinu, G., and van Genabith, J. (2008). Learning morphology with morfette. In Calzolari, N., Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., and Tapias, D., editors, *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC’08)*, Marrakech, Morocco. European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2008/.

Church, K. (2005). The DDI approach to morphology. pages 25–34.

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*, ANLC ’88, pages 136–143, Stroudsburg, PA, USA. Association for Computational Linguistics.

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - Volume 10*, EMNLP ’02, pages 1–8, Stroudsburg, PA, USA. 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*, ACL ’04, Stroudsburg, PA, USA. Association for Computational Linguistics.

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

Creutz, M. and Lagus, K. (2002). Unsupervised discovery of morphemes. In *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning - Volume 6*, MPL ’02, pages 21–30, Stroudsburg, PA, USA. 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*, ANLC ’92, pages 133–140, Stroudsburg, PA, USA. 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. *Comput. Linguist.*, 14(1):31–39.

Forney, G. D. J. (2005). The viterbi algorithm: A personal history.

Francis, W. N. (1964). *A standard sample of present-day English for use with digital computers*. Distributed by ERIC Clearinghouse [Washington, D.C.].

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 Comput.*, 4(1):1–58.

Goldberg, Y. and Elhadad, M. (2011). Learning sparser perceptron models. *Technical report*.

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 on Interactive Poster and Demonstration Sessions*, ACL '07, pages 209–212, Stroudsburg, PA, USA. 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 (GSCL-2015)*, Essen, Germany.

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 (ACL)*.

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 HLT '12, pages 142–151, Stroudsburg, PA, USA. 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*, NAACL-Short '09, pages 213–216, Stroudsburg, PA, USA. Association for Computational Linguistics.

Huang, Z., Harper, M., and Wang, W. (2007). Mandarin part-of-speech tagging and discriminative reranking. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 1093–1102, Prague, Czech Republic. 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, Gothenburg, Sweden. Association for Computational Linguistics.

Hulden, M. and Francom, J. (2012). Boosting statistical tagger accuracy with simple rule-based grammars. In Chair), N. C. C., Choukri, K., Declerck, T., Doğan, M. U., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA).

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, Oslo, Norway. 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.

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, Berlin / New York.

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 ACL-02 conference on Empirical methods in natural language processing - Volume 10*, EMNLP '02, pages 9–16, Stroudsburg, PA, USA. Association for Computational Linguistics.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, Cambridge, Massachusetts, USA.

Koskenniemi, K. (1984). A general computational model for word-form recognition and production. In *COLING-84*, pages 178–181, Stanford University, California, USA. 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*, ICML '01, pages 282–289, San Francisco, CA, USA. 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 *sfcm 2009*, pages 28—47.

Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Math. Program.*, 45(3):503–528.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330.

Merialdo, B. (1994). Tagging english text with a probabilistic model. *Comput. Linguist.*, 20(2):155–171.

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 Màrquez, L., Callison-Burch, C., Su, J., Pighin, D., and Marton, Y., editors, *EMNLP*, pages 2268–2274. The 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 (EMNLP 2013)*, pages 322–332, Seattle, Washington, USA.

Nguyen, N. and Guo, Y. (2007). Comparisons of sequence labeling algorithms and extensions. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 681–688, New York, NY, USA. ACM.

Orosz, G. and Novák, A. (2013). Purepos 2.0: a hybrid tool for morphological disambiguation. In Angelova, G., Bontcheva, K., and Mitkov, R., editors, *RANLP*, pages 539–545. RANLP 2013 Organising Committee / ACL.

Östling, R. (2013). Stagger: an open-source part of speech tagger for swedish. *Northern European Journal of Language Technology (NEJLT)*, 3:1–18.

Pal, C., Sutton, C., and McCallum, A. (2006). Sparse forward-backward using minimum divergence beams for fast training of conditional random fields. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 5, pages V–V.

Pearl, J. (1982). Reverend bayes on inference engines: a distributed hierarchical approach. In *in Proceedings of the National Conference on Artificial Intelligence*, pages 133–136.

Pirinen, T., Silfverberg, M., and Lindén, K. (2012). Improving finite-state spell-checker suggestions with part of speech n-grams. In *13th International Conference on Intelligent Text Processing and Computational Linguistics CICLing 2014*.

Pirinen, T. A. (2011). Modularisation of finnish finite-state language description—towards wide collaboration in open source development of a morphological analyser. In *Nodalida 2011*, pages 299—302.

Porter, M. F. (1997). Readings in information retrieval. chapter An Algorithm for Suffix Stripping, pages 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286. IEEE.

Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP, Pennsylvania, USA. 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, Philadelphia, PA, USA. AAI9840230.

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, Oslo, Norway. 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, volume 2: Short Papers*, pages 74–78, Gothenburg, Sweden. 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*, ACL '98, pages 246–253, Stroudsburg, PA, USA. 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 (NODALIDA 2009)*, pages 174–181.

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*, NEALT Proceedings Series. Northern European Association for Language Technology.

Silfverberg, M. and Lindén, K. (2010). Part-of-speech tagging using parallel weighted finite-state transducers. In Loftsson, H., Rögnvaldsson, E., and Helgadóttir, S., editors, *Proceedings of the 7th International Conference on NLP, IceTAL 2010*, 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 (Volume 2: Short Papers)*, pages 259–264, Baltimore, Maryland. 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.

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*, HLT '05, pages 475–482, Stroudsburg, PA, USA. 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: Short Papers - Volume 2*, HLT '11, pages 48–52, Stroudsburg, PA, USA. 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*, EACL '09, pages 763–771, Stroudsburg, PA, USA. 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*, ACL '07, pages 67–74, Stroudsburg, PA, USA. 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*, ICML '07, pages 863–870, New York, NY, USA. ACM.

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*, ANLC ’97, pages 64–71, Stroudsburg, PA, USA. Association for Computational Linguistics.

Taskar, B., Guestrin, C., and Koller, D. (2004). Max-margin markov networks. In *Advances in Neural Information Processing Systems (NIPS 2003)*, Vancouver, Canada. Winner of the Best Student Paper Award.

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*, NAACL ’03, pages 173–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res.*, 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*, pages 28–33.

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*, ICML ’06, pages 969–976, New York, NY, USA. ACM.

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*, EACL ’95, pages 157–164, San Francisco, CA, USA. 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 Constraint Grammar Applications*, pages 41–49, Riga, Latvia.

Weiss, D. J. and Taskar, B. (2010). Structured prediction cascades. In Teh, Y. W. and Titterington, D. M., editors, *AISTATS*, volume 9 of *JMLR Proceedings*, pages 916–923. JMLR.org.

Weiss, Y. (2000). Correctness of local probability propagation in graphical models with loops. *Neural Comput.*, 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, Gothenburg, Sweden. Association for Computational Linguistics.

Part-of-Speech Tagging using Parallel Weighted Finite-State Transducers

Miikka Silfverberg and Krister Lindén

Department of Modern Languages
University of Helsinki
Helsinki, Finland
miikka.silfverberg,krister.linden@helsinki.fi

Abstract. We use parallel weighted finite-state transducers to implement a part-of-speech tagger, which obtains state-of-the-art accuracy when used to tag the Europarl corpora for Finnish, Swedish and English. Our system consists of a weighted lexicon and a guesser combined with a bigram model factored into two weighted transducers. We use both lemmas and tag sequences in the bigram model, which guarantees reliable bigram estimates.

Key words: Weighted Finite-State Transducer,Part-of-Speech Tagging, Markov Model,Europarl

1 Introduction

Part-of-Speech (POS) taggers play a crucial role in many language applications such as parsers, speech synthesizers, information retrieval systems and translation systems. Systems, which need to process a lot of data, benefit from fast taggers. Generally it is easier to find faster implementations for simple models than for complex ones, so simple models should be preferred, when tagging speed is crucial.

We demonstrate that a straightforward first order Markov model, is sufficient to obtain state-of-the-art accuracy when tagging English, Finnish and Swedish Europarl corpora [Koehn 2005]. The corpora were tagged using the Connexor fdg parsers [Järvinen et al. 2004] and we used the tagged corpora both for training and as a gold standard in testing. Our results indicate that bigram probabilities yield accurate tagging, if lemmas are included in POS analyzes.

Our model consists of a weighted lexicon, a guessing mechanism for unknown words, and two bigram models. We analyze each word in a sentence separately using the weighted lexicon and guesser. The analyzes are then combined into one acyclic minimal weighted finite-state transducer (WFST), whose paths correspond to possible POS analyzes of the sentence. The paths in the sentence WFST are re-scored using the bigram models.

The bigram models assign weights for pairs of successive word forms and corresponding POS analyzes including lemmas. One of the models assigns weight for POS analyzes of word form bigrams starting at even positions in the sentence and the other one assigns weights for bigrams starting at odd positions. Both bigram models are implemented as WFSTs.

The sentence WFST and bigram model WFSTs are combined using weighted intersecting composition [Silfverberg and Lindén 2009], which composes the sentence WFST with the simulated intersection of the bigram models. Finally the POS analysis of the sentence is obtained using a best paths algorithm [Mohri and Riley 2002]. The WFSTs and algorithms for parsing were implemented using an open source transducer library HFST [Linden et al. 2009].

The paper is structured as follows. We first review earlier relevant research in section 2. We then formalize the POS tagging task in section 3 and present our model for a POS tagger as an instance of the general formulation in section 4. In section 5 we demonstrate how to implement the model using WFSTs.

The remainder of the paper deals with training and testing the POS tagger. We present the corpora and parsers used in training and tests in section 6, describe training of the model in section 7, evaluate the implementation in section 8 and analyze the results of the evaluation and present future research directions in section 9. Lastly we conclude the paper in section 10.

2 Previous Research

Statistical POS tagging is a common task in natural language applications. POS taggers can be implemented using a variety of statistical models including Hidden Markov Models (HMM) [Church 1999] [Brants 2000] and Conditional Random Fields [Lafferty et al. 2001].

Markov models are probably the most widely used technique for POS tagging. Some older systems such as [Cutting 1992] used first order models, but the accuracies reported were not very good. E.g. [Cutting 1992] report an accuracy of 96 % for tagging English text. Newer systems like [Brants 2000] have used second order models, which generally lead to better tagging accuracy. [Brants 2000] reports accuracy of 96.46% for tagging the Penn Tree Bank. More recent second order models further improve on accuracy. [Collins 2002] reports 97.11% accuracy and [Shen et al. 2007] 97.33% accuracy on the Penn Tree Bank.

We use lemmas in our bigram model as did [Thede and Harper 1999], who used lexical probabilities in their second order HMM for tagging English and obtained improved accuracy (96% – 97%) w.r.t. a second order model using plain tag sequences. In contrast to this, our model uses only bigram probabilities and it is not an HMM, since we only use frequency counts of POS analyzes for word pairs. In addition we split our bigram model into two components, which reduces its size thus allowing us to use a larger training material.

The idea of syntactic parsing and POS tagging using parallel finite-state constraints was outlined by [Koskenniemi 1990]. The general idea in our system is the same, but instead of a rule-based morphological disambiguator, we implement a statistical tagger using WFSTs. Still, hand-crafted tagging constraints could be added to the system.

3 Formulation of the POS Tagging Task

In this section we formulate the task of Part-of-Speech (POS) tagging and describe probabilistic POS taggers formally.

By a sentence, we mean a sequence of syntactic tokens $s = (s_1 \ ... \ s_n)$ and by a POS analysis of the sentence s, we mean a sequence of POS analyzes $t = (t_1, \ ..., \ t_n)$. We include lemmas in POS analyzes. For each i, the analysis t_i corresponds to the token s_i in sentence s. We denote the set of all sentences by S and the set of all analyzes by T.

A *POS tagger* is a machine which associates each sentence s with its most likely *POS analysis* t_s. To find the most likely POS analyzes for the sentence s, the model estimates the probabilities for all possible analyzes of s using a distribution P. For the sentence s and every possible POS analysis t, the distribution P associates a probability $P(t, s)$. Keeping t fixed, the mapping $s \mapsto P(t, s)$ is a normalized probability distribution. The most likely analysis t_s of the sentence s is the analysis which maximizes the probability $P(t, s)$, i.e.

$$t_s = \arg\max_t P(t, s).$$

The distribution P can consist of a number of component distributions P_i, each giving probability $P_i(s, t)$ for sentence s and analysis t. The component probabilities are combined using some function $F : [0, 1]^n \to [0, 1]$ to obtain

$$P(s, t) = F\big(P_i(t, s), \ ..., \ P_n(t, s)\big).$$

The function F should be chosen in such a way that P is nonnegative and satisfies

$$\sum_{t \in T} P(t, s) = 1$$

for each sentence s.

Often a convex linear function F is used to combine estimates given by the component models. In such a case the model P is called a *linear interpolation* of the models P_i.

4 A Probabilistic First Order Model

In this section we describe the idea behind our POS tagger. We use a bigram model for POS tagging. Thus the probability of a given tagging of a sentence is estimated using analyzes of word pairs.

Since we make use of extensive training material, we may include lemmas in bigrams. Although the training material is extensive, the tagger will still encounter bigrams which did not occur in the training material or only occurred once or twice. In such cases we want to use unigram probabilities for estimating the best POS analysis. Hence we weight all analyzes using probabilities given by both the unigram and bigram models, but weight bigram probabilities heavily while only giving unigram probabilities a small weight. Hence unigram probabilities become significant only when bigram probabilities are very close to each other.

4.1 The Unigram model

The unigram model emits plain unigram probabilities $p_u(t, s_x)$ for analyzes t given a word form s_x (we use the index x to signify that $p_u(t, s_x)$ is independent of the context of the word form s_x). Unigram probabilities are readily computed from training material. The probability of the analysis $t = (t_1 \ldots t_n)$ given the sentence $s = (s_1 \ldots s_n)$ assigned by the unigram model is

$$P_u(t, s) = \prod_{i=1}^{n} p_u(t_i, s_i).$$

In practice it is not possible to train the unigram model for all possible word forms in highly inflecting languages with productive compounding mechanism such as Finnish or Turkish. Instead the probabilities for analyzes given a word form need to be estimated using probabilities for words with similar suffixes. For instance, if the word form *foresaw* was not observed during training, we can give it a similar distribution of analyzes as the word *saw* receives, since *saw* shares a three-letter suffix with *foresaw*.

In practice such estimation relying on analogy is accomplished by a so called POS guesser, which seeks words with maximally long suffixes in common with an unknown word. It then assigns probabilities for POS analyzes of the unknown word on basis of the analyzes of the known words. [Linden 2009a] shows how a guesser can be integrated with a weighted lexicon in a consistent way.

4.2 The Bigram Models

We use two bigram models Q_o and Q_e giving probabilities for bigrams starting at even and odd positions in the sentence. The estimates are built using plain bigram probabilities for tagging a word-pair s_1 and s_2 with analyzes t_1 and t_2 respectively[1]. These probabilities $p_b(t_1, s_1, t_2, s_2)$ are easily computed from a training corpus.

For an analysis $t = t_1 \ldots t_{2k}$ and a sentence $s = s_1 \ldots s_{2k}$ of even length $2k$, the models Q_o and Q_e give bigram scores

$$Q_o(t, s) = \prod_{i=1}^{k} p_b(t_{2i-1}, s_{2i-1}, t_{2i}, s_{2i}),\ Q_e(t, s) = \prod_{i=1}^{k-1} p_b(t_{2i}, s_{2i}, t_{2i+1}, s_{2i+1})$$

For an analysis $t = t_1 \ldots t_{2k+1}$ and a sentence $s = s_1 \ldots s_{2k+1}$ of odd length $2k + 1$, the models Q_o and Q_e give bigram scores

$$Q_o(t, s) = \prod_{i=1}^{k} p_b(t_{2i-1}, s_{2i-1}, t_{2i}, s_{2i}),\ Q_e(t, s) = \prod_{i=1}^{k} p_b(t_{2i}, s_{2i}, t_{2i+1}, s_{2i+1})$$

[1] In literature, it is often suggested that one should instead compute probabilities of word form bigrams given POS analysis bigrams. We cannot do this, since we include lemmas in POS analyzes. This makes the probability of a word form given a POS analysis either 0 or 1 since most analyzes only have one realization as a word form.

4.3 Combining the Unigram and Bigram Models

The standard way of forming a model from P_u, Q_o and Q_e would be to use linear interpolation. We do not want to do this, since we aim to convert probabilities into penalty weights in the tropical semiring using the mapping $p \mapsto -\log p$, which is not compatible with sums. Instead we take a weighted product of powers of the component probabilities. Hence we get a model

$$P(t, s) = P_u(t, s)^{w_u} Q_o(t, s)^{w_o} Q_e(t, s)^{w_e}$$

where w_u, w_e and w_o are parameters, which need to be estimated.

If each of the models P_u, Q_e and Q_o agree on the probability p of an analysis t given a sentence s, we want P to give the same probability. This is accomplished exactly when $w_u + w_e + w_o = 1$. There does not seem to be any reason to prefer either of the models Q_e or Q_o, which makes it plausible to assume that $w_e = w_o$. Hence an implementation of the model only requires estimating two non-negative parameters: the unigram parameter w_u and the bigram parameter w_b. They should satisfy $w_u + 2w_b = 1$.

It is possible that $P(t, s)$ will not be a normalized distribution when s is kept fixed, but it can easily be normalized by scaling linearly with factor $\Sigma_t P(t, s)$. For the present implementation, it is not crucial that P is normalized.

5 Implementing the Statistical Model using Weighted Finite-State Transducers

We describe the implementation of the POS tagger model using weighted finite-state transducers (WFSTs). We implement each of the components of the statistical model as a WFST, which are trained using corpus data.

In order to speed up computations and prevent roundoff errors, we convert probabilities p, given by the models, into penalty weights in the tropical semiring using the transformation $p \mapsto -\log p$. In the tropical semiring the product of probabilities pq translates to the sum of corresponding penalty weights $-\log p + -\log q$. The kth power of the probability p, namely p^k, translates to a scaling of its weight $-k \log p$. These observations follow from familiar algebraic rules for logarithms.

In our system, tagging of sentences is performed in three stages using four different WFSTs. The first two WFSTs, a weighted lexicon and a guesser for unknown words, implement a unigram model. They produce weighted suggestions for analyzes of individual word forms. The latter two WFSTs re-score the suggestions using bigram probabilities. The weights $-\log p$ given by the unigram model and the bigram model are scaled by multiplying with a constant in order to prefer analyzes which are strong bigrams. The scaled weights $-k \log p$ are then added to give the total scoring of the input sentence. This corresponds to multiplying the powers p^k of the corresponding probabilities.

In the first stage we use a weighted lexicon, which gives the five best analyzes for each known word form. In initial tests, the correct tagging for a known word could be found among the five best analyzes in over 99% of tagged word forms, so we get sufficient coverage while reducing computational complexity.

For an unknown word x, we use a guesser which estimates the probability of analyzes using the probabilities for analyzes of known words. We find the set of known word forms W, whose words share the longest possible suffix with the word form x. We then determine the five best analyzes for the unknown word form x by finding the five best analyzes for words in the set W.

For each word s_i in a sentence $s = s_1 \dots s_n$, we form a WFST W_i which is a disjunction of its five best analyzes $t_1 \dots t_5$ according to the weights $w(s_i, t_i)$ given by the unigram model. In case there are less than five analyzes for a word, we take as many as there are. We then compute a weighted concatenation W_s of the individual WFSTs W_i. The transducer W_s is the disjunction of all POS analyzes of the sentence s, where each word receives one of its best five analyzes given by the unigram model.

To re-score the analysis suggestions given by the lexicon and the guesser, we use two WFSTs whose combined effect gives the bigram weighting for the sentence. One of the model scores bigrams starting at even positions in the sentence and the other one scores bigrams starting at odd positions. Thus we give a score for all bigrams in the sentence without having to compute a WFST equivalent to the intersection of the models which might be quite large.

Using weighted intersecting composition [Silfverberg and Lindén 2009] we simultaneously apply both bigram scoring WFSTs to the sentence WFST W_s. The POS analysis of the sentence s is the best path of the result of the composition.

The WFSTs and algorithms for parsing were implemented using the Helsinki Finite-State Technology (HFST) interface [Linden et al. 2009].

We now describe the lexicon, guesser and the bigram WFSTs in more detail.

5.1 The Weighted Lexicon

Using a tagged corpus, we form a weighted lexicon L which re-writes word forms to their lemmas and analyzes. POS analyzes for a word form s_i are weighted according to their frequencies, which are transformed into tropical weights.

In order to estimate the weights for words which were not seen in the training corpus, we construct a guesser. For an unknown word, the guesser will try to construct a series of analyzes relying on information about the analyzes of known similar words.

Figure 5.1 shows an example guesser, which can be constructed from a reversed weighted lexicon. Guessing begins at the end of the word. We allow guessing at a particular analysis for a word only if the word has a suffix agreeing with the analysis. See [Linden 2009a] for more information on guessers.

5.2 The Bigram Models

To re-score analyzes given by the unigram model, we use two WFSTs whose combination serves as a bigram model. The first one, B_e, scores each known

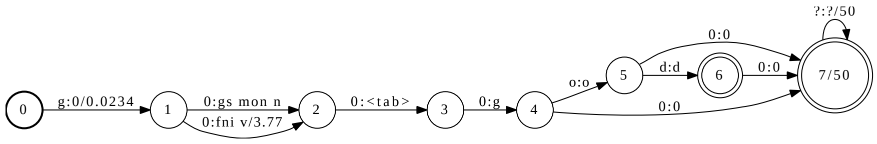


Fig. 1. Guesser constructed from a weighted lexicon. Guessing starts at the end of a word. Skipping letters gives a high penalty and analyzes, where equally many letters are skipped, are weighted according to the frequency of the analyzes.

word form/analysis bigram s_{2k}, s_{2k+1} and t_1, t_2 in the sentence starting at an even position $2k$ according to the maximum likelihood estimate of the tag bigram $t_1 t_2$ w.r.t. the word form bigram $s_{2k} s_{2k+1}$. The WFST B_o is similar to B_e except it weights bigrams starting at odd positions $s_{2k-1} s_{2k}$.

Given a word form pair s_1, s_2, we compute the probability $P(t_1, s_1, t_2, s_2)$ for each POS analysis pair t_1, t_2. These sum to 1 when w_1 and w_2 remain fixed. Then we form a transducer B, whose paths transform word form pairs $s_1 s_2$ into analysis pairs $t_1 t_2$ with weight $-\log P(t_1, s_1, t_2, s_2)$. Lastly we disjunct B with a default bigram, which transforms arbitrary word form sequences to arbitrary analyzes with a penalty weight, which is greater than the penalty received by all other transformations.

In addition to the model B, we also compute a general word model W, which transforms an arbitrary sequence of symbols into an arbitrary lemma and an analysis. The word model W is used to skip words at the beginning and end of sentences.

From the transducers above, we form the models B_e and B_o using weighted finite state operations

$$B_e = W B^* W^{\{0,1\}} \text{ and } B_o = B^* W^{\{0,1\}}.$$

Here $W^{\{0,1\}}$ signifies an optional instance of W.

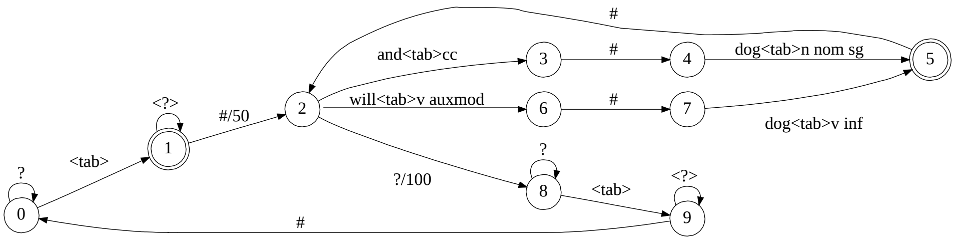


Fig. 2. A small example of an even bigram model B_e. ? signifies an arbitrary symbol and <?> signifies an arbitrary POS analysis symbol.

5.3 Parsing using Weighted Intersecting Composition

In our system, parsing a sentence S is in principle equivalent to finding the best path of the transducer

$$(S \circ L) \circ (B_e \cap B_o).$$

Since the intersection of B_o and B_e could become prohibitively large, we instead use intersecting composition [Silfverberg and Lindén 2009] to simulate the intersection of B_e and B_o during composition with the unigram tagged sentence $S \circ L$.

Intersecting composition is an operation first used in compiling two-level grammars [Karttunen 1994]. We use a weighted version of the operation.

After the intersecting composition, we extract the best path from the resulting transducer. This is the tagged sentence.

6 Data

In this section we describe the data used for testing and training the POS tagger.

For testing and training, we used the Europarl parallel corpus [Koehn 2005].

The Europarl parallel corpus is a collection of proceedings of the European Parliament in eleven European languages. The corpus has markup to identify speaker and some html-markup, which we removed to produce a file in raw text format. We used the Finnish, English and Swedish corpora. Since the training and testing materials are the same for all three languages, the results we obtain for the different languages are comparable.

We parsed the Europarl corpora using Connexor functional dependency parsers fi-fdg for Finnish, sv-fdg for Swedish and en-fdg for English [Järvinen et al. 2004]. From the parses of the corpora we extracted word forms, lemmas and POS tags. For training and testing, we preserved the original tokenization of the fdg-parsers and removed *prop* tags marking proper nouns, *abbr* tags marking abbreviations and *heur* tags marking guesses made by the fdg-parser. The tag sequence counts in table 1 represent the number of tag sequences after *abbr*, *prop* and *heur* tags were removed.

Table 1. Some figures describing the test and training material for the POS tagger.

Language	Syntactic tokens	Sentences	POS tag sequences
English	43 million	1 million	122
Finnish	25 million	1 million	2194
Swedish	38 million	1 million	243

Table 1 describes the data used in training and testing the POS tagger. We see that the fi-fdg parser for Finnish emitted more than ten times as many tag sequences as sv-fdg for Swedish or en-fdg for English. The en-fdg parse emitted clearly fewest tag sequences.

7 Training the Model

We now describe training the model, which consists of two phases. In the first phase we build the weighted lexicon and guesser and the bigram models. In the second phase we estimate experimentally coefficients w_u and w_b, which maximize the accuracy of the interpolated model

$$P(t,s) = P_u(t,s)^{w_u} Q_o(t,s)^{w_b} Q_e(t,s)^{w_b}$$

Using a small material covering 1000 syntactic tokens, we estimated $w_u = 0.1$ and $w_b = 0.45$. This shows that it is beneficial to weight the bigram model heavily, which seems natural, since bigrams provide more information than unigrams.

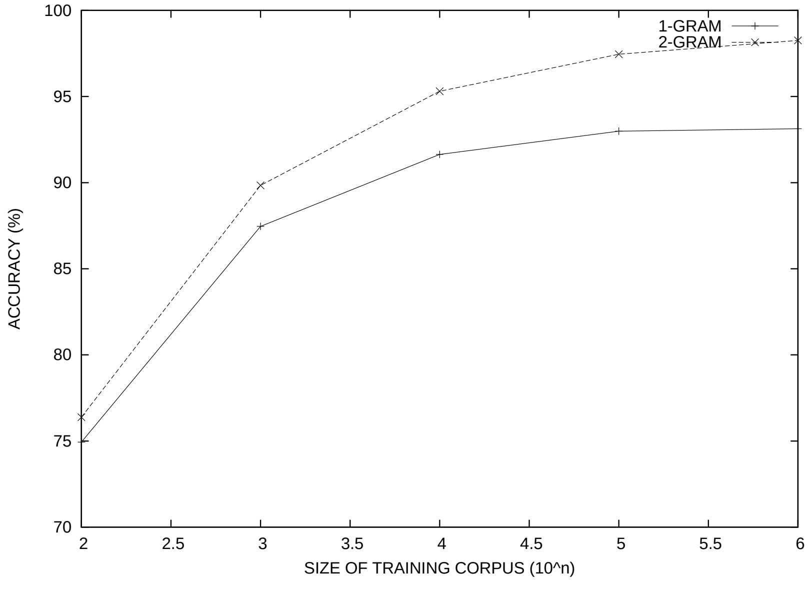


Fig. 3. The accuracy for the English POS tagger as a function of the size of training data. We used between 10^2 and 10^6 sentences for training. The lower curve displays the accuracy using only the unigram model, whilst the upper curve displays the accuracy of the combined unigram and bigram model.

Figure 3 shows learning curves for the English language POS tagger using 10^2 to 10^6 sentences for training. The lower curve displays accuracies for the unigram model and the upper curve shows the accuracy for the combined unigram and bigram model. For the unigram model, we can see that little improvement is obtained by increasing the training data from 10^4 sentences. In contrast, there is significant improvement ($\approx 0.82\%$) for the bigram model even when we move from 10^5 to 10^6 sentences.

8 Evaluation

We describe the methods we used to evaluate the POS tagger and the results we got.

We used ten-fold cross-validation to evaluate the POS tagger, that is we split the training material in ten equally sized parts and used nine parts for training the model and the remaining part for testing. Varying the tenth used for testing we trained ten POS taggers for each language.

For each of the languages we trained two sets of taggers. One set used only unigram probabilities for assigning POS tags. The other used both unigram and bigram probabilities. We may consider the unigram taggers as a baseline.

For each tree languages, we computed the average and standard deviation of the accuracy of the unigram and bigram taggers. In addition we computed the Wilcoxon matched-pairs signed-ranks test for the bigram and unigram accuracies in all three languages. The test does not assume that the data is normally distributed (unlike the paired t-test). The results of our tests can be seen in table 2.

Table 2. Average accuracies and standard deviations for POS taggers in Finnish, English and Swedish. The sixth column shows the improvement, which results for adding the bigram model. In the seventh column, we show the results of the Wilcoxon matched-pairs signed-ranks test between unigram and bigram accuracies.

Language	Unigram Acc.	σ	Bigram Acc.	σ	Diff.	Conf.
English	93.10%	0.09	98.29%	0.01	5.19%	$\geq 99.8\%$
Finnish	94.38%	0.07	96.63%	0.03	2.25%	$\geq 99.8\%$
Swedish	94.12%	0.20	97.31%	0.11	3.19%	$\geq 99.6\%$

9 Discussion and Future Work

It is interesting to see that a bigram tagger can perform equally well or better than trigram taggers at least on certain text genres. The mean accuracy 98.29%, we obtained for tagging the English Europarl corpus is exceptionally high (for example [Shen et al. 2007], report a 97.33% accuracy on tagging the Penn Tree Bank). The improvement of 5.19 percentage points from the unigram model to the combined unigram and bigram model is also impressive. There is also a clear improvement for Finnish and Swedish, when the bigram model is used in tagging and accuracy for these languages is also high. We had problems finding accuracies figures for statistical taggers of Finnish, but for Swedish [Megyesi 2001] reports accuracies between 94% and 96%, which means that we get state-of-the-art accuracy for Swedish.

Of course the Europarl corpus is probably more homogeneous than the Penn Tree Bank or the Brown Corpus, both of which include texts from a variety of genres. Furthermore tagging is easier because the en-fdg parser only emits 122 different POS analyzes. Still, Europarl texts represent an important genre, because the EU is constantly producing written materials, which need to be translated into all official languages of the union.

The accuracy for Finnish shows less improvement than English and Swedish. We believe this is a result of the fact that Finnish words carry a lot of information but the bonds between words in sentences may be quite weak. This conclusion is supported by the fact that unigram accuracy for Finnish is best of all three languages.

We do not believe, that using trigram statistics would bring much improvement for Finnish. Instead we would like to write a set of linguistic rules which would cover most typically occurring tagging errors. Especially we would like to try out constraints, which would mark certain analyzes as illegal in some contexts. Such negative information is hard to learn using statistical methods. Still, it may be very useful, so it could be provided by hand-crafted rules.

Clearly our figures for accuracy need to be considered in relation to the tagging accuracy of the fdg parsers. We did not succeed in finding a study on the POS tagging accuracy of the fdg parsers. Instead we examined the POS tagging for one word per twenty thousand in the first tenth of the Europarl corpora for Finnish, English and Swedish. This amounted to 131 examined words for Finnish, 219 examined words for English and 191 examined words for Swedish. According to these tests, the POS tagging accuracy of the fdg parsers for Finnish is 95.4%, for English it is 97.3% and for Swedish it is 97.5%.

10 Conclusions

We introduced a model for a statistical POS tagger using bigram statistics with lemmas included. We showed how the tagger can be implemented using WFSTs. We also demonstrated a new way to factor a first order model into a model tagging bigrams at even positions in the sentence and another model tagging bigrams at odd positions.

In order to test our model, we implemented POS taggers for Finnish, English and Swedish, training them and evaluating them using Europarl corpora in the respective languages and Connexor fdg parsers.

We obtained a clear, statistically significant, improvement for all three languages when compared to the baseline unigram tagger. At least for English and Swedish, we obtain state-of-the-art accuracy.

Acknowledgements

We thank the anonymous referees. We also want thank our colleagues in the Hfst team. The first author is funded by Langnet Graduate School for Language Studies.

References

[Brants 2000] Thorsten Brants: TnT – A Statistical Part-of-Speech Tagger. ANLP-2000, 2000.

[Church 1999] Kenneth Church: A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. Proceedings of the Second Conference on Applied Natural Language Processing, 1988.

[Collins 2002] Michael Collins: Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms, EMNLP, 2002.

[Cutting 1992] Doug Cutting, Julian Kupiec, Jan Pedersen and Penelope Sibun: A Practical Part-of-Speech Tagger, Proceedings of the Third Conference on Applied Natural Language Processing, 1992.

[Järvinen et al. 2004] Timo Järvinen, Mikko Laari, Timo Lahtinen, Sirkku Paajanen, Pirkko Paljakka, Mirkka Soininen and Pasi Tapanainen: Robust Language Analysis Components for Practical Applications. Robust and Adaptive Information Processing for Mobile Speech Interfaces (eds. Björn Gambäck and Kristiina Jokinen), 2004.

[Karttunen 1994] Lauri Karttunen: Constructing Lexical Transducers, COLING-94, pp. 406–411, 1994.

[Koehn 2005] Philipp Koehn: Europarl: A Parallel Corpus for Statistical Machine Translation. Machine Translation Summit X, 2005, pp. 79–86. Phuket, Thailand.

[Koskenniemi 1990] Kimmo Koskenniemi: Finite-state parsing and disambiguation, 13th COLING, 1990.

[Lafferty et al. 2001] John Lafferty, Andrew MacCallum and Fernando Pereira: Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. ICML 2001, 2001.

[Linden et al. 2009] Krister Lindén, Miikka Silfverberg and Tommi Pirinen: Hfst Tools for Morphology – an Efficient Open-Source Package for Construction of Morphological Analyzers (eds. Cerstin Mahlow and Michael Piotrowski). sfcm 2009, vol 41 in LNCS, pp 28–47, Springer, 2009.

[Linden 2009a] Krister Lindén: Entry Generation by Analogy Encoding New Words for Morphological Lexicons, vol 1 in NEJLT, 2009.

[Linden 2009b] Krister Lindén: Guessers for Finite-State Transducer Lexicons. CICling-2009, Mexico, 2009.

[Megyesi 2001] Beáta Megyesi: Comparing data-driven learning algorithms for POS tagging of Swedish. EMNLP 2001.

[Mohri and Riley 2002] Mehryar Mohri, Michael Riley 2002: An Efficient Algorithm for the n-Best-Strings Problem, ICSLP 02, 2002.

[Mikheev 1997] Andrei Mikheev: Automatic Rule Induction for Unknown-Word Guessing, CL, vol 23, 1997.

[Shen et al. 2007] Libin Shen, Giorgio Satta and Aravind Joshi: Guided Learning for Bidirectional Sequence Classification, ACL 2007, 2007.

[Silfverberg and Lindén 2009] Miikka Silfverberg and Krister Lindén: Conflict Resolution Using Weighted Rules in HFST-TwolC. NODALIDA 2009.

[Thede and Harper 1999] Scott Thede and Mary Harper: A Second-Order Hidden Markov Model for Part-of-Speech Tagging, 37th ACL, 1999.

Combining Statistical Models for POS Tagging using Finite-State Calculus

Miikka Silfverberg
Helsinki University
Helsinki, Finland
miikka.silfverberg@helsinki.fi

Krister Lindén
Helsinki University
Helsinki, Finland
krister.linden@helsinki.fi

Abstract

We introduce a framework for POS tagging which can incorporate a variety of different information sources such as statistical models and hand-written rules. The information sources are compiled into a set of weighted finite-state transducers and tagging is accomplished using weighted finite-state algorithms. Our aim is to develop a fast and flexible way for trying out different tagger designs and combining them into hybrid systems. We test the applicability of the framework by constructing HMM taggers with augmented lexical models for English and Finnish. We compare our taggers with two existing statistical taggers TnT and Hunpos and find that we achieve superior accuracy.

1 Introduction

Part-of-Speech (POS) tagging, and other sequential labeling tasks like named entity recognition and chunking, constitute core tasks of language technology. Highly successful POS taggers for English have been constructed both using rule-based methods e.g. finite-state constraints used by Voutilainen (1995) and statistical methods e.g. Hidden Markov Models (HMM) used by Brants (2000).

Besides HMMs, other statistical models such as Conditional Random Fields and Maximum Entropy Models have recently been used to construct POS taggers, but HMMs remain one of the most widely used in practice. Though the more recent models surpass HMMs in accuracy, the great tagging speed and a fast development cycle of HMMs ensure a continuing popularity.

Accuracies for state of the art statistical taggers for English newspaper text surpass 97%, but results for applying these models on other languages are not always as encouraging. E.g. Dredze and Wallenberg (2008) report an accuracy 92.06% on tagging Icelandic using bidirectional sequence classification.

Low accuracy is partly due to the lack of sufficiently large tagged corpora, which can be used as training material. Reduction of accuracy can also result from the fact that the syntax and morphology of many languages differ substantially from English syntax and morphology. E.g. many languages do not have as rigid word order as English and many languages incorporate far more extensive morphological phenomena. Thus models, which have been developed for English, may not work well on many other languages.

These practical and theoretical problems associated with constructing POS taggers for virtually all of the world's languages, demonstrate the need for POS tagging models, which can incorporate a variety of different information sources including different kinds of statistical models but also more linguistic models like the ones utilized by Voutilainen (1995). Ideally the linguistic models could be used to fine-tune the result of the statistical tagging.

We propose a general framework for building POS taggers, where various kinds of statistical models and other POS tagging models can be combined using weighted finite-state calculus. Using this framework, developers can test a variety of models for tagging a language and apply the models in parallel. E.g. a statistical model trained with insufficient training data can be augmented with hand-made or machine-learned rules for common tagging errors.

In order to test the framework, we trained standard second order HMMs for Finnish and English and augmented these with extended lexical models using tag context.

We train and evaluate the English tagger using the Wall Street Journal (WSJ) corpus from Penn

Bolette Sandford Pedersen, Gunta Nešpore and Inguna Skadiņa (Eds.)
NODALIDA 2011 Conference Proceedings, pp. 183–190

Treebank II (Marcus et al., 1994). We compare the accuracy obtained by our model with the well known and widely used HMM tagger TnT (Brants, 2000) and a more recent open-source HMM tagger Hunpos (Halácsy et al., 2007), which also utilizes an extended lexical model. After improving upon the lexical model of Hunpos, our tagger obtains an accuracy of 96.67%, outperforming both TnT (96.46%) and Hunpos (96.58%).

For training and testing the Finnish tagger, we use morphologically analyzed and disambiguated newspaper text. The optimization of the model for Finnish requires some changes in the model. Together these changes improve the accuracy from a baseline second order HMM by more than 1%. We also train a Hunpos tagger for Finnish and compare it with our own tagger. The 96.02% accuracy, we obtain on the Finnish material, clearly outperforms Hunpos (95.62%).

We implemented all taggers using the freely available HFST-interface for weighted finite-state transducers (Lindén et al., 2009). An open-source interface for constructing taggers in our framework will be made publicly available.

This paper is structured in the following way. We first review some earlier work on enhancing the accuracy of HMMs. We then introduce our framework for constructing taggers in section 4. In section 5 we introduce an HMM tagger augmented with contextual lexical probabilities, which we implemented for English and Finnish in our framework. We then evaluate the English and Finnish taggers using corpus data and compare them with TnT and Hunpos. Following evaluation, we present a brief discussion on our results and future work. Finally we conclude the paper.

2 Previous Work

Statistical POS tagging is a common task in natural language applications. POS taggers can be implemented using a variety of statistical models including Hidden Markov Models (HMM) (Church, 1999; Brants, 2000), Maximum Entropy Models (Tsuruoka et al., 2005) and Conditional Random Fields (Lafferty et al., 2001).

HMMs are probably the most widely used technique for POS tagging and one of the best known imlpementations of an HMM is TnT by Brants (2000). When tagging the WSJ corpus using the splits introduced by Collins (2000), TnT achieves an accuracy of 96.46%. Although more recent statistical techniques result in improved accuracy, HMMs have remained in use chiefly because of the speed of both developing a tagger and tagging.

Recently Banko and Moore (2004) and Halácsy et al. (2007) have worked on improving the accuracy of HMMs by adding tag context into the lexical model of the HMM. The technique was pioneered by Toutanova et al. (2003) in the context of Conditional Markov Models.

The strength of Banko and Moore (2004) is that their lexical models use both left and right context when determining the conditional probability which should be associated to a wordform given a tag. The Hunpos tagger by Halácsy et al. (2007) uses only the left tag context, but it does not require a full lexicon, which makes it very practical.

We combine the left and right tag context in lexical models with a guesser for unknown wordforms. Our approach differs from Hunpos in that we only use contextually dependent lexical probabilities for known words.

Besides evaluating our approach to POS tagging by constructing a tagger for English text, we also test our approach on Finnish. Work with statistical POS tagging for Finnish seems to be virtually non-existent. Silfverberg and Lindén (2010) derive a Finnish POS tagger for the Finnish Europarl corpus (Koehn, 2005), which achieves high accuracy i.e. 96.63%, but these results could be contested on the grounds that the Europarl corpus is translated into Finnish from other languages. Silfverberg and Lindén (2010) also use an extremely large (25 million tokens) corpus. We use Finnish newspaper text to train and evaluate the tagger. Our training corpus is comparable in size to the Wall Street Journal corpus.

3 Note on Terminology

We use the terms *analysis*, *POS tag* and *tag* interchangeably to refer to POS tags, which are given for words. The *correct tag* or *analysis* refers to the intended analysis of a word in a gold standard corpus. By the term an *analysis of a sentence*, we signify one possible way to assign a unique POS tag to each of the words in the sentence. We use the term *correct analysis of a sentence* to denote the unique analysis where all of the words receive their correct analyses.

The term *analysis or tag profile of a word* refers to the set of tags which can occur as its POS analyses.

If all of the analyses of a sentence are compiled into a transducer, the paths of the transducer correspond exactly to the analyses of the sentence. In this setting, we use the terms analysis and path interchangeably. We call the transducer, compiled from the tag profiles and associated probabilities, the *sentence transducer*.

4 A Framework for Constructing POS Taggers

Our framework factors POS tagging into two tasks: (i) assigning tag profiles and probabilities $p(w|t)$ to each word w in a sentence and each of its possible analyses t and (ii) re-scoring the different analyses of the entire sentence using parallel weighted models for word and tag sequences.[1]

In the first task, the tag profile for a word w and the probabilities $p(w|t)$ for each of its tags is estimated from a training corpus. The probabilities are independent of surrounding words and tags. For unknown words u, a number of guessers can be included. These estimate the probabilities $p(u|t)$ using the probabilities $p(s|t)$ for the suffixes of u. The suffix probabilities can be estimated from a training corpus.

A number of guessers can be used to estimate the distribution of analyses for different kinds of unknown words. Like Hunpos and TnT, we always include different guessers for upper case words and lower case words, which improves accuracy.

The tag profiles of words along with tag probabilities are compiled into a weighted finite-state transducer, which associates a probability for every possible analysis of the sentence. The probability assigned to a path at this stage is the product of lexical probabilities.

After assigning tag profiles and probabilities for words, the second task is to re-score the paths of the sentence transducer. Different models can be used to accomplish this. Each of the models adds some weight to each of the analyses of the sentence and their combined effect determines the best path i.e. the most probable path. We could also incorporate models which forbid some analyses. This means that the analyses are discarded in favor of other analyses which initially seemed less likely. Such models could be used to correct systematic errors stemming from the statistical models.

The result of applying the re-scoring models to the tag profiles is computed using weighted intersecting composition by Silfverberg and Lindén (2009). After re-scoring, a best paths algorithm (Mohri and Riley, 2002) is used to extract the most probable analysis for the sentence.

5 Augmented HMM POS Tagger for English and Finnish

For English and Finnish we constructed POS taggers based on traditional second order HMMs augmented with models, which re-score lexical probabilitites according to tag context (this is the factor $p(w_i|t_{i-1}, t_i, t_{i+1})$ in the formula below). For the sentence w_i, ..., w_n, the taggers attempt to maximize the probability $p(t_1, ..., t_n|w_i, ..., w_n)$ over tag sequences t_1, ..., t_n. Because of the data sparseness problem, it is impossible to compute the probability directly, so the tagger instead maximizes its approximation

$$\prod_{i=1}^{n} p(t_i|t_{i-1}, t_{i-2}) p(w_i|t_{i-1}, t_i, t_{i+1}) p(w_i|t_i)$$

where the tag sequences t_1 ..., t_n ranges over all analyses of the sentence. The term $p(t_i|t_{i-1}, t_{i-2})$ is the standard second order HMM approximation for the probability of the tag t_i. The term $p(w_i|t_{i-1}, t_i, t_{i+1})$ conditions the probability of the word w_i on its tag context. Finally the term $p(w_i|t_i)$ is the standard HMM lexical probability.

In order to get the indices to match in the formula above, three additional symbols are needed, i.e. t_{-1}, t_0 and t_{n+1} denote sentence boundary symbols, which are added during training and tagging for improved accuracy. Using sentence boundary symbols is adopted from Brants (2000).

In order to get some estimates for the probability of tag trigrams, which did not occur in the training data, we use tag bigram $p(t_i|t_{i-1})$ and tag unigram $p(t_i)$ models in parallel to the trigram model. Similarly we use models which assign probability $p(w_i|t_{i-1}, t_i)$ and $p(w_i|t_i, t_{i+1})$ in order to deal with previous unseen tag trigrams and wordforms. Of course the lexical model also weights analyses of words, serving as a backup model even in the case where the tag bigrams with the wordform were previously unseen.

[1] Although the probabilities $p(t|w)$ would seem like a more natural choice in the lexical model, the approximation for probabilities used in the HMM model of the tagger require the inverted probabilities $p(w|t)$. For a more thorough discussion of HMMs see Manning and Schütze (1999).

5.1 Lexical Models

For each tag t and word w, our lexical model estimates the probability $p(w|t)$. For unknown words, we construct similar guessers as Brants (2000) and Halácsy et al. (2007). The guessers estimate the probability $p(w|t)$ using the probabilities $p(s_i|t)$ for each of the suffixes of w. These can be computed from training material. The estimate $p(w|t)$ is a smoothed sum of the estimates for all of the suffixes, as explained by Brants (2000).

Like Brants (2000), we train separate guessers for upper and lower case words. For Finnish, we additionally train a guesser for sentence initial words, because preliminary tests revealed that there were a lot of unknown sentence initial words. Using a separate guesser for these words yielded better results than using the upper case or the lower case guesser. For English, a separate guesser for sentence initial words does not improve accuracy.

For Finnish another modification was needed in addition to the added guesser. For unknown words, it seemed beneficial to use only the 10 highest ranking guesses. For English, reducing the number of guesses also reduces accuracy. The maximum number of guesses is therefore a parameter which needs to be estimated experimentally and can vary between languages.

5.2 Tag Sequence Models

We construct a set of finite-state transducers whose effect is equivalent to an HMM. For the sake of space reduction, we do not compile a single transducer equivalent to an HMM. Instead we split the HMM into component models, each of which weights n-grams of wordforms and tags in the sentence. We give a short overview here and refer to Silfverberg and Lindén (2010) for a more thorough discussion on how this is done.

We simulate the tag n-gram models of a second order HMM using six models compiled into transducers. We use one transducer which assigns probabilitites for the tag unigrams in the sentence, two transducers which assign probabilitites for tag bigrams and three transducers assigning probabilities for tag trigrams.

As an example of how the transducers operate, we explain the structure of the three transducers which assign probabilities to tag trigrams. As explained above: After lexical probabilities have been assigned to the words in a sentence, the words and their analyses are compiled into a finite-state transducer, which assigns probabilities to the possible analyses of the entire sentence. Each of the three component models of the trigram model re-weight the paths of this transducer.

The first one of the models starts with the first three words (1st, 2nd and 3rd word) of the sentence and assigns a probability for each analysis trigram of the word triplet. It then moves on to the next three words (4th, 5th and 6th word) and their analyses, and so on. Hence the first model assigns a probability for each triplet of words and its analyses, which begins at indices $3k + 1$ in the sentence.

The second model skips the first word of the sentence, but after that it behaves as the first model re-scoring first the analyses of the triple (2nd, 3rd and 4th) and going on. As a result, it assigns probabilities to trigrams starting at indices $3k + 2$ in the sentence. By skipping the first two words, the third trigram model assigns weight to triplets beginning at indices $3k$. The net effect is that each trigram of wordforms and tags gets weighted once by the trigram model.

The models re-weighting tag bigrams and tag unigrams are constructed in an analogous way to the tag trigram models and the unigram bigram and trigram probabilities are smoothed using deleted interpolation, as suggested by Brants (2000).

Each of the models assigns a minimum penalty probability $1/(N + 1)$ to unknown tag n-grams. Here N is the size of the training corpus.

5.3 Context Dependent Lexical Models

In addition to the transducers making up the HMM model, we construct context dependent lexical models, which assign probabilities

$$p(w_i|t_{i-1}, t_i, t_{i+1}),\ p(w_i|t_{i-1}, t_i),\ p(w_i|t_i, t_{i+1})$$

to word and tag combinations in analyses. The models which assign probabilities to word and tag bigram combinations are included in order to estimate the probability $p(w_i|t_{i-1}, t_i, t_{i+1})$ when the combination of w_i with tags t_{i-1}, t_i and t_{i+1} has not been seen during training.

The context dependent lexical models are only applied to known words, but they do also provide additional improvement for tagging accuracy of unknown words by directly using neighboring words in estimating their tag profiles and probabilities. This is more reliable than using tag sequences.

The choice to only apply the models on known words is a convenient one. For known words context dependent lexical models were very easy for us to compile, since they are quite similar to ordinary tag n-gram models. Integrating them with the transducers making up the HMM model did not require any extra work besides estimating experimentally three coefficients which weight the models w.r.t. the HMM and each other. Weighted intersecting composition can be used to combine the sentence transducer and the re-scoring models regardless of how many models there are.

Similarly as in the HMM, unknown combinations of tags and words receive probability $1/(N+1)$, where N is the training corpus size.

6 Data

We trained taggers for English and Finnish using corpora compiled from newspaper text.

For English we used the Wall Street Journal Corpus in the Penn Treebank. We adopted the practice, introduced by Collins (2000), to use sections 0-18 for training lexical and tag models, sections 19-21 for fine tuning (like computing deleted interpolation coefficients) and sections 22-24 for testing.

For Finnish, we used a morphologically analyzed and disambiguated corpus of news from the 1995 volume of Helsingin Sanomat, the leading Finnish newspaper[2] (We used the news from the KA section of the corpus).

The morphological tagging in the Finnish corpus is machine-made and it has not been checked manually. This soon becomes evident when one examines the corpus, since there are a number of tagging errors. Thus our results for Finnish have to be considered tentative.

Table 1 shows the number of tokens in the training, fine-tuning and test materials used to construct and evaluate the taggers. The tokenization of the corpora is used as is and all token counts include words and punctuation. As the table shows, token counts for the Finnish and English corpora are comparable.

	English	Finnish
Training	969905	1027514
Tuning	148158	181437
Testing	171138 (2.43%)	156572 (10.41%)

Table 1: Summary of token counts for the data used for evaluation. The counts include words and punctuation. The amount of words, which were not seen during training, is indicated in parentheses.

	English	Finnish
POS Tags	81	776

Table 2: Number of POS tags in the Finnish and English corpora.

The amount of unknown words in the test corpus for Finnish is high. This is to be expected given the extensive morphology of the language. The extensive morphology is also reflected in the tag counts in table 2, which shows that the tag profile of the Finnish corpus is nearly ten times as large as the tag profile of the WSJ.[3]

Of the tags, in the Finnish corpus, 471 occur ten times or more, 243 occur one hundred times or more and 86 occur one thousand times or more. We conclude that there is a large number of tags which are fairly frequent. The corresponding figures for English are 58 tags occurring ten times or more, 44 tags occurring one hundred times or more and 38 tags occurring one thousand times or more.

The average number of possible analyses for words in the English corpus is 2.34. In the Finnish corpus, a word receives on average 1.45 analyses. The high number of analyses in the English corpus is partly explained by certain infrequent analyses of the frequent words ”a” and ”the”. When these words are excluded, the average number of analyses drops to 2.06.

When reporting accuracy, we divide the number of correctly tagged tokens with the total number of tokens in the test material, i.e. accuracy counts include punctuation. In this we follow the ACLWiki State of the Art page for POS tagging[4]. All re-

[2] Information about the corpus is available from `http://www.csc.fi/english/research/software/ftc`. It was compiled by The Research Institute for the Languages of Finland and CSC - IT Center for Science Ltd. The corpus can be obtained for academic use.

[3] There are 45 unique POS markers (such as `NN` and `JJ`) used in WSJ, but there are some unresolved ambiguities left in the corpus. That is why some words have POS tags consisting of more than one marker (eg. `VBG|NN|JJ`) making the total number of POS tags 81.

[4] `http://www.aclweb.org/aclwiki/`

sults on accuracy are reported for the test materials, which were not seen during training.

7 Evaluation

We trained four separate taggers both for English and Finnish. The accuracies for the different models are shown in table 3.

	1	2	3	4
Eng	96.42%	96.55%	96.70%	96.77%
Fin	95.56%	95.87%	95.98%	96.02%

1. Second order HMM.
2. Second order HMM augmented with lexical probabilities $p(w_i|t_{i-1}, t_i)$.
3. Second order HMM augmented with lexical probabilities $p(w_i|t_{i-1}, t_i)$ and $p(w_i|t_i, t_{i+1})$.
4. Second order HMM augmented with lexical probabilities $p(w_i|t_{i-1}, t_i)$, $p(w_i|t_i, t_{i+1})$ and $p(w_i|t_{i-1}, t_i, t_{i+1})$.

Table 3: Summary of tagging accuracies using different models. For Finnish, a separate guesser is used for sentence initial words.

The first tagger is a standard second order HMM. The only divergence from the HMM introduced by Brants (2000) is training a separate guesser for sentence initial words for Finnish and limiting the number of guesses to 10 for Finnish. This considerably improves the accuracy of the tagger from 94.91% to 95.56%.

The second tagger, we evaluate, is an HMM augmented by lexical probabilities conditioned on left tag context $p(w_i|t_{i-1}, t_i)$. This model roughly corresponds to the model Hunpos uses for POS tagging. As pointed out in section 5.3, the difference is that we do not estimate context dependent lexical probabilities for unknown words. This seems to lead to a slight reduction in accuracy.

In the third tagger, we add lexical probabilitites conditioned on right context $p(w_i|t_i, t_{i+1})$ and in the fourth tagger we add the final statistical model, which additionally uses lexical probabilities conditioned on both right and left tag context $p(w_i|t_{i-1}, t_i, t_{i+1})$.

The second tagger performs nearly as well as Hunpos and the third and fourth taggers perform better. This is to be expected, since the taggers incorporate right lexical context, which Hunpos cannot utilize.

	Seen	Unseen	Overall
TnT	96.77%	85.19%	96.46%
Hunpos	96.88%	86.13%	96.58%
Hfst	97.13%	83.72%	96.77%

Table 4: Summary of tagging accuracies for WSJ using TnT, Hunpos and Hfst. The accuracies are given for seen, unseen and all tokens. Hfst is our own tagger.

Table 4 shows accuracies for TnT, Hunpos and the best of our models, which we call **Hfst**, when tagging WSJ. It clearly performs the best out of all the taggers on all tokens and it has very high accuracy on known tokens. For unknown words, its accuracy is nevertheless somewhat lower than for TnT and Hunpos.

	Seen	Unseen	Overall
Hunpos	98.06%	76.83%	95.62%
Hfst	97.98%	81.04%	96.02%

Table 5: Summary of tagging accuracies for the Finnish test corpus using Hunpos and Hfst. The accuracies are given for seen, unseen and all tokens. Hfst is our own tagger.

Table 5 shows accuracies for Hunpos and Hfst on the Finnish test corpus. The accuracy on known words is markedly high for both taggers. This is probably partly due to the low average number of analyses per word, which makes analyzing known words easier than in English text. Conversely, the accuracy on unknown words is quite low and much lower for Hunpos than for Hfst.

By increasing the number of guesses from 10 to 40 for unknown words, we accomplish a similar reduction in accuracy on unknown words (from 81.04% to 79.29%) for Hfst as Hunpos exhibits. This points to the direction that the problems Hunpos encounters in tagging unknown Finnish words are in fact due to its unrestricted guesser.

In conclusion, the Hfst tagger has better overall performance than both Hunpos and TnT.

8 Discussion and Future Work

Because of extensive and fairly regular morphology, words in Finnish contain a lot of information about their part-of-speech and inflection. Hence

index.php?title=POS_Tagging_(State_of_the_art) words which share long suffixes are probably more likely to get the same correct POS analysis in Finnish than in English.

By considering more guesses for a Finnish unknown word, one at the same time considers more guesses which were suggested on basis of words with short suffixes in common with the unknown word. This problem is worsened by smoothing, which reduces the differences between the probabilities of suggestions. In fact we believe that this implies that the kind of guesser suggested by Brants (2000) and used in Hunpos is not the ideal choice for Finnish. And an architecture which makes it possible to try out different guesser designs, could make a tagger toolkit adaptable for a larger variety of languages than a traditional HMM.

The need for an added guesser for sentence initial words in Finnish can be understood rather easily by examining the test corpus. Sentences are fairly short, on average 10.3 words. The number of unknown words in the corpus is high and sentence initial words are not an exception. Both using the guesser for lower case words and upper case words produces poor results, because the first one underestimates the number of proper names among sentence initial words and the second one grossly overestimates it. Hence a guesser trained either on all words in the test material or only sentence initial words is needed.

The need for a sentence initial guesser in fact speaks in favor of Hunpos, since it incorporates a contextually dependent lexical model also for unknown words. Therefore it needs no special tweaks in order to perform well on sentence boundaries. Still, our baseline second order HMM achieves 95.56% which is extremely close to Hunpos 95.62%. The baseline model uses context dependent lexical probabilities only in the sense that it uses the separate guesser for sentence initial words. Perhaps this is indeed the only place where a context sensitive guesser has added effect in the Finnish corpus.

There is a lot of work left with the Hfst tagger. The accuracy of the guesser needs to be improved even when tagging English. There should not be any reason why it could not be made at least as accurate as the guesser in TnT.

Another improvement would be a rule compiler which compiles hand-written rules into sequential models, that are compatible with the statistical models which are used currently. Especially for Finnish, such a rule compiler would be a significant asset, because e.g. the disambiguation of analyses of verb forms often leads to long distance dependencies, which n-gram models capture poorly. It is not evident how TnT or Hupos could be adapted to using e.g. hand-written tagging rules in order to improve performance. But it would be an easy task for Hfst, if only there existed a suitable rule compiler.

A third direction of future work, would be to try the framework for other sequential labelling tasks such as tokenizing, chunking and named entity recognition.

9 Conclusion

We have demonstrated a framework for constructing POS taggers, which is capable of incorporating a variety of knowledge sources for POS tagging. We showed that it is possible, even straight forward, to combine different statistical models into one tagger. We hope that we have also demonstrated that it would be fairly straight forward to incorporate other kinds of models as well.

We constructed taggers for English and Finnish, which obtain superior accuracy compared to two widely known and used taggers TnT and Hunpos based on HMMs. For Finnish we modified the guessers used to tag unknown words in order to achieve added accuracy. Because of the modular design of our system, this did not require changes in any of the other models. We believe that the accuracy on tagging Finnish 96.02% shows that our taggers can be successfully adapted to languages which differ substantially from English.

Acknowledgments

We thank the HFST team for their support. We would also like to thank the anonymous reviewers of this paper. Their comments were appreciated. Miikka Silfverberg was financed by LangNet the Finnish doctoral programme in language studies.

References

Michelle Banko and Robert C. Moore. 2004. *Part of Speech Tagging in Context.* Proceedings of the 20th international conference on Computational Linguistics, COLING-2004, Stroudsburg, PA, USA.

Thorsten Brants. 2000. *A Statistical Part-of-Speech Tagger.* Proceedings of the sixth conference on

Applied natural language processing, ANLP-2000, Seattle, USA.

Kenneth Church. 1988. *A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text*. Proceedings of the second conference on Applied natural language processing, ANLP-1988, Austin, Texas, USA.

Michael Collins. 2002. *Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms*. Proceedings of the ACL-02 conference on Empirical methods in natural language processing, EMNLP-2002, Philadelphia, USA.

Dóra Csendes, János Csirik and Tibor Gyimóthy. 2004. *The Szeged Corpus: A POS tagged and Syntactically Annotated Hungarian Natural Language Corpus*. Proceedings of the 7th International Conference on Text Speech and Dialogue, TSD-2004, Brno, Czech Repulic.

Mark Dredze and Joel Wallenberg. 2008. *Icelandic data driven part of speech tagging*. Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies, HLT-2008, Stroudsburg, PA, USA.

Péter Halácsy, András Kornai and Csaba Oravecz. 2007. *HunPos – An Open Source Trigram Tagger*. Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, ACL-2007, Prague, Czech Republic.

Philipp Koehn. 2005. *Europarl: A Parallel Corpus for Statistical Machine Translation*. Machine Translation Summit, MTS-2005, Phuket, Thailand.

John Lafferty, Andrew MacCallum and Fernando Pereira. 2001. *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data*. Proceedings of the Eighteenth International Conference on Machine Learning, ICML-2001, Williamtown, MA, USA.

Krister Lindén, Miikka Silfverberg and Tommi Pirinen. 2009. *Hfst Tools for Morphology – an Efficient Open-Source Package for Construction of Morphological Analyzers*. Workshop on Systems and Frameworks for Computational Morphology, SFCM-2009, Zürich, Switzerland.

Christopher Manning and Hinrich Schütze. 1999. *Foundations of Natural Language Processing*. The MIT Press, Massachusets, USA.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz and Britta Schasberger. 1994. *The Penn Treebank: Annotating Predicate Argument Structure*. ARPA Human Language Technology Workshop, ARPA-1994, Plainsboro, New Jersey, USA.

Mehryar Mohri and Michael Riley. 2002. *An Efficient Algorithm for the n-Best-Strings Problem*. 7th International Conference on Spoken Language Processing, ICSLP-2002, Denver, USA.

Miikka Silfverberg and Krister Lindén. 2010. *Part-of-Speech Tagging Using Parallel Weighted Finite-State Transducers*. 7th International Conference on Natural Language Processing, ICETAL-2010, Reykjavik, Iceland.

Miikka Silfverberg and Krister Lindén. 2009. *Conflict Resolution Using Weighted Rules in HFST-TwolC*. The 17th Nordic Conference of Computational Linguistics, NODALIDA-2009, Odense, Denmark.

Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. 2003. *Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network*. Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology, HLT-NAACL-2003, Edmonton, Canada.

Yoshimasa Tsuruoka, Yuka Tateishi, Jin-Dong Kim, Tomoko Ohta, John McNaught, Sophia Ananiadou and Jun'ichi Tsuji. *Developing a Robust Part-of-Speech Tagger for Biomedical Text, Advances in Informatics*. 10th Panhellenic Conference on Informatics, PCI-2005, Volos, Greece.

Atro Voutilainen. *A Syntax-Based Part-of-Speech Analyser*. Proceedings of the seventh conference on European chapter of the Association for Computational Linguistics, EACL-1995, Dublin, Ireland.

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

Tommi A Pirinen and Miikka Silfverberg and Krister Lindén

University of Helsinki
Department of Modern Languages
University of Helsinki, 00014
tommi.pirinen@helsinki.fi, miikka.silfverberg@helsinki.fi, krister.linden@helsinki.fi

Abstract. In this paper we demonstrate a finite-state implementation of context-aware spell checking utilizing an N-gram based part of speech (POS) tagger to rerank the suggestions from a simple edit-distance based spell-checker. We demonstrate the benefits of context-aware spell-checking for English and Finnish and introduce modifications that are necessary to make traditional N-gram models work for morphologically more complex languages, such as Finnish.

1 Introduction

Spell-checking by computer is perhaps one of the oldest and most researched applications in the field of language technology starting from the mid 20th century [3]. One of the crucial parts of spell-checking—both from an interactive user-interface point of view and for unsupervised correction of errors—is the production of spelling suggestions. In this article we test various finite-state methods for using context and shallow morphological analysis to improve the suggestions generated by traditional edit distance measures or unigram frequencies such as [12].

The spell-checking task can be split into two parts, i.e. *detection* and actual *correction* of the spelling errors. The spelling errors can be detected in text as word forms that are unlikely to belong to the natural language in question, such as writing ‘cta’ instead of ‘cat’. This form of spelling errors is commonly called *non-word (spelling) errors*. Another form of spelling errors is word forms that do not belong to the given context under certain syntactic or semantic requirements, such as writing ‘their’ instead of ‘there’. This form is correspondingly called *real-word (spelling) errors*. The non-word type of spelling errors can easily be detected using a dictionary, whereas the detection of the latter type of errors typically requires syntactic analysis or probabilistic methods [8]. For the purpose of this article we do not distinguish between them, as the same correction methods can be applied to both.

The correction of spelling errors usually means generating a list of word forms belonging to the language for a user to chose from. The mechanism for generating correction suggestions for the erroneous word-forms is an *error-model*. The purpose of an error-model is to act as a filter to revert the mistakes the user typing the erroneous word-form has made. The simplest and most traditional model for making such corrections is the Levenshtein-Damerau edit distance algorithm, attributed initially to [6] and especially in the context of spell-checking to [3]. The Levenshtein-Damerau edit distance assumes that spelling errors are one of insertion, deletion or changing of a single character to another, or swapping two adjacent characters, which models well the spelling errors caused by an accidental slip of finger on a keyboard. It was originally discovered that for most languages and spelling errors, this simple method already covers 80 % of all spelling errors [3]. This model is also language-independent, ignoring the differences in character repertoires of a given language. Various other error models have also been developed, ranging from confusion sets to phonemic folding [5].

In this paper, we evaluate the use of context for further fine-tuning of the correction suggestions. The context is still not commonly used in spell-checkers. According to [5] it was lacking in the majority of spell-checkers and while the situation may have improved slightly for some commercial office suite products, the main spell-checkers for open source environments are still primarily context-ignorant, such as hunspell[1] which is widely used in the open source world. For English, the surface word-form trigrams model has been demonstrated to be reasonably efficient both for non-word cases [2] and for for real-word cases[7]. As an additional way to improve the set of suggestions, we propose to use morphosyntactically relevant analyses in context. In this article, we evaluate a model with a statistical morphological tagger [14].

The system described is fully built on freely available tools and data, available for download and use from `http://hfst.svn.sourceforge.net/viewvc/hfst/trunk/cicling-2011-contextspell/`. The only exception to this is the training data for Finnish, since there is no available morphological training data for Finnish as of yet, the download does not contain the source material for training but only the trigram models compiled into binary format automata.

Furthermore, we test the context-based spelling methods using both English and Finnish language materials to ensure the applicability of the method for morphologically different languages. The reason for doing this is two-fold; firstly the fact that English has rather low morphological productivity may make it behave statistically differently from other languages. On the other hand, English has the largest amount of freely available text corpora. For other languages, the availability of free corpora, especially annotated material, is often seen as a problem.

The article is laid out as follows: In Section 2, we outline the implementation of a finite-state context-aware spell-checker and describe the statistical methods used. In Section 3, we introduce the corpora and dictionaries used for spell-checking and training material as well as the corpora used for obtaining the spelling errors with context. In Section 4, we show how the created spelling correctors improve the results and explain the errors left. In Section 5, we com-

[1] `http://hunspell.sf.net`

pare our work to other current systems and enumerate possible improvements for both.

2 Methods

The spelling correction in this article is performed in several phases: assuming misspelled word *cta* for *cat*, we first apply the error model to the already known incorrect string *cta* to produce candidates for probable mistypings. For this purpose we use the Damerau-Levenshtein edit-distance algorithm in finite-state form. When applied to *cta* we get all strings with one or two typing mistakes, i.e. *ata*, *bta*, ..., *acta*, *bcta*, ..., *ta*, *ca*, ..., *tca*, and the correct *cat*. This set of strings is simultaneously matched against the language model, which will produce a set of corrections, such as *cat*, *act* or *car*. Since both the error-model and the language model contain information on likelihoods of errors and words respectively, the resulting list will be sorted according to a combination of the edit distance measure and the probability of the word in a reference corpus. The rankings based on edit distance alone and the edit distance combined with word probabilities form our two baseline models.

The context-based models we introduce here use the suggestion list gained from a contextless spelling-checker and the context of the words as input to rerank suggestions based on N-gram models. Each of the suggestions is tried against the N-gram models, and the ones with higher likelihoods will be lifted. For example when correcting the misspelling of ‘an’ as ‘anx’ in the sentence “this is anx example sentence”, as shown in the Table 1, we have the surface trigrams {this, is, _}, {is, _, example}, {_, example, sentence}, and corresponding analysis trigrams {DET, VVBZ, _}, {VVBZ, _, NN}, {_, NN, NN}. The suggestions for anx at edit distance one include ‘ax’, ‘an’ (one deletion), ‘ant’, ‘and’, ‘any’ (one change) and so on. To rank the possible suggestions, we substitute s_3 with the suggestions, and calculate the likelihood of their analyses.

Table 1. Example trigram combinations

this$_{s_1}$	is$_{s_2}$	_$_{s_3}$	example$_{s_4}$	sentence$_{s_5}$
DET$_{a_1}$	VVBZ$_{a_2}$	_ $_{a_3}$	NN$_{a_4}$	NN$_{a_5}$

2.1 Weighted Finite-State Interpretation of the Method

In this article we use a finite-state formulation of spell-checking. We assume the standard notation for finite-state algebra and define the language model as a weighted finite-state automaton assigning a weight to each correctly spelled word-form of a language, and an error model automaton mapping a misspelled string to a set of corrected strings and their weights. The probabilistic interpretation of the components is such that the weighted fsa as a language model assigns weight $w(s)$ to word s corresponding to the probability $P(s)$ for the word to be a correct word in the language. The error model assigns weight $w(s:r)$ to string pair s, r corresponding to the probability $P(s|r)$ of a user writing word r when intending to write the word s, and the context model assigns weight $w(s_3 a_3)$ to word s_3 with associated POS tagging a_3 corresponding to the standard HMM estimate $P(a_3 s_3)$ of the analysis being in a 3-gram context given by equation (1).

$$P(a_3 s_3) = \prod_{i=3}^{5} P(s_i|a_i) P(a_i|a_{i-2},\ a_{i-1}) \tag{1}$$

In a weighted finite-state system, the probabilistic data needs to be converted to the algebra supported by the finite-state weight structure. In this case we use the tropical semi-ring by transforming the frequencies into penalty weights with the formula $-\log \frac{f}{CS}$, where f is the frequency and CS the corpus size in number of tokens. If the language model allows for words that are not in the dictionary, a maximal weight is assigned to the unseen word forms that may be in the language model but not in the training corpus, i.e. any unseen word has a penalty weight of $-\log \frac{1}{CS}$.

The spelling corrections suggested by these unigram lexicon-based spell-checkers are initially generated by composing an edit-distance automaton [13] with an error weight corresponding to the probability of the error estimated in a corpus, i.e. $-\log \frac{f_F}{CS+1}$, where f_F is the frequency of the misspelling in a corpus. This weight is attached to the edit distance type error. In practice, this typically still means that the corrections are initially ordered primarily by the edit distance of the correction, and secondarily by the unigram frequency of the word-form in the reference corpus. This order is implicitly encoded in the weighted paths of the resulting automaton; to list the corrections we use the n-best paths algorithm [9]. This ordering is also used as our second baseline.

For a context-based reordering of the corrections, we use the POS tagging probabilities for the given suggestions. The implementation of the analysis N-gram probability estimation is similar to the one described in [14] with the following adaptations for the spelling correction. For the suggestion which gives the highest ranking, the most likely analysis is selected. The N-gram probability is estimated separately for each spelling suggestion and then combined with the baseline probability given by the unigram probability and the edit distance weight. The ideal scaling for the weights of unigram probabilities, i.e. edit distance probabilities with respect to N-gram probabilities, was acquired by performing tests on an automatically generated spelling error corpus.

The resulting finite-state system consists of three sets of automata, i.e. the dictionary for spell-checking, the error-model as described in [12], and the new N-gram model automata. The automata sizes are given in Table 2 for reference. The sizes also give an estimate of the memory usage of the spell-checking system, although the actual memory-usage during correction will rise depending on the actual extent of the search space during the correction phase.

Table 2. Automata sizes.

Automaton	States	Transitions	Bytes
	English		
Dictionary	25,330	42,448	1.2 MiB
Error model	1,303	492,232	5.9 MiB
N-gram lexicon	363,053	1,253,315	42 MiB
N-gram sequences	46,517	200,168	4.2 MiB
	Finnish		
Dictionary	179,035	395,032	16 MiB
Error model	1,863	983,227	12 MiB
N-gram lexicon	70,665	263,298	8.0 MiB
N-gram sequences	3,325	22,418	430 KiB

2.2 English-Specific Finite-State Weighting Methods

The language model for English was created as described in [10]. [2]. It consists of the word-forms and their probabilities in the corpora. The edit distance is composed of the standard English alphabet with an estimated error likelihood of 1 in 1000 words. Similarly for the English N-gram material, the initial analyses found in the WSJ corpus were used in the finite-state tagger as such. The scaling factor between the dictionary probability model and the edit distance model was acquired by estimating the optimal multipliers using the automatic misspellings and corrections of a Project Gutenberg Ebook[3] *Alice's Adventures in Wonderland.*

2.3 Finnish-Specific Finite-State Weighting Methods

The Finnish language model was based on a readily-available morphological weighted analyser of Finnish language [11]. We further modified the automaton to penalize suggestions with newly created compounds and derivations by adding a weight greater than the maximum to such suggestions, i.e. $-A \log \frac{1}{CS+1}$ in the training material. This has nearly the same effect as using a separate dictionary for suggestions that excludes the heavily weighted forms without requiring the extra space. Also for Finnish, a scaling factor was estimated by using automatic misspellings and corrections of a Project Gutenberg Ebook[4] *Juha.*

In the initial Finnish tagger, there was a relatively large tagset, all of which did not contain information necessary for the task of spell-checking, such as discourse particles, which are relatively context-agnostic [4], so we opted to simplify the tagging in these cases. Furthermore, the tagger used for training produced

[2] The finite-state formulation of this is informally described in `http://blogs.helsinki.fi/tapirine/2011/07/21/how-to-write-an-hfst-spelling-corrector/`

[3] `http://www.gutenberg.org/cache/epub/11/pg11.txt`

[4] `http://www.gutenberg.org/cache/epub/10863/pg10863.txt`

heuristic readings for unrecognized word-forms, which we also removed. Finally, we needed to add some extra penalties to the word forms unknown to the dictionary in the N-gram model, since this phenomenon was more frequent and diverse for Finnish than English.

3 Material

To train the spell-checker lexicons, word-form probabilities can be acquired from arbitrary running text. By using unigram frequencies, we can assign all word-forms some initial probabilities in isolation, i.e. with no spell-checking context. The unigram-trained models we used were acquired from existing spell-checker systems [10, 12], but we briefly describe the used corpora here as well.

To train the various N-gram models, corpora are required. For the surface-form training material, it is sufficient to capture running N-grams in the text. For training the statistical tagger with annotations, we also require disambiguated readings. Ideally of course this means hand-annotated tree banks or similar gold standard corpora.

The corpora used are summarized in Table 3. The sizes are provided to make it possible to reconstruct the systems. In practice, they are the newest available versions of the respective corpora at the time of testing. In the table, the first row is the training material used for the finite-state lexicon, i.e. the extracted surface word-forms without the analyses for unigram training. The second row is for the analyzed and disambiguated material for the N-gram based taggers for suggestion improvement. The third line is the corpora of spelling errors used only for the evaluation of the systems. As we can see from the figures of English compared with Finnish, there is a significant difference in freely available corpora such as Wikipedia. When going further to lesser resourced languages, the number will drop enough to make such statistical approaches less useful, e.g. Northern Sámi in [12].

Table 3. Sizes of training and evaluation corpora.

	Sentences	Tokens	Word-forms
	English		
Unigrams		2,110,728,338	128,457
N-grams	42,164	969,905	39,690
Errors	85	606	217
	Finnish		
Unigrams		17,479,297	968,996
N-grams	98,699	1,027,514	144,658
Errors	333	4,177	2,762

3.1 English corpora

The English dictionary is based on a frequency weighted word-form list of the English language as proposed in [10]. The word-forms were collected from the English Wiktionary[5], the English EBooks from the project Gutenberg[6] and the British National Corpus[7]. This frequency weighted word-list is in effect used as a unigram lexicon for spell-checking.

To train an English morphosyntactic tagger, we use the WSJ corpus. In this corpus each word is annotated by a single tag that encodes some morphosyntactic information, such as part-of-speech and inflectional form. The total number of tags in this corpus is 77 [8].

The spelling errors of English were acquired by extracting the ones with context from the Birkbeck error corpus[9]. In this corpus, the errors are from a variety of sources, including errors made by children and language-learners. For the purpose of this experiment we picked the subset of errors which had context and also removed the cases of word joining and splitting to simplify the implementation of parsing and suggestion.

3.2 Finnish Corpora

As the Finnish dictionary, we selected the freely available open source finite-state implementation of a Finnish morphological analyser[10]. The analyser had the frequency-weighted word-form list based on Finnish Wikipedia[11] making it in practice an extended unigram lexicon for Finnish. The Finnish morphological analyser, however, is capable of infinite compounding and derivation, which makes it a notably different approach to spell checking than the English finite word-form list.

The Finnish morphosyntactic N-gram model was trained using a morphosyntactically analyzed Finnish Newspaper[12]. In this format, the annotation is based on a sequence of tags, encoding part of speech and inflectional form. The total number of different tag sequences for this annotation is 747.

For Finnish spelling errors, we ran the Finnish unigram spell-checker through Wikipedia, europarl and a corpus of Finnish EBooks from the project Gutenberg[13] to acquire the non-word spelling errors, and picked at random the errors

[5] `http://en.wiktionary.org`

[6] `http://www.gutenberg.org/browse/languages/en`

[7] `http://www.kilgarriff.co.uk/bnc-readme.html`

[8] We used a deprecated version of the Penn Treebank where the tag set includes compound tags like `VB|NN` in addition to the usual Penn Treebank tags. For the final version of the article we will rerun the tests on the regular WSJ corpus.

[9] `http://ota.oucs.ox.ac.uk/headers/0643.xml`

[10] `http://home.gna.org/omorfi`

[11] `http://download.wikipedia.org/fiwiki/`

[12] http://www.csc.fi/english/research/software/ftc

[13] `http://www.gutenberg.org/browse/languages/fi`

having frequencies in range 1 to 8 instances; a majority of higher frequency non-words were actually proper nouns or neologisms missing from the dictionary. Using all of Wikipedia, europarl and Gutenberg provides a reasonable variety of both contemporary and old texts in a wide range of styles.

4 Tests and Evaluation

The evaluation of the correction suggestion quality is described in Table 4. The Table 4 contains the precision for the spelling errors. The precision is measured by ranked suggestions. In the tables, we give the results separately for ranks 1—5, and then for the accumulated ranks 1—10. The rows of the table represent different combinations of the N-gram models. The first row is a baseline score achieved by the weighted edit distance model alone, and the second is with unigram-weighted dictionary over edit-distance 2. The last two columns are the traditional word-form N-gram model and our POS tagger based extension to it.

Table 4. Precision of suggestion algorithms with real spelling errors.

Algorithm	1	2	3	4	5	1—10
English						
Edit distance 2 (baseline)	25.9 %	2.4 %	2.4 %	1.2 %	3.5 %	94.1 %
Edit distance 2 with Unigrams	28.2 %	5.9 %	29.4 %	3.5%	28.2 %	97.6 %
Edit distance 2 with Word N-grams	29.4 %	10.6 %	34.1 %	5.9 %	14.1 %	97.7 %
Edit distance 2 with POS N-grams	68.2 %	18.8 %	3.5 %	2.4 %	0.0 %	92.9 %
Finnish						
Edit distance 2 (baseline)	66.5 %	8.7 %	4.0 %	4.7 %	1.9 %	89.8 %
Edit distance 2 with Unigrams	61.2 %	13.4 %	1.6 %	3.1 %	3.4 %	88.2 %
Edit distance 2 with Word N-grams	65.0 %	14.4 %	3.8 %	3.1 %	2.2 %	90.6 %
Edit distance 2 with POS N-grams	71.4 %	9.3 %	1.2 %	3.4 %	0.3 %	85.7 %

It would appear that POS N-grams will in both cases give a significant boost to the results, whereas the word-form N-grams will merely give a slight increase to the results. In next subsections we further disect the specific changes to results the different approaches give.

4.1 English Error-Analysis

In [10], the authors identify errors that are not solved using simple unigram weights, such as correcting *rember* to *remember* instead of *member*. Here, our scaled POS N-gram context-model as well as the simpler word N-gram model, which can bypass the edit distance model weight will select the correct suggestion. However, when correcting e.g. *ment* to *meant* in stead of *went* or *met* the POS based context reranking gives no help as the POS stays the same.

4.2 Finnish Error-Analysis

In Finnish results we can easily notice that variation within the first position in the baseline results and reference system is more sporadic. This can be traced back to the fuzz factor caused by a majority of probabilities falling into the same category in our tests. The same edit-distance and unigram probability leaves the decision to random factors irrelevant to this experiment, such as alphabetical ordering that comes from data structures backing up the program code. The N-gram based reorderings are the only methods that can tiebreak the results here.

An obvious improvement for Finnish with POS N-grams comes from correcting agreeing NP's towards case agreement, such as *yhdistetstä* to *yhdisteistä* ('of compounds' Pl Ela) instead of the statistically more common *yhdisteestä* ('of compound' Sg Ela). However, as with English, the POS information does fail to rerank cases where two equally rare word-forms with the same POS occur at the same edit distance, which seems to be common with participles, such as correcting *varustunut* to *varautunut* in stead of *varastanut*.

Furthermore we note that the the discourse particles that were dropped from the POS tagger's analysis tag set in order to decrease the tag set size will cause certain word forms in the dictionary to be incorrectly reranked, such as when correcting the very common misspelling *muillekkin* into *muillekokin* ('for others as well?' Pl All Qst Kin) instead of the originally correct *muillekin* ('for others as well' Pl All Kin), since the analyses Qst (for question enclitic) and Kin (for additive enclitic) are both dropped from the POS analyses.

4.3 Performance Evaluation

We did not work on optimizing the N-gram analysis and selection, but we found that the speed of the system is reasonable—even in its current form. Table 5 summarizes the average speed of performing the experiments in Table 4.

Table 5. The speed of ranking the errors.

Material Algorithm	English	Finnish
Unigram (baseline)	10.0 s	51.8 s
	399.1 wps	6.2 wps
POS N-grams	377.4 s	1616.2 s
	10.6 wps	0.14 wps

The performance penalty that is incurred on Finnish spell-checking but not so much on English comes from the method of determining readings for words unknown to the language model, i.e. from the guessing algorithm. The amount of words unknown to the language model in Finnish was greater than for English due to the training data sparseness and the morphological complexity of the language.

5 Future Work and Discussion

We have shown that the POS based N-gram models are suitable for improving the spelling corrections for both morphologically more complex languages such as Finnish and for further improving languages with simpler morphologies like English. To further verify the claim, the method may still need to be tested on a typologically wider spectrum of languages.

In this article, we used readily available and hand-made error corpora to test our error correction method. A similar method as the one we use for error correction should be possible in error detection as well, especially when detecting real-word errors [7]. In future research, an obvious development is to integrate the N-gram system as a part of a real spell-checker system for both detection and correction of spelling errors, as is already done for the unigram based spell checker demonstrated in [12].

The article discussed only the reranking over basic edit distance error models, further research should include more careful statistical training for the error model as well, such as one demonstrated in [1].

6 Conclusion

In this paper we have demonstrated the use of finite-state methods for trigram based generation of spelling suggestions. We have shown that the basic wordform trigram methods suggested for languages like English do not seem to be as useful without modification for morphologically more complex languages like Finnish. Instead a more elaborate N-gram scheme using POS n-grams is successful for Finnish as well as English.

Acknowledgements

We are grateful to Sam Hardwick for making the spell checking software available and the HFST research group for fruitful discussions. We also thank the anonymous reviewers for useful suggestions and pointers.

References

1. Brill, E., Moore, R.C.: An improved error model for noisy channel spelling correction. In: ACL '00: Proceedings of the 38th Annual Meeting on Association for Computational Linguistics. pp. 286–293. Association for Computational Linguistics, Morristown, NJ, USA (2000)
2. Church, K., Gale, W.: Probability scoring for spelling correction. Statistics and Computing pp. 93–103 (1991)

3. Damerau, F.J.: A technique for computer detection and correction of spelling errors. Commun. ACM (7) (1964)
4. Hakulinen, A., Vilkuna, M., Korhonen, R., Koivisto, V., Heinonen, T.R., Alho, I.: Iso suomen kielioppi (2008), referred on 31.12.2008, available from `http://kaino.kotus.fi/visk`
5. Kukich, K.: Techniques for automatically correcting words in text. ACM Comput. Surv. 24(4), 377–439 (1992)
6. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics—Doklady 10, 707–710. Translated from Doklady Akademii Nauk SSSR pp. 845–848 (1966)
7. Mays, E., Damerau, F.J., Mercer, R.L.: Context based spelling correction. Inf. Process. Manage. 27(5), 517–522 (1991)
8. Mitton, R.: Ordering the suggestions of a spellchecker without using context*. Nat. Lang. Eng. 15(2), 173–192 (2009)
9. Mohri, M., Riley, M.: An efficient algorithm for the n-best-strings problem (2002)
10. Norvig, P.: How to write a spelling corrector. Web Page, Visited February 28th 2010, Available `http://norvig.com/spell-correct.html` (2010), `http://norvig.com/spell-correct.html`
11. Pirinen, T.A.: Modularisation of finnish finite-state language description—towards wide collaboration in open source development of a morphological analyser. In: Pedersen, B.S., Nešpore, G., Inguna Skadi n. (eds.) Nodalida 2011. NEALT Proceedings, vol. 11, pp. 299—302. NEALT (2011), `http://www.helsinki.fi/\%7Etapirine/publications/Pirinen-nodalida2011.pdf`
12. Pirinen, T.A., Lindén, K.: Finite-state spell-checking with weighted language and error models. In: Proceedings of the Seventh SaLTMiL workshop on creation and use of basic lexical resources for less-resourced languagages. pp. 13–18. Valletta, Malta (2010), `http://siuc01.si.ehu.es/\%7Ejipsagak/SALTMIL2010_Proceedings.pdf`
13. Savary, A.: Typographical nearest-neighbor search in a finite-state lexicon and its application to spelling correction. In: CIAA '01: Revised Papers from the 6th International Conference on Implementation and Application of Automata. pp. 251–260. Springer-Verlag, London, UK (2002)
14. Silfverberg, M., Lindén, K.: Combining statistical models for pos tagging using finite-state calculus. In: Proceedings of the 18th Conference on Computational Linguistics, NODALIDA 2011. pp. 183–190 (2011)

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

Teemu Ruokolainen[a] **Miikka Silfverberg**[b] **Mikko Kurimo**[a] **Krister Lindén**[b]
[a] Department of Signal Processing and Acoustics, Aalto University, firstname.lastname@aalto.fi
[b] Department of Modern Languages, University of Helsinki, firstname.lastname@helsinki.fi

Abstract

We discuss a simple estimation approach for conditional random fields (CRFs). The approach is derived heuristically by defining a variant of the classic perceptron algorithm in spirit of pseudo-likelihood for maximum likelihood estimation. The resulting approximative algorithm has a linear time complexity in the size of the label set and contains a minimal amount of tunable hyper-parameters. Consequently, the algorithm is suitable for learning CRF-based part-of-speech (POS) taggers in presence of large POS label sets. We present experiments on five languages. Despite its heuristic nature, the algorithm provides surprisingly competetive accuracies and running times against reference methods.

1 Introduction

The conditional random field (CRF) model (Lafferty et al., 2001) has been successfully applied to several sequence labeling tasks in natural language processing, including part-of-speech (POS) tagging. In this work, we discuss accelerating the CRF model estimation in presence of a large number of labels, say, hundreds or thousands. Large label sets occur in POS tagging of morphologically rich languages (Erjavec, 2010; Haverinen et al., 2013).

CRF training is most commonly associated with the (conditional) maximum likelihood (ML) criterion employed in the original work of Lafferty et al. (2001). In this work, we focus on an alternative training approach using the averaged perceptron algorithm of Collins (2002). While yielding competitive accuracy (Collins, 2002; Zhang and Clark, 2011), the perceptron algorithm avoids extensive tuning of hyper-parameters and regularization required by the stochastic gradient descent algorithm employed in ML estimation (Vishwanathan et al., 2006). Additionally, while ML and perceptron training share an identical time complexity, the perceptron is in practice faster due to sparser parameter updates.

Despite its simplicity, running the perceptron algorithm can be tedious in case the data contains a large number of labels. Previously, this problem has been addressed using, for example, k-best beam search (Collins and Roark, 2004; Zhang and Clark, 2011; Huang et al., 2012) and parallelization (McDonald et al., 2010). In this work, we explore an alternative strategy, in which we modify the perceptron algorithm in spirit of the classic *pseudo-likelihood* approximation for ML estimation (Besag, 1975). The resulting novel algorithm has linear complexity w.r.t. the label set size and contains only a single hyper-parameter, namely, the number of passes taken over the training data set.

We evaluate the algorithm, referred to as the *pseudo-perceptron*, empirically in POS tagging on five languages. The results suggest that the approach can yield competitive accuracy compared to perceptron training accelerated using a violation-fixed 1-best beam search (Collins and Roark, 2004; Huang et al., 2012) which also provides a linear time complexity in label set size.

The rest of the paper is as follows. In Section 2, we describe the pseudo-perceptron algorithm and discuss related work. In Sections 3 and 4, we describe our experiment setup and the results, respectively. Conclusions on the work are presented in Section 5.

2 Methods

2.1 Pseudo-Perceptron Algorithm

The (unnormalized) CRF model for input and output sequences $x = (x_1, x_2, \dots, x_{|x|})$ and

Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, pages 74–78,
Gothenburg, Sweden, April 26-30 2014. ©2014 Association for Computational Linguistics

$y = (y_1, y_2, \ldots, y_{|x|})$, respectively, is written as

$$p\,(y\,|\,x; \boldsymbol{w}) \propto \exp\Big(\boldsymbol{w} \cdot \boldsymbol{\Phi}(y, x)\Big)$$
$$= \prod_{i=n}^{|x|} \exp\Big(\boldsymbol{w} \cdot \boldsymbol{\phi}(y_{i-n}, \ldots, y_i, x, i)\Big)\,, \tag{1}$$

where $\boldsymbol{w}$ denotes the model parameter vector, $\boldsymbol{\Phi}$ the vector-valued global feature extracting function, $\boldsymbol{\phi}$ the vector-valued local feature extracting function, and n the model order. We denote the tag set as $\mathcal{Y}$. The model parameters $\boldsymbol{w}$ are estimated based on training data, and test instances are decoded using the Viterbi search (Lafferty et al., 2001).

Given the model definition (1), the parameters $\boldsymbol{w}$ can be estimated in a straightforward manner using the structured perceptron algorithm (Collins, 2002). The algorithm iterates over the training set a single instance (x, y) at a time and updates the parameters according to the rule $\boldsymbol{w}^{(i)} = \boldsymbol{w}^{(i-1)} + \Delta\boldsymbol{\Phi}(x, y, z)$, where $\Delta\boldsymbol{\Phi}(x, y, z)$ for the ith iteration is written as $\Delta\boldsymbol{\Phi}(x, y, z) = \boldsymbol{\Phi}(x, y) - \boldsymbol{\Phi}(x, z)$. The prediction z is obtained as

$$z = \underset{u \in \mathcal{Y}(x)}{\arg\max}\, \boldsymbol{w} \cdot \boldsymbol{\Phi}(x, u) \tag{2}$$

by performing the Viterbi search over $\mathcal{Y}(x) = \mathcal{Y} \times \cdots \times \mathcal{Y}$, a product of $|x|$ copies of $\mathcal{Y}$. In case the perceptron algorithm yields a small number of incorrect predictions on the training data set, the parameters generalize well to test instances with a high probability (Collins, 2002).

The time complexity of the Viterbi search is $O(|x| \times |\mathcal{Y}|^{n+1})$. Consequently, running the perceptron algorithm can become tedious if the label set cardinality $|\mathcal{Y}|$ and/or the model order n is large. In order to speed up learning, we define a variant of the algorithm in the spirit of pseudo-likelihood (PL) learning (Besag, 1975). In analogy to PL, the key idea of the pseudo-perceptron (PP) algorithm is to obtain the required predictions over single variables y_i while fixing the remaining variables to their true values. In other words, instead of using the Viterbi search to find the z as in (2), we find a z' for each position $i \in 1..|x|$ as

$$z' = \underset{u \in \mathcal{Y}'_i(x)}{\arg\max}\, \boldsymbol{w} \cdot \boldsymbol{\Phi}(x, u)\,, \tag{3}$$

with $\mathcal{Y}'_i(x) = \{y_1\} \times \cdots \times \{y_{i-1}\} \times \mathcal{Y} \times \{y_{i+1}\} \times \cdots \times \{y_{|x|}\}$. Subsequent to training, test instances are decoded in a standard manner using the Viterbi search.

The appeal of PP is that the time complexity of search is reduced to $O(|x| \times |\mathcal{Y}|)$, i.e., linear in the number of labels in the label set. On the other hand, we no longer expect the obtained parameters to necessarily generalize well to test instances.[1] Consequently, we consider PP a heuristic estimation approach motivated by the rather well-established success of PL (Korč and Förstner, 2008; Sutton and McCallum, 2009).[2]

Next, we study yet another heuristic pseudo-variant of the perceptron algorithm referred to as the *piecewise-pseudo-perceptron* (PW-PP). This algorithm is analogous to the piecewise-pseudo-likelihood (PW-PL) approximation presented by Sutton and McCallum (2009). In this variant, the original graph is first split into smaller, possibly overlapping subgraphs (pieces). Subsequently, we apply the PP approximation to the pieces. We employ the approach coined *factor-as-piece* by Sutton and McCallum (2009), in which each piece contains $n + 1$ consecutive variables, where n is the CRF model order.

The PW-PP approach is motivated by the results of Sutton and McCallum (2009) who found PW-PL to increase stability w.r.t. accuracy compared to plain PL across tasks. Note that the piecewise approximation in itself is not interesting in chain-structured CRFs, as it results in same time complexity as standard estimation. Meanwhile, the PW-PP algorithm has same time complexity as PP.

2.2 Related work

Previously, impractical running times of perceptron learning have been addressed most notably using the k-best beam search method (Collins and Roark, 2004; Zhang and Clark, 2011; Huang et al., 2012). Here, we consider the "greedy" 1-best beam search variant most relevant as it shares the time complexity of the pseudo search. Therefore, in the experimental section of this work, we compare the PP and 1-best beam search.

We are aware of at least two other learning approaches inspired by PL, namely, the pseudo-max and piecewise algorithms of Sontag et al. (2010) and Alahari et al. (2010), respectively. Compared to these approaches, the PP algorithm provides a simpler estimation tool as it avoids the

[1] We leave formal treatment to future work.

[2] Meanwhile, note that pseudo-likelihood is a consistent estimator (Gidas, 1988; Hyvärinen, 2006).

hyper-parameters involved in the stochastic gradient descent algorithms as well as the regularization and margin functions inherent to the approaches of Alahari et al. (2010) and Sontag et al. (2010). On the other hand, Sontag et al. (2010) show that the pseudo-max approach achieves consistency given certain assumptions on the data generating function. Meanwhile, as discussed in previous section, we consider PP a heuristic and do not provide any generalization guarantees. To our understanding, Alahari et al. (2010) do not provide generalization guarantees for their algorithm.

3 Experimental Setup

3.1 Data

For a quick overview of the data sets, see Table 1.

Penn Treebank. The first data set we consider is the classic Penn Treebank. The complete treebank is divided into 25 sections of newswire text extracted from the Wall Street Journal. We split the data into training, development, and test sets using the sections 0-18, 19-21, and 22-24, according to the standardly applied division introduced by Collins (2002).

Multext-East. The second data we consider is the multilingual Multext-East (Erjavec, 2010) corpus. The corpus contains the novel *1984* by George Orwell. From the available seven languages, we utilize the Czech, Estonian and Romanian sections. Since the data does not have a standard division to training and test sets, we assign the 9th and 10th from each 10 consecutive sentences to the development and test sets, respectively. The remaining sentences are assigned to the training sets.

Turku Dependency Treebank. The third data we consider is the Finnish Turku Dependency Treebank (Haverinen et al., 2013). The treebank contains text from 10 different domains. We use the same data split strategy as for Multext East.

3.2 Reference Methods

We compare the PP and PW-PP algorithms with perceptron learning accelerated using 1-best beam search modified using the early update rule (Huang et al., 2012). While Huang et al. (2012) experimented with several violation-fixing methods (early, latest, maximum, hybrid), they appeared to reach termination at the same rate in POS tagging. Our preliminary experiments using the latest violation updates supported this. Consequently, we employ the early updates.

lang.	train.	dev.	test	tags	train. tags
eng	38,219	5,527	5,462	45	45
rom	5,216	652	652	405	391
est	5,183	648	647	413	408
cze	5,402	675	675	955	908
fin	5,043	630	630	2,355	2,141

Table 1: Overview on data. The training (train.), development (dev.) and test set sizes are given in sentences. The columns titled *tags* and *train. tags* correspond to total number of tags in the data set and number of tags in the training set, respectively.

We also provide results using the CRFsuite toolkit (Okazaki, 2007), which implements a 1st-order CRF model. To best of our knowledge, CRFsuite is currently the fastest freely available CRF implementation.[3] In addition to the averaged perceptron algorithm (Collins, 2002), the toolkit implements several training procedures (Nocedal, 1980; Crammer et al., 2006; Andrew and Gao, 2007; Mejer and Crammer, 2010; Shalev-Shwartz et al., 2011). We run CRFsuite using these algorithms employing their default parameters and the feature extraction scheme and stopping criterion described in Section 3.3. We then report results provided by the most accurate algorithm on each language.

3.3 Details on CRF Training and Decoding

While the methods discussed in this work are applicable for nth-order CRFs, we employ 1st-order CRFs in order to avoid overfitting the relatively small training sets.

We employ a simple feature set including word forms at position $t-2,\ldots,t+2$, suffixes of word at position t up to four letters, and three orthographic features indicating if the word at position t contains a hyphen, capital letter, or a digit.

All the perceptron variants (PP, PW-PP, 1-best beam search) initialize the model parameters with zero vectors and process the training instances in the order they appear in the corpus. At the end of each pass, we apply the CRFs using the latest averaged parameters (Collins, 2002) to the development set. We assume the algorithms have converged when the model accuracy on development

[3] See benchmark results at `http://www.chokkan.org/software/crfsuite/benchmark.html`

has not increased during last three iterations. After termination, we apply the averaged parameters yielding highest performance on the development set to test instances.

Test and development instances are decoded using a combination of Viterbi search and the *tag dictionary* approach of Ratnaparkhi (1996). In this approach, candidate tags for known word forms are limited to those observed in the training data. Meanwhile, word forms that were unseen during training consider the full label set.

3.4 Software and Hardware

The experiments are run on a standard desktop computer. We use our own C++-based implementation of the methods discussed in Section 2.

4 Results

The obtained training times and test set accuracies (measured using accuracy and out-of-vocabulary (OOV) accuracy) are presented in Table 2. The training CPU times include the time (in minutes) consumed by running the perceptron algorithm variants as well as evaluation of the development set accuracy. The column labeled *it.* corresponds to the number of passes over training set made by the algorithms before termination.

We summarize the results as follows. First, PW-PP provided higher accuracies compared to PP on Romanian, Czech, and Finnish. The differences were statistically significant[4] on Czech. Second, while yielding similar running times compared to 1-best beam search, PW-PP provided higher accuracies on all languages apart from Finnish. The differences were significant on Estonian and Czech. Third, while fastest on the Penn Treebank, the CRFsuite toolkit became substantially slower compared to PW-PP when the number of labels were increased (see Czech and Finnish). The differences in accuracies between the best performing CRFsuite algorithm and PP and PW-PP were significant on Czech.

method	it.	time (min)	acc.	OOV
English				
PP	**9**	6	96.99	87.97
PW-PP	10	7	96.98	88.11
1-best beam	17	8	96.91	88.33
Pas.-Agg.	**9**	**1**	**97.01**	**88.68**
Romanian				
PP	9	8	96.81	83.66
PW-PP	**8**	**7**	96.91	84.38
1-best beam	17	10	96.88	**85.32**
Pas.-Agg.	13	9	**97.06**	84.69
Estonian				
PP	10	8	**93.39**	78.10
PW-PP	**8**	**6**	93.35	**78.66**
1-best beam	23	15	92.95	75.65
Pas.-Agg.	15	12	93.27	77.63
Czech				
PP	**11**	26	89.37	70.67
PW-PP	16	41	89.84	72.52
1-best beam	14	**19**	88.95	70.90
Pegasos	15	341	**90.42**	**72.59**
Finnish				
PP	**11**	58	87.09	58.58
PW-PP	**11**	**56**	87.16	58.50
1-best beam	21	94	**87.38**	**59.29**
Pas.-Agg.	16	693	87.17	57.58

Table 2: Results. We report CRFsuite results provided by most accurate algorithm on each language: the *Pas.-Agg.* and *Pegasos* refer to the algorithms of Crammer et al. (2006) and Shalev-Shwartz et al. (2011), respectively.

5 Conclusions

We presented a heuristic perceptron variant for estimation of CRFs in the spirit of the classic pseudo-likelihood estimator. The resulting approximative algorithm has a linear time complexity in the label set cardinality and contains only a single hyper-parameter, namely, the number of passes taken over the training data set. We evaluated the algorithm in POS tagging on five languages. Despite its heuristic nature, the algorithm provided competetive accuracies and running times against reference methods.

Acknowledgements

This work was financially supported by Langnet (Finnish doctoral programme in language studies) and the Academy of Finland under the grant no 251170 (Finnish Centre of Excellence Program (2012-2017)). We would like to thank Dr. Onur Dikmen for the helpful discussions during the work.

[4] We establish significance (with confidence level 0.95) using the standard 1-sided Wilcoxon signed-rank test performed on 10 randomly divided, non-overlapping subsets of the complete test sets.

References

Karteek Alahari, Chris Russell, and Philip H.S. Torr. 2010. Efficient piecewise learning for conditional random fields. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 895–901.

Galen Andrew and Jianfeng Gao. 2007. Scalable training of L_1-regularized log-linear models. In *Proceedings of the 24th international conference on Machine learning*, pages 33–40.

Julian Besag. 1975. Statistical analysis of non-lattice data. *The statistician*, pages 179–195.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111.

Michael Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, volume 10, pages 1–8.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585.

Tomaž Erjavec. 2010. Multext-east version 4: Multilingual morphosyntactic specifications, lexicons and corpora. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*.

Basilis Gidas. 1988. Consistency of maximum likelihood and pseudo-likelihood estimators for Gibbs distributions. In *Stochastic differential systems, stochastic control theory and applications*, pages 129–145.

Katri Haverinen, Jenna Nyblom, Timo Viljanen, Veronika Laippala, Samuel Kohonen, Anna Missilä, Stina Ojala, Tapio Salakoski, and Filip Ginter. 2013. Building the essential resources for Finnish: the Turku Dependency Treebank. *Language Resources and Evaluation*.

Liang Huang, Suphan Fayong, and Yang Guo. 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*, pages 142–151.

Aapo Hyvärinen. 2006. Consistency of pseudolikelihood estimation of fully visible Boltzmann machines. *Neural Computation*, 18(10):2283–2292.

Filip Korč and Wolfgang Förstner. 2008. Approximate parameter learning in conditional random fields: An empirical investigation. *Pattern Recognition*, pages 11–20.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289.

Ryan McDonald, Keith Hall, and Gideon Mann. 2010. Distributed training strategies for the structured perceptron. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 456–464.

Avihai Mejer and Koby Crammer. 2010. Confidence in structured-prediction using confidence-weighted models. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 971–981.

Jorge Nocedal. 1980. Updating quasi-Newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782.

Naoaki Okazaki. 2007. CRFsuite: a fast implementation of conditional random fields (CRFs). *URL http://www.chokkan.org/software/crfsuite*.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proceedings of the conference on empirical methods in natural language processing*, volume 1, pages 133–142.

Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. 2011. Pegasos: Primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127(1):3–30.

David Sontag, Ofer Meshi, Tommi Jaakkola, and Amir Globerson. 2010. More data means less inference: A pseudo-max approach to structured learning. In *Advances in Neural Information Processing Systems 23*, pages 2181–2189.

Charles Sutton and Andrew McCallum. 2009. Piecewise training for structured prediction. *Machine learning*, 77(2):165–194.

S.V.N. Vishwanathan, Nicol Schraudolph, Mark Schmidt, and Kevin Murphy. 2006. Accelerated training of conditional random fields with stochastic gradient methods. In *Proceedings of the 23rd international conference on Machine learning*, pages 969–976.

Yue Zhang and Stephen Clark. 2011. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151.

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

Miikka Silfverberg[a] Teemu Ruokolainen[b] Krister Lindén[a] Mikko Kurimo[b]

[a] Department of Modern Languages, University of Helsinki,
firstname.lastname@helsinki.fi

[b] Department of Signal Processing and Acoustics, Aalto University,
firstname.lastname@aalto.fi

Abstract

We discuss part-of-speech (POS) tagging in presence of large, fine-grained label sets using conditional random fields (CRFs). We propose improving tagging accuracy by utilizing dependencies within sub-components of the fine-grained labels. These sub-label dependencies are incorporated into the CRF model via a (relatively) straightforward feature extraction scheme. Experiments on five languages show that the approach can yield significant improvement in tagging accuracy in case the labels have sufficiently rich inner structure.

1 Introduction

We discuss part-of-speech (POS) tagging using the well-known conditional random field (CRF) model introduced originally by Lafferty et al. (2001). Our focus is on scenarios, in which the POS labels have a *rich inner structure*. For example, consider

PRON+1SG	V+NON3SG+PRES	N+SG	
I	like	ham	’

where the *compound* labels PRON+1SG, V+NON3SG+PRES, and N+SG stand for pronoun first person singular, verb non-third singular present tense, and noun singular, respectively. Fine-grained labels occur frequently in morphologically complex languages (Erjavec, 2010; Haverinen et al., 2013).

We propose improving tagging accuracy by utilizing dependencies within the *sub-labels* (PRON, 1SG, V, NON3SG, N, and SG in the above example) of the compound labels. From a technical perspective, we accomplish this by making use of the fundamental ability of the CRFs to incorporate arbitrarily defined feature functions. The newly-defined features are expected to alleviate data sparsity problems caused by the fine-grained labels. Despite the (relative) simplicity of the approach, we are unaware of previous work exploiting the sub-labels to the extent presented here.

We present experiments on five languages (English, Finnish, Czech, Estonian, and Romanian) with varying POS annotation granularity. By utilizing the sub-labels, we gain significant improvement in model accuracy given a sufficiently fine-grained label set. Moreover, our results indicate that exploiting the sub-labels can yield larger improvements in tagging compared to increasing model order.

The rest of the paper is organized as follows. Section 2 describes the methodology. Experimental setup and results are presented in Section 3. Section 4 discusses related work. Lastly, we provide conclusions on the work in Section 5.

2 Methods

2.1 Conditional Random Fields

The (unnormalized) CRF model (Lafferty et al., 2001) for a sentence $x = (x_1, \dots, x_{|x|})$ and a POS sequence $y = (y_1, \dots, y_{|x|})$ is defined as

$$p\,(y \,|\, x; \boldsymbol{w}) \propto \prod_{i=n}^{|x|} \exp\Big(\boldsymbol{w} \cdot \boldsymbol{\phi}(y_{i-n}, \dots, y_i, x, i)\Big)\,, \quad (1)$$

where n denotes the model order, $\boldsymbol{w}$ the model parameter vector, and $\boldsymbol{\phi}$ the feature extraction function. We denote the tag set as $\mathcal{Y}$, that is, $y_i \in \mathcal{Y}$ for $i \in 1 \dots |x|$.

2.2 Baseline Feature Set

We first describe our baseline feature set $\{\phi_j(y_{i-1}, y_i, x, i)\}_{j=1}^{|\boldsymbol{\phi}|}$ by defining *emission* and *transition* features. The emission feature set associates properties of the sentence position i with

Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 259–264,
Baltimore, Maryland, USA, June 23-25 2014. ©2014 Association for Computational Linguistics

the corresponding label as

$$\{\chi_j(x,i)\mathbb{1}(y_i = y'_i) \,|\, j \in 1 \ldots |\mathcal{X}| \,, \forall y'_i \in \mathcal{Y}\} \,, \tag{2}$$

where the function $\mathbb{1}(q)$ returns one if and only if the proposition q is true and zero otherwise, that is

$$\mathbb{1}(y_i = y'_i) = \begin{cases} 1 & \textbf{if} \quad y_i = y'_i \\ 0 & \textbf{otherwise} \end{cases} \,, \tag{3}$$

and $\mathcal{X} = \{\chi_j(x,i)\}_{j=1}^{|\mathcal{X}|}$ is the set of functions characterizing the word position i. Following the classic work of Ratnaparkhi (1996), our $\mathcal{X}$ comprises simple binary functions:

1. Bias (always active irrespective of input).

2. Word forms $x_{i-2}, \ldots, x_{i+2}$.

3. Prefixes and suffixes of the word form x_i up to length $\delta_{suf} = 4$.

4. If the word form x_i contains (one or more) capital letter, hyphen, dash, or digit.

Binary functions have a return value of either zero (inactive) or one (active). Meanwhile, the transition features

$$\begin{aligned} \{\mathbb{1}(y_{i-k} = y'_{i-k}) \ldots \mathbb{1}(y_i = y'_i) \,|\, \\ y'_{i-k}, \ldots, y'_i \in \mathcal{Y} \,, \forall k \in 1 \ldots n\} \end{aligned} \tag{4}$$

capture dependencies between adjacent labels irrespective of the input x.

2.2.1 Expanded Feature Set Leveraging Sub-Label Dependencies

The baseline feature set described above can yield a high tagging accuracy given a conveniently simple label set, exemplified by the tagging results of Collins (2002) on the Penn Treebank (Marcus et al., 1993). (Note that conditional random fields correspond to discriminatively trained hidden Markov models and Collins (2002) employs the latter terminology.) However, it does to some extent overlook some beneficial dependency information in case the labels have a rich sub-structure. In what follows, we describe expanded feature sets which explicitly model the sub-label dependencies.

We begin by defining a function $\mathcal{P}(y_i)$ which partitions any label y_i into its sub-label components and returns them in an unordered set. For example, we could define $\mathcal{P}(\text{PRON+1+SG}) = \{\text{PRON, 1, SG}\}$. (Label partitions employed in the experiments are described in Section 3.2.) We denote the set of all sub-label components as $\mathcal{S}$.

Subsequently, instead of defining only (2), we additionally associate the feature functions $\mathcal{X}$ with all sub-labels $s \in \mathcal{S}$ by defining

$$\{\chi_j(x,i)\mathbb{1}(s \in \mathcal{P}(y_i)) \,|\, \forall j \in 1 \ldots |\mathcal{X}| \,, \forall s \in \mathcal{S}\} \,, \tag{5}$$

where $\mathbb{1}(s \in \mathcal{P}(y_i))$ returns one in case s is in $\mathcal{P}(y_i)$ and zero otherwise. Second, we exploit *sub-label transitions* using features

$$\begin{aligned} \{\mathbb{1}(s_{i-k} \in \mathcal{P}(y_{i-k})) \ldots \mathbb{1}(s_i \in \mathcal{P}(y_i)) \,|\, \\ \forall s_{i-k}, \ldots, s_i \in \mathcal{S} \,, \forall k \in 1 \ldots m\} \,. \end{aligned} \tag{6}$$

Note that we define the sub-label transitions up to order m, $1 \leq m \leq n$, that is, an nth-order CRF model is not obliged to utilize sub-label transitions all the way up to order n. This is because employing high-order sub-label transitions may potentially cause overfitting to training data due to substantially increased number of features (equivalent to the number of model parameters, $|\boldsymbol{w}| = |\boldsymbol{\phi}|$). For example, in a second-order ($n = 2$) model, it might be beneficial to employ the sub-label emission feature set (5) and first-order sub-label transitions while discarding second-order sub-label transitions. (See the experimental results presented in Section 3.)

In the remainder of this paper, we use the following notations.

1. A standard CRF model incorporating (2) and (4) is denoted as CRF(n,-).

2. A CRF model incorporating (2), (4), and (5) is denoted as CRF(n,0).

3. A CRF model incorporating (2), (4), (5), and (6) is denoted as CRF(n,m).

2.3 On Linguistic Intuition

This section aims to provide some intuition on the types of linguistic phenomena that can be captured by the expanded feature set. To this end, we consider an example on the plural number in Finnish.

First, consider the plural nominative word form *kissat* (*cats*) where the plural number is denoted by the 1-suffix *-t*. Then, by employing the features (2), the suffix *-t* is associated solely with the compound label NOMINATIVE+PLURAL. However, by incorporating the expanded feature set (5), *-t* will also be associated to the sub-label PLURAL. This can be useful because, in Finnish, also adjectives and numerals are inflected according to number and denote the plural number with the suffix *-t* (Hakulinen et al., 2004, §79). Therefore, one can exploit *-t* to predict the plural number also in words such as *mustat* (*plural of black*) with a compound analysis ADJECTIVE+PLURAL.

Second, consider the number agreement (congruence). For example, in the sentence fragment *mustat kissat juoksevat (black cats are running)*, the words *mustat* and *kissat* share the plural number. In other words, the analyses of both *mustat* and *kissat* are required to contain the sub-label PLURAL. This short-span dependency between sub-labels will be captured by a first-order sub-label transition feature included in (6).

Lastly, we note that the feature expansion sets (5) and (6) will, naturally, capture any short-span dependencies within the sub-labels irrespective if the dependencies have a clear linguistic interpretation or not.

3 Experiments

3.1 Data

For a quick overview of the data sets, see Table 1.

Penn Treebank. The English Penn Treebank (Marcus et al., 1993) is divided into 25 sections of newswire text extracted from the Wall Street Journal. We split the data into training, development, and test sets using the sections 0-18, 19-21, and 22-24, according to the standardly applied division introduced by Collins (2002).

Turku Depedency Treebank. The Finnish Turku Depedendency Treebank (Haverinen et al., 2013) contains text from 10 different domains. The treebank does not have default partition to training and test sets. Therefore, from each 10 consecutive sentences, we assign the 9th and 10th to the development set and the test set, respectively. The remaining sentences are assigned to the training set.

Multext-East. The third data we consider is the multilingual Multext-East (Erjavec, 2010) corpus, from which we utilize the Czech, Estonian and Romanian sections. The corpus corresponds to translations of the novel *1984* by George Orwell. We apply the same data splits as for Turku Dependency Treebank.

lang.	train.	dev.	test	tags	train. tags
Eng	38,219	5,527	5,462	45	45
Rom	5,216	652	652	405	391
Est	5,183	648	647	413	408
Cze	5,402	675	675	955	908
Fin	5,043	630	630	2,355	2,141

Table 1: Overview on data. The training (train.), development (dev.) and test set sizes are given in sentences. The columns titled *tags* and *train. tags* correspond to total number of tags in the data set and number of tags in the training set, respectively.

3.2 Label Partitions

This section describes the employed compound label splits. The label splits for all data sets are submitted as data file attachments. All the splits are performed *a priori* to model learning, that is, we do not try to optimize them on the development sets.

The POS labels in the Penn Treebank are split in a way which captures relevant inflectional categories, such as tense and number. Consider, for example, the split for the present tense third singular verb label $\mathcal{P}(\text{VBZ}) = \{\text{VB}, \text{Z}\}$.

In the Turku Dependency Treebank, each morphological tag consists of sub-labels marking word-class, relevant inflectional categories, and their respective values. Each inflectional category, such as case or tense, combined with its value, such as nominative or present, constitutes one sub-label. Consider, for example, the split for the singular, adessive noun $\mathcal{P}(\text{N+CASE\_ADE+NUM\_SG}) = \{\text{POS\_N}, \text{CASE\_ADE}, \text{NUM\_SG}\}$.

The labeling scheme employed in the Multext-East data set represents a considerably different annotation approach compared to the Penn and Turku Treebanks. Each morphological analysis is a sequence of feature markers, for example Pw3–r. The first feature marker (P) denotes word class and the rest (w, 3, and r) encode values of inflectional categories relevant for that word class. A feature marker may correspond to several different values depending on word class and its position in the analysis. Therefore it becomes rather difficult to split the labels into similar pairs of inflectional category and value as we are able to do for the Turku Dependency Treebank. Since the interpretation of a feature marker depends on its position in the analysis and the word class, the markers have to be numbered and appended with the word class marker. For example, consider the split $\mathcal{P}(\text{Pw3–r}) = \{0 : \text{P}, 1 : \text{Pw}, 2 : \text{P3}, 5 : \text{Pr}\}$.

3.3 CRF Model Specification

We perform experiments using first-order and second-order CRFs with zeroth-order and first-order sub-label features. Using the notation introduced in Section 2, the employed models are CRF(1,-), CRF(1,1), CRF(2,-), CRF(2,0), and CRF(2,1). We do not report results using CRF(2,2) since, based on preliminary experiments, this model overfits on all languages.

The CRF model parameters are estimated using the averaged perceptron algorithm (Collins, 2002). The model parameters are initialized with a zero vector. We evaluate the latest averaged parameters on the held-out development set after each pass over the training data and terminate training if no improvement in accuracy is obtained during three last passes. The best-performing parameters are then applied on the test instances.

We accelerate the perceptron learning using beam search (Zhang and Clark, 2011). The beam width, b, is optimized separately for each language on the development sets by considering $b = 1, 2, 4, 8, 16, 32, 64, 128$ until the model accuracy does not improve by at least 0.01 (absolute).

Development and test instances are decoded using Viterbi search in combination with the tag dictionary approach of Ratnaparkhi (1996). In this approach, candidate tags for known word forms are limited to those observed in the training data. Meanwhile, word forms that were unseen during training consider the full label set.

3.4 Software and Hardware

The experiments are run on a standard desktop computer (Intel Xeon E5450 with 3.00 GHz and 64 GB of memory). The methods discussed in Section 2 are implemented in C++.

3.5 Results

The obtained tagging accuracies and training times are presented in Table 2. The times include running the averaged perceptron algorithm and evaluation of the development sets. The column labeled *it.* corresponds to the number of passes over the training data made by the perceptron algorithm before termination. We summarize the results as follows.

First, compared to standard feature extraction approach, employing the sub-label transition features resulted in improved accuracy on all languages apart from English. The differences were statistically significant on Czech, Estonian, and Finnish. (We establish statistical significance (with confidence level 0.95) using the standard 1-sided Wilcoxon signed-rank test performed on 10 randomly divided, non-overlapping subsets of the complete test sets.) This results supports the intuition that the sub-label features should be most useful in presence of large, fine-grained label sets, in which case the learning is most affected by data sparsity.

Second, on all languages apart from English, employing a first-order model with sub-label features yielded higher accuracy compared to a second-order model with standard features. The differences were again statistically significant on Czech, Estonian, and Finnish. This result suggests that, compared to increasing model order, exploiting the sub-label dependencies can be a preferable approach to improve the tagging accuracy.

Third, applying the expanded feature set inevitably causes some increase in the computational cost of model estimation. However, as shown by the running times, this increase is not prohibitive.

4 Related Work

In this section, we compare the approach presented in Section 2 to two prior systems which attempt to utilize sub-label dependencies in a similar manner.

Smith et al. (2005) use a CRF-based system for tagging Czech, in which they utilize expanded emission features similar to our (5). However, they do not utilize the full expanded transition features (6). More specifically, instead of utilizing a single chain as in our approach, Smith et al. employ five parallel structured chains. One of the chains models the sequence of word-class labels such as noun and adjective. The other four chains model gender, number, case, and lemma sequences, respectively. Therefore, in contrast to our approach, their system does not capture cross-dependencies between inflectional categories, such as the dependence between the word-class and case of adjacent words. Unsurprisingly, Smith et al. fail to achieve improvement over a generative HMM-based POS tagger of Hajič (2001). Meanwhile, our system outperforms the generative trigram tagger HunPos (Halácsy et al., 2007) which is an im-

model	it.	time (min)	acc.	OOV.
English				
CRF(1, -)	8	9	97.04	88.65
CRF(1, 0)	6	17	97.02	88.44
CRF(1, 1)	8	22	97.02	88.82
CRF(2, -)	9	15	97.18	88.82
CRF(2, 0)	11	36	97.17	89.23
CRF(2, 1)	8	27	97.15	89.04
Romanian				
CRF(1, -)	14	29	97.03	85.01
CRF(1, 0)	13	68	96.96	84.59
CRF(1, 1)	16	146	97.24	85.94
CRF(2, -)	7	19	97.08	85.21
CRF(2, 0)	18	99	97.02	85.42
CRF(2, 1)	12	118	97.29	86.25
Estonian				
CRF(1, -)	15	28	93.39	78.66
CRF(1, 0)	17	66	93.81	80.44
CRF(1, 1)	13	129	93.77	79.37
CRF(2, -)	15	30	93.48	77.13
CRF(2, 0)	13	53	93.78	79.60
CRF(2, 1)	16	105	94.01	79.53
Czech				
CRF(1, -)	6	28	89.28	70.90
CRF(1, 0)	10	112	89.94	74.44
CRF(1, 1)	10	365	90.78	76.83
CRF(2, -)	19	91	89.81	72.44
CRF(2, 0)	13	203	90.35	76.37
CRF(2, 1)	24	936	91.00	77.75
Finnish				
CRF(1, -)	10	80	87.37	59.29
CRF(1, 0)	13	249	88.58	63.46
CRF(1, 1)	12	474	88.41	62.63
CRF(2, -)	11	106	86.74	56.96
CRF(2, 0)	13	272	88.52	63.46
CRF(2, 1)	12	331	88.68	63.62

Table 2: Results.

proved open-source implementation of the well-known TnT tagger of Brants (2000). The obtained HunPos results are presented in Table 3.

	Eng	Rom	Est	Cze	Fin
HunPos	96.58	96.96	92.76	89.57	85.77

Table 3: Results using a generative HMM-based HunPos tagger of Halacsy et al. (2007).

Ceauşu (2006) uses a maximum entropy Markov model (MEMM) based system for tagging Romanian which utilizes transitional behavior between sub-labels similarly to our feature set (6). However, in addition to ignoring the most informative emission-type features (5), Ceauşu embeds the MEMMs into the tiered tagging framework of Tufis (1999). In tiered tagging, the full morphological analyses are mapped into a coarser tag set and a tagger is trained for this reduced tag set. Subsequent to decoding, the coarser tags are mapped into the original fine-grained morphological analyses. There are several problems associated with this tiered tagging approach. First, the success of the approach is highly dependent on a well designed coarse label set. Consequently, it requires intimate knowledge of the tag set and language. Meanwhile, our model can be set up with relatively little prior knowledge of the language or the tagging scheme (see Section 3.2). Moreover, a conversion to a coarser label set is necessarily lossy (at least for OOV words) and potentially results in reduced accuracy since recovering the original fine-grained tags from the coarse tags may induce errors. Indeed, the accuracy 96.56, reported by Ceauşu on the Romanian section of the Multext-East data set, is substantially lower than the accuracy 97.29 we obtain. These accuracies were obtained using identical sized training and test sets (although direct comparison is impossible because Ceauşu uses a non-documented random split).

5 Conclusions

We studied improving the accuracy of CRF-based POS tagging by exploiting sub-label dependency structure. The dependencies were included in the CRF model using a relatively straightforward feature expansion scheme. Experiments on five languages showed that the approach can yield significant improvement in tagging accuracy given sufficiently fine-grained label sets.

In future work, we aim to perform a more fine-grained error analysis to gain a better understanding where the improvement in accuracy takes place. One could also attempt to optimize the compound label splits to maximize prediction accuracy instead of applying a priori partitions.

Acknowledgements

This work was financially supported by Langnet (Finnish doctoral programme in language studies) and the Academy of Finland under the grant no 251170 (Finnish Centre of Excellence Program (2012-2017)). We would like to thank the anonymous reviewers for their useful comments.

References

Thorsten Brants. 2000. Tnt: A statistical part-of-speech tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, pages 224–231.

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

Michael Collins. 2002. Discriminative training methods for Hidden Markov Models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, volume 10, pages 1–8.

Tomaž Erjavec. 2010. Multext-east version 4: Multilingual morphosyntactic specifications, lexicons and corpora. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*.

Jan Hajič, Pavel Krbec, Pavel Květoň, Karel Oliva, and Vladimír Petkevič. 2001. Serial combination of rules and statistics: A case study in czech tagging. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 268–275.

Auli Hakulinen, Maria Vilkuna, Riitta Korhonen, Vesa Koivisto, Tarja Riitta Heinonen, and Irja Alho. 2004. *Iso suomen kielioppi*. Suomalaisen Kirjallisuuden Seura, Helsinki, Finland.

Péter Halácsy, András Kornai, and Csaba Oravecz. 2007. Hunpos: An open source trigram tagger. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 209–212.

Katri Haverinen, Jenna Nyblom, Timo Viljanen, Veronika Laippala, Samuel Kohonen, Anna Missilä, Stina Ojala, Tapio Salakoski, and Filip Ginter. 2013. Building the essential resources for Finnish: the Turku Dependency Treebank. *Language Resources and Evaluation*.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289.

M.P. Marcus, M.A. Marcinkiewicz, and B. Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proceedings of the conference on empirical methods in natural language processing*, volume 1, pages 133–142. Philadelphia, PA.

Noah A. Smith, David A. Smith, and Roy W. Tromble. 2005. Context-based morphological disambiguation with random fields. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 475–482.

Dan Tufis. 1999. Tiered tagging and combined language models classifiers. In *Proceedings of the Second International Workshop on Text, Speech and Dialogue*, pages 28–33.

Yue Zhang and Stephen Clark. 2011. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151.

Noname manuscript No.
(will be inserted by the editor)

FinnPos: An Open-Source Morphological Tagging and Lemmatization Toolkit for Finnish

Miikka Silfverberg · Teemu Ruokolainen · Krister Lindén · Mikko Kurimo

the date of receipt and acceptance should be inserted later

Abstract 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 evaluation 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.

Keywords Morphological tagging · Data-driven lemmatization · Averaged perceptron · Finnish · Open-source

M. Silfverberg (✉) · K. Lindén
University of Helsinki, Helsinki, Finland

M. Silfverberg
E-mail: mpsilfve@iki.fi

T. Ruokolainen · M. Kurimo
Aalto University, Helsinki, Finland

1 Introduction

This paper presents FinnPos, an open-source morphological tagging and lemmatization toolkit for Finnish. Our work stems from the recently published Turku Dependency Treebank (Haverinen et al, 2009, 2013) and FinnTreebank (Voutilainen, 2011). The Turku Dependency Treebank has been prepared by manually correcting the output of an automatic annotation process, whereas the FinnTreeBank has been prepared completely manually. These data sets are the first treebanks published for Finnish and are freely available for research purposes.

The morphological tagging component of the FinnPos system is based on the well-known averaged structured perceptron classifier (Collins, 2002). We accelerate perceptron learning using a combination of two approximations, namely, beam search and a model cascade inspired by the work of Müller et al (2013) on the conditional random field (CRF) model. Approximative learning is necessary since exact model estimation is infeasible given the vast number of morphological labels present in the data. Meanwhile, the lemmatization is performed using OMorFi, an open-source morphological analyzer for Finnish (Pirinen, 2008). In order to lemmatize word forms unknown to OMorFi, FinnPos implements a statistical lemmatization model which learns to perform lemmatization for any given word form in a data-driven manner following Chrupala et al (2008).

Earlier work on morphological tagging of Finnish has utilized rule-based methodology, exemplified by the Constraint Grammar approach of Karlsson (1990). Due to the recentness of the corpora applicable for learning, there exists relatively little work on statistical morphological tagging of Finnish. An exception is the work of Silfverberg et al (2011) who investigated a finite state machine implementation of the classic hidden Markov model tagging approach of Brants (2000). More recently, Bohnet et al (2013) investigated joint morphological tagging and dependency parsing of Finnish on the Turku Dependency Treebank. In their work, the morphological tagging was performed using the MarMot system (Müller et al, 2013).

The main contributions of this work are as follows. First, we present the first morphological tagging and lemmatization toolkit designed specifically for Finnish. The presented toolkit, FinnPos, is provided as an open-source implementation. Second, we compare the FinnPos system to three established morphological analysis toolkits, namely, HunPos (Halácsy et al, 2007), Morfette (Chrupala et al, 2008), and MarMot (Müller et al, 2013). The performed empirical evaluation shows that FinnPos performs favorably to the reference systems in terms of tagging and lemmatization accuracy, as well as computational efficiency of learning new models and assigning analyses to novel sentences.

The rest of the paper is organized as follows. In Section 2, we describe the treebanks employed for training and evaluation of the FinnPos system. The methods implemented by the FinnPos system are discussed in Section 3. In Section 4, we present the empirical evaluation. Finally, conclusions on the work are presented in Section 5.

2 Treebanks

This section describes the treebanks employed for training and evaluation of the FinnPos system.

Turku Dependency Treebank. The Turku Dependency Treebank (TDT) (Haverinen et al, 2009, 2013) contains text from ten varying domains, such as Wikipedia articles, blog entries, and financial news. The annotation has been prepared by manually correcting the output of an automatic annotation process. The morphological analyses of word tokens are post-processed outputs of OMorFi, an open-source morphological analyzer for Finnish (Pirinen, 2008). The resulting 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. Table 1 shows the analysis of an exemplar sentence *Hän ei asu pienessä kylässä* (*(S)he doesn't live in a small village*). We refer to the combination of the POS and the more specific tags as the *morphological label*.

word form	lemma	POS	other tags
Hän	hän	Pronoun	pers sg nom up
ei	ei	Verb	neg sg3 act
asu	asua	Verb	prs ind conneg
pienessä	pieni	Adjective	sg ine pos
kylässä	kylä	Adverb	sg ine

Table 1: A disambiguated analysis for an exemplar sentence *Hän ei asu pienessä kylässä* using Turku Treebank annotation adapted from Haverinen et al (2013).

FinnTreeBank. The FinnTreebank (FTB) (Voutilainen, 2011) 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). Similarly to TDT, FTB contains text from various domains including newspapers and fiction. The major difference between TDT and FTB, therefore, is that FTB contains a variety of grammatical examples, whereas TDT contains more real-life language use. Both the morphological tagging and dependency structures have been manually prepared. Similarly to TDT, the morphological analyses of word tokens in FTB are post-processed outputs of OMorFi (Pirinen, 2008). 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. Finally, for a summarizing presentation of TDT and FTB, see Table 2.

3 Methods

In the FinnPos system, we regard the morphological tagging and lemmatization tasks as two separate sub-problems. Given a sentence, each word form is assigned a mor-

	TDT	FTB
size	13,572 sent. (183,118 tok.)	19,121 sent. (162,028 tok.)
# labels	2,014	1,399
OMorFi coverage	94.2%	99.0%

Table 2: Summary of Turku Dependency Treebank (TDT) (Haverinen et al, 2009, 2013) and FinnTreeBank (FTB) (Voutilainen, 2011). The OMorFi coverage refers to coverage per token.

phological label by the morphological tagger based on the averaged structured perceptron (Collins, 2002). Subsequent to assigning the morphological label, selecting the appropriate word lemma is, in principle, straightforward given the set of full analyses (morphological labels and lemmas) provided by the OMorFi analyzer. However, OMorFi does not have full vocabulary coverage, that is, for some word forms no analyses are returned. In these cases, a simple baseline solution would be to simply return the original word form as the lemma. However, a more appealing approach is to learn a lemmatization model in a data-driven manner and apply it to lemmatize the unknown word forms (Chrupala et al, 2008). In what follows, we describe the applied structured perceptron tagger and data-driven lemmatizer in Sections 3.1 and 3.2, respectively.

3.1 Averaged Structured Perceptron

In this section, we describe the morphological tagging component based on the averaged structured perceptron classifier presented originally for sequence labeling by Collins (2002). The issues covered include the model definition, feature extraction, model estimation from training data, and decoding.

3.1.1 Scoring Function

Given a sentence $x = (x_1, \ldots, x_{|x|})$ and a label sequence $y = (y_1, \ldots, y_{|x|})$, the structured perceptron classifier assigns the pair (x, y) a score

$$\text{score}(x, y; \boldsymbol{w}) = \sum_{i=n}^{|x|} \boldsymbol{w} \cdot \boldsymbol{\phi}(y_{i-n}, \ldots, y_i, x, i)\,, \tag{1}$$

where n denotes the model order, $\boldsymbol{w}$ the model parameter vector, and $\boldsymbol{\phi}$ the feature extraction function. The word forms x_i are assigned labels from a potentially large label set $\mathcal{Y}$, that is, $y_i \in \mathcal{Y}$ for all $i = 1, \ldots, |x|$. As shown in Table 2, for TDT and FTB, the label sets $\mathcal{Y}$ contain roughly 2,000 and 1,400 morphological tags, respectively.

3.1.2 Feature Extraction

The appeal of the perceptron classifier (1) lies in its capability of utilizing rich, overlapping feature sets. The individual features correspond to the elements of feature vector $\phi(y_{i-n}, \dots, y_i, x, i)$. In the FinnPos system, the feature extraction scheme follows the *node-observation* presentation of Sutton and McCallum (2011), in which each label position is associated with a set of features describing the input. Specifically, we follow the classic work of Ratnaparkhi (1996) on morphological tagging and include the following input feature set:

1. Bias (always active irrespective of input).
2. Word forms $x_{i-2}, \dots, x_{i+2}$.
3. Prefixes and suffixes of the word form x_i up to length δ_{affix}.
4. If the word form x_i contains (one or more) capital letter, hyphen, dash, or digit.

In addition, we use the following binary functions:

5. The lower-cased word form x_i.
6. The word pairs (x_{i-1}, x_i) and (x_i, x_{i+1}).

When using a morphological analyzer, we also include:

7. Each morphological label of word x_i.

In addition, the node-observation scheme utilizes label transition features to capture the fact that some label transitions occur more often than others. For example, the Finnish word *asu* could occur with a noun (*clothing*) or verb (a negative or imperative form of *to live*) label. In the example in Table 1, it is, however, preceded by a negator (*ei*). Therefore, in this context, *asu* is more likely to be a verb rather than a noun since in Finnish negators seldomly precede nouns.

The above features treat the morphological labels as single entities. However, they overlook some beneficial dependency information given the rich inner structure of the TDT and FTB labels discussed in Section 2. Therefore, we follow Silfverberg et al (2014) and utilize an expanded feature set which aims to capture these dependencies. For example, consider the word form *kissat* (*cats*) where the suffix *-t* denotes plural number. Then, given the feature extraction scheme of Silfverberg et al (2014), instead of associating the suffix *-t* solely with a compound label (Noun, nominative, plural), we also relate it with the sub-label Plural. This is because one can exploit the suffix *-t* to predict the plural number also in words such as *vihreät* (*plural of green*) with an analysis (Adjective, nominative, plural).

In addition to associating the input to sub-labels as described above, the expanded feature set exploits transitional behavior of the sub-labels. For example, consider the sentence fragment *kissat juovat* (*cats drink*) where the words *kissat* and *juovat* have compound analyses (Noun, nominative, plural) and (Verb, 3rd person, plural, present tense, active), respectively. Then, instead of merely modeling the transitional dependency between the compound labels, we also model the congruence, that is, both analyses need to contain the sub-label denoting plural number.

3.1.3 Estimation from Data

In this section, we describe the averaged perceptron parameter estimation procedure implemented in the FinnPos system.

Averaged Perceptron Learning. The perceptron learning algorithm operates by iteratively searching for the highest scoring label sequence for a training instance x and updating the model parameters in case of an incorrect search result. From implementation perspective, the exact search is performed using the standard Viterbi algorithm. Inconveniently, however, perceptron learning utilizing exact Viterbi search is impractically slow in presence of large label sets. Therefore, in order to speed up the estimation, we implement the perceptron algorithm utilizing *beam search* using *minimum divergence beams* following Pal et al (2006).[1] Finally, the parameter averaging technique (Freund and Schapire, 1999; Collins, 2002) provides a simple, hyper-parameterless means of model regularization.

Model Cascade. In a recent work, Müller et al (2013) presented an approximative high-order CRF estimation technique utilizing a cascade of CRF models of increasing orders. In a general cascade system for structured prediction, one learns a series of increasingly complex models by restricting the search space of each model using the predictions of the less complex models (Weiss and Taskar, 2010). Müller et al (2013) implement this approach for CRFs using a coarse-to-fine decoding technique (Charniak and Johnson, 2005; Rush and Petrov, 2012) and show large savings in the computational cost of maximum likelihood training.

In the FinnPos system, we implement another cascading variant by applying a series of two models, an orthography-based label guesser and a conventional nth-order perceptron classifier. In this approach, the idea is simply to utilize the minimalistic, orthography-based label guesser to narrow down the label search space. This roughly corresponds to the zero-order pruning performed by Müller et al (2013). In order to apply the cascade, we first learn the label guesser from the training data. The guesser ranks morphological tags according to their probability for any given word form. We then use the guesser to limit the candidate label set for each word x_i in sentence x and, subsequently, use beam search to find the highest scoring label sequence among the limited candidate sequences. Parameter updates are performed in a standard manner.

The implemented label guesser is based on the lexical model for OOV words used by Brants (2000). It assigns a probability $p(y|x)$ for any label $y \in \mathcal{Y}$ and an arbitrary word x based on the suffixes of x. Appealingly, the guesser can be trained and applied in mere seconds even when using large data sets. We use the label guesser to extract the minimal set of highest ranking label guesses y_i whose combined probability mass $\sum_{i=0}^{n} p(y_i|x)$ exceeds a threshold $\kappa \in [0,\ 1]$. The threshold is considered a hyper-parameter of the learning procedure, which is tuned on a held-out development set. Essentially, if one employs too small a κ, the model will underfit the training data, while increasing the threshold results in increasingly accurate approximations of the original perceptron learning problem.

[1] In addition to applying standard beam search and parameter updates, we experimented with the maximum violation and early updates of Huang et al (2012) but obtained no improvements in model accuracy.

3.1.4 Decoding

Subsequent to parameter estimation using the learning algorithms discussed in Section 3.1.3, the resulting perceptron classifier can be applied to any given word sequence. In this decoding stage, the model assigns the highest scoring label sequence to a given word sequence. The search is performed using beam search with the minimum divergence beams following Pal et al (2006). In addition, since model estimation is performed utilizing a label guesser, the label guesser is also applied during decoding with the same threshold value κ. Lastly, if a morphological analyzer is available during decoding, the labels of test instances are restricted according to the output of the analyzer.

3.2 Lemmatizer

In order to lemmatize words unknown to the OMorFi analyzer, we follow Chrupala et al (2008) and treat the lemmatization problem as a classification task, in which each class corresponds to a *suffix edit script*. For example, consider $[ies \to y]$, which removes a suffix "-ies" from the end of a word form, such as the English word form "beauties", and replaces it with another suffix "-y", thus producing the lemma "beauty". While Chrupala et al (2008) use rather general edit scripts which can additionally modify prefixes and infixes of the word, we rely on the suffix-based approach because Finnish words mostly inflect at the end. The task of the lemmatizer is then to find the most appropriate edit script based on features extracted from the word form, its morphological label and its context. The script is chosen among *minimal edit scripts*, where the removed suffix is as short as possible (Chrupala et al, 2008).

3.3 Implementation Details

This section describes low-level details involved in the implementation of the tagging and lemmatization methods discussed above in Sections 3.1 and 3.2, respectively. The covered issues include hyper-parameter tuning, initialization procedures, and software.

Feature Extraction. The default feature extraction follows the presentation in Section 3. However, users can freely define their own feature sets.[2] The toolkit implements label and sub-label transitions discussed in Section 3.1.2 up to second order. The transition orders are optimized based on the development set. We use prefixes and suffixes of words up to length 10 ($\delta_{affix} = 10$).

[2] See documentation at `github.com/mpsilfve/FinnPos/wiki`

Averaged Perceptron and Label Guesser. The perceptron algorithm initializes model parameters with a zero vector. In order to reduce overfitting, we apply the parameter averaging approach following Collins (2002) and an early stopping criterion based on the held-out development set. In early stopping, we apply the averaged parameters to the development set after each pass over the training data and terminate training in case the accuracy has not improved during the previous 3 passes. Subsequently, we apply the best performing parameter setting to the test instances. The label guesser is trained using all words in the training data utilizing all word suffixes up to length 10.

Lemmatizer. Given lemmatized training data, we first extract all minimal suffix edit scripts. We then train an averaged perceptron classifier (Freund and Schapire, 1999) to disambiguate between all applicable suffix edit scripts for each word form in the training data. The classifier uses the following feature set:

1. The word form w.
2. Suffixes of w up to length 10.
3. Infixes (w_{n-4}, w_{n-3}), (w_{n-3}, w_{n-2}) and (w_{n-2}, w_{n-1}) of the word form $w = (w_1, \ldots, w_n)$.
4. The morphological label of w as well as its sub-labels.
5. If the word form w contains (one or more) capital letters, hyphens, dashes, or digits.

Additionally, we use combination features where each feature is combined with the morphological label of w and its sub-labels. Overfitting to training data is controlled using parameter averaging and early stopping based on the development data.

Implementation. To guarantee efficient estimation and inference, FinnPos is implemented in C++. In order to facilitate compilation and avoid clashes between library versions, we eliminated most dependencies on external software and libraries. Currently, the only required external utility is the lookup program for the OMorFi morphological analyzer distributed with the HFST library (Lindén et al, 2011).

4 Experiments

In this section, we present an empirical evaluation of the FinnPos system on two Finnish treebanks. The evaluation considers tagging and lemmatization accuracy and computational efficiency of learning and decoding. For comparison, we provide results using three reference toolkits.

4.1 Data

The experiments are conducted on the Turku Dependecy Treebank (Haverinen et al, 2009, 2013) and FinnTreeBank (Voutilainen, 2011) described in Section 2. The treebanks do not have default partitions to training and test sets. Therefore, from each 10 consecutive sentences, we assign the 9th and 10th to the development sest and the test

	TDT	FTB
train	10,858 sent. (145,775 tok.)	15,297 sent. (129,374 tok.)
dev	1,357 sent. (18,060 tok.)	1,912 sent. (16,579 tok.)
test	1,357 sent. (19,283 tok.)	1,912 sent. (16,075 tok.)
OOV in test	21.9%	22.1%

Table 3: Sizes of the training, development and test sets for FTB and TDT. The last row indicates the amount of tokens in the test set that are not found in the train set.

sets, respectively. The remaining sentences are assigned to the training sets. Statistics for the data splits are given in Table 3.

Tables 4 and 5 show the distributions of main POS classes for the test sets of TDT and FTB, respectively. Although the morphological labeling schemes in both FTB and TDT follow the labeling scheme of the OMorFi morphological analyzer, they are based on different versions of OMorFi. Therefore, the treebanks have differing main POS inventories. For example, the class Particle in FTB overlaps with the classes Conjunction and Adverb in TDT.

The encoding of nouns in FTB and TDT differs with regard to coordinated compounds. In Finnish, a coordination of two compound words which share an identical part can be written in an abbreviated manner. For example, *isotuloiset ja pienituloiset* (*people with high income and people with low income*) can be abbreviated as *iso- ja pienituloiset* because the coordinated compounds share the final part *-tuloiset*. FTB denotes the compound prefix *iso-* by a separate main POS Truncated. In contrast, TDT labels these prefixes as regular nouns or adjectives.

Both FTB and TDT group common and proper nouns under the main POS Noun. The distinction is, however, denoted by an additional subcategory label.

		all words		OOV words	
label	example	absolute frequency	relative frequency (%)	absolute frequency	relative frequency (%)
Noun	talo (a house)	6565	34.0	2656	62.9
Verb	istua (to sit)	3810	19.8	872	20.6
Punctuation	. ” ,	2897	15.0	0	0.0
Adverb	nopeasti (quickly)	1407	7.3	79	1.9
Adjective	hidas (slow)	1243	6.4	447	10.6
Pronoun	sinä (you)	1241	6.4	36	0.9
Conjunction	kun (when)	1096	5.7	2	0.0
Numeral	kolme (three)	652	3.4	73	1.7
Adposition	alla (under)	285	1.5	6	0.1
Foreign	live (live)	37	0.2	29	0.7
Symbol	:D	32	0.2	19	0.4
Interjection	nam (yum)	18	0.1	5	0.1

Table 4: The main POS distributions of all and out-of-vocabulary (OOV) words for the test set of Turku Dependency Treebank.

		all words		OOV words	
label	example	absolute frequency	relative frequency (%)	absolute frequency	relative frequency (%)
Noun	talk (a house)	4354	27.1	2079	58.6
Verb	its (to talk)	3831	23.8	755	21.3
Punctuation	. " ,	2302	14.3	0	0.0
Particle	näin (thus)	1502	9.3	23	0.6
Pronoun	sinä (you)	1437	8.9	37	1.0
Adverb	nopeasti (quickly)	1040	6.5	112	3.2
Adjective	hidas (slow)	1033	6.4	431	12.1
Numeral	kolme (three)	278	1.7	74	2.1
Adposition	alla (under)	273	1.7	16	0.5
Unknown	live (live)	16	0.1	14	0.4
Truncated	iso- (big)	9	0.1	8	0.2

Table 5: The main POS distributions of all and out-of-vocabulary (OOV) words for the test set of FinnTreeBank.

4.2 Reference Systems

This section summarizes the reference systems, namely, Morfette (Chrupala et al, 2008), MarMot (Müller et al, 2013), and HunPos (Halácsy et al, 2007).

Morfette. Morfette is a toolkit for learning a morphological tagging and lemmatization model from annotated training data.[3] Given a corpus of sentences annotated with lemmas and morphological labels, and optionally a morphological analyzer, Morfette learns to assign analyses for new sentences. The Morfette tagging model is based on the averaged perceptron classifier. Meanwhile, lemmatization is handled as a classification task, in which each lemmatization class corresponds to a set of string edit operations required to transform the inflected word form into the corresponding lemma. This general approach is adopted by FinnPos.

MarMot. MarMot is a CRF-based morphological tagging toolkit.[4] Given a corpus of sentences annotated with morphological labels, and optionally a morphological analyzer, MarMot learns to assign morphological tags for new sentences. The model estimation of MarMot is based on the maximum likelihood criterion utilizing a pruning approach which enables efficient learning of high-order models. In contrast to FinnPos and Morfette systems, MarMot is solely a morphological tagging toolkit and does not perform lemmatization.

HunPos. HunPos is an improved, open-source implementation of the morphological TnT tagger of Brants (2000).[5] Given a corpus of sentences annotated with morphological labels, and optionally a morphological analyzer, HunPos learns to assign

[3] Available at `https://sites.google.com/site/morfetteweb/`.

[4] Available at `https://code.google.com/p/cistern/wiki/marmot`.

[5] Available at `http://code.google.com/p/hunpos/`.

morphological tags for new sentences. Similarly to MarMot, HunPos is solely a morphological tagging toolkit and does not perform lemmatization. The HunPos tagger is based on the generative HMM framework which makes it sensitive to rich feature sets compared to the discriminatively trained perceptron classifier and CRFs. On the other hand, due to the generative estimation procedure and simple feature sets, the system is extremely fast to both train and apply. While the HunPos system was originally designed for morphological tagging of Hungarian, it is a natural choice for a Finnish morphological tagger due to the relatedness of Hungarian and Finnish languages: Hungarian and Finnish are both agglutinative and morphologically rich languages belonging to the Finno-Ugric family.

4.3 Evaluation

Test performances in tagging and lemmatization (when applicable) are evaluated using *per-token* accuracies. These accuracies are reported separately for all words and words not seen in the training data. We establish statistical significance (with confidence level 0.95) using the standard 2-sided Wilcoxon signed-rank test performed on 10 randomly divided, non-overlapping subsets of the complete test sets.

4.4 Hardware

The experiments are run on a desktop computer (Intel Core i5-4300U with 1.90 GHz and 16 GB of memory).

4.5 Results

Obtained tagging and lemmatization accuracies, training times, and decoding speeds for TDT and FTB are presented in Tables 6 and 7, respectively. In what follows, we will compare the FinnPos system individually with the reference systems.

FinnPos versus Morfette. We begin by comparing FinnPos and Morfette, both of which perform morphological tagging and lemmatization. The FinnPos system outperforms the Morfette with respect to both tagging and lemmatization accuracy. The differences in accuracies are statistically significant. Furthermore, compared to FinnPos, the training time of Morfette is substantially higher and decoding speed substantially lower.

FinnPos versus MarMot. The FinnPos system outperforms MarMot with respect to tagging accuracy. However, the differences in accuracies are not statistically significant. Compared to FinnPos, the training time of MarMot is substantially higher and decoding speed substantially lower. Finally, MarMot does not perform lemmatization.

	tag acc.		lemma acc.		train. time		dec. speed (tok/s)
toolkit	all	OOV	all	OOV	tagger	lemmatizer	
HunPos	91.64	76.07	-	-	2 s	-	101,000
MarMot	96.29	91.04	-	-	38 min	-	1,000
Morfette	93.91	82.19	89.33	72.04	203 min	16 min	40
FinnPos	**96.31**	**91.64**	**93.29**	**84.28**	4 min	5 min	16,000

Table 6: Results for Turku Dependency Treebank.

	tag acc.		lemma acc.		train. time		dec. speed (tok/s)
toolkit	all	OOV	all	OOV	tagger	lemmatizer	
HunPos	93.65	82.55	-	-	2 s	-	141,000
MarMot	96.21	91.46	-	-	24 min	-	1,000
Morfette	95.03	86.81	95.66	83.12	128 min	8 min	60
FinnPos	**96.23**	**92.34**	**97.49**	**92.85**	3 min	3 min	18,000

Table 7: Results for FinnTreeBank.

FinnPos versus HunPos. The training time of the HunPos system is substantially lower compared to FinnPos or any other system. While faster to estimate and apply, however, the tagging accuracy of HunPos is significantly lower compared to FinnPos on both data sets. The HunPos system does not perform lemmatization.

4.6 Error Analysis

In this section, we present and discuss the distribution of the errors yielded by the FinnPos system. In particular, we examine how the errors are distributed across the main POS classes. In addition, we examine individual error types, that is, which categories are most often confused for one another.

First, consider Tables 8 and 9 which contain the errors distributions for TDT and FTB, respectively. For both data sets, the majority of errors take place in the noun and verb categories. This is expected as these categories are most frequent in the test sets and, as shown in Tables 4 and 5, and contain the most OOV word forms.

Second, consider Tables 10 and 11 which contain confusion matrices of errors for TDT and FTB, respectively. Due to space constraints, the matrices include 25 most prominent confusion pairs. For both data sets, the majority of errors take place when a noun is confused with a noun or a verb is confused with a verb, that is, the tagger yields the correct main POS class but an incorrect detailed morphological label. For example, consider the noun phrase *kiveä ja terästä oleva monumentti* (*a monument made of stone and steel*).[6] The word form *terästä* could be the partitive form of the noun *teräs* (steel) or the elative form of the noun *terä* (a blade). From a syntactical point of view, both interpretations are possible. From a semantical point of view, however, only the partitive interpretation is valid.

[6] The example is taken from FinnTreeBank.

	all words		OOV words	
main POS	absolute frequency	relative frequency (%)	absolute frequency	relative frequency (%)
Noun	271	38.1	182	53.5
Verb	205	28.8	61	17.9
Adjective	65	9.1	33	9.7
Pronoun	48	6.8	17	5.0
Adverb	45	6.3	11	3.2
Foreign	23	3.2	21	6.2
Numeral	15	2.1	4	1.2
Adposition	15	2.1	2	0.6
Conjunction	13	1.8	1	0.3
Symbol	7	1.0	7	2.1
Interjection	4	0.6	1	0.3

Table 8: Error distribution over main POS classes for Turku Dependency Treebank.

	all words		OOV words	
main POS	absolute frequency	relative frequency (%)	absolute frequency	relative frequency (%)
Noun	184	30.4	127	49.0
Verb	155	25.6	54	20.8
Adverb	71	11.7	11	4.2
Particle	47	7.8	4	1.5
Pronoun	46	7.6	3	1.2
Adjective	40	6.6	27	10.4
Numeral	24	4.0	11	4.2
Adposition	17	2.8	3	1.2
Unknown	12	2.0	11	4.2
Truncated	8	1.3	8	3.1
Punctuation	2	0.3	-	-

Table 9: Error distribution over main POS classes for FinnTreeBank.

	Noun	Verb	Adjective	Pronoun	Adverb	OTHER
Noun	26.0	2.7	3.1	0.7	1.5	4.1
Verb	5.1	20.5	2.3	0.4	0.1	0.4
Adjective	2.3	2.4	3.0	0.0	1.4	0.1
Pronoun	1.5	0.1	0.0	2.4	1.3	1.4
Adverb	1.0	0.1	0.6	1.1	0.3	3.2
OTHER	4.2	1.0	0.1	0.1	2.1	3.2

Table 10: The confusion matrix of errors for Turku Dependency Treebank with relative error frequencies. For example, labeling a noun as a verb comprises 2.7 percentages of all errors, whereas labeling a verb as a noun comprises 5.1 percentages of all errors. For the five main POS classes with most labeling errors, results are shown separately. The class OTHER comprises all remaining main POS classes.

4.7 Discussion

Compared to the reference toolkits, the FinnPos system provides the highest accuracies with respect to tagging and lemmatization accuracy. In addition, the system is

	Noun	Verb	Adverb	Particle	Pronoun	OTHER
Noun	17.7	1.7	3.0	1.5	0.0	6.6
Verb	2.5	17.2	0.3	1.2	0.5	4.0
Adverb	1.2	0.0	1.7	3.5	2.0	3.5
Particle	1.0	0.5	1.3	4.1	0.7	0.2
Pronoun	0.2	0.0	2.5	0.3	3.6	1.0
OTHER	5.6	2.8	3.0	0.3	0.7	4.6

Table 11: The confusion matrix of errors for FinnTreeBank with relative error frequencies.

computationally more efficient to train and apply compared to the MarMot and Morfette systems which also utilize discriminative learning. As discussed in Section 2, the TDT and FTB corpora differ somewhat in the included text domains as well as the labeling schemes. However, these differences appear to have a minor effect on the tagging and lemmatization accuracy of the FinnPos system.

According to the error analysis in Section 4.6, while the main POS label is often correct, the detailed morphological information is more difficult to infer. The analysis shows that substantial improvement in tagging accuracy would require improved inference of the detailed morphological information for nouns and verbs specifically. This, however, is a difficult task because the immediate syntactical context often does not provide adequate clues for disambiguation. The choice between different detailed labels is often lexically and semantically conditioned which makes it particularly difficult for OOV words.

5 Conclusions

We presented FinnPos, an open-source morphological tagging and lemmatization toolkit for Finnish. 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. The performed empirical evaluation showed that FinnPos performs favorably to several reference systems (MarMot, Morfette, HunPos) in terms of tagging and lemmatization accuracy, as well as computational efficiency of learning new models and assigning analyses to novel sentences.

The FinnPos system should be readily applicable for learning taggers for languages closely related to Finnish, such as Hungary and Estonian. On the other hand, the default feature extraction scheme may also perform well on other morphologically rich European languages, such as Czech and Romanian. Therefore, in future work, the toolkit could be evaluated empirically on multiple languages.

References

Bohnet B, Nivre J, Boguslavsky I, Ginter RFF, Hajic J (2013) Joint morphological and syntactic analysis for richly inflected languages. Transactions of the Association for Computational Linguistics 1

Brants T (2000) TnT: A statistical part-of-speech tagger. In: Proceedings of the Sixth Conference on Applied Natural Language Processing, pp 224–231

Charniak E, 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, Association for Computational Linguistics, pp 173–180

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

Collins M (2002) Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In: Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing, vol 10, pp 1–8

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

Hakulinen A, Korhonen R, Vilkuna M, Koivisto V (2004) Iso suomen kielioppi. Suomalaisen kirjallisuuden seura, URL `http://scripta.kotus.fi/visk`

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

Haverinen K, Ginter F, Laippala V, Viljanen T, Salakoski T (2009) Dependency annotation of Wikipedia: First steps towards a Finnish treebank. In: Eighth International Workshop on Treebanks and Linguistic Theories, p 95

Haverinen K, Nyblom J, Viljanen T, Laippala V, Kohonen S, Missilä A, Ojala S, Salakoski T, Ginter F (2013) Building the essential resources for Finnish: the Turku Dependency Treebank. Language Resources and Evaluation

Huang L, Fayong S, 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, pp 142–151

Karlsson F (1990) Constraint grammar as a framework for parsing running text. In: Proceedings of the 13th Conference on Computational Linguistics

Lindén K, Axelson E, Hardwick S, Pirinen T, Silfverberg M (2011) HFST – Framework for compiling and applying morphologies. In: Systems and Frameworks for Computational Morphology

Müller T, Schmid H, Schütze H (2013) Efficient higher-order CRFs for morphological tagging. In: In Proceedings of Empirical Methods in Natural Language Processing

Pal C, Sutton C, McCallum A (2006) Sparse forward-backward using minimum divergence beams for fast training of conditional random fields. In: Internation Conference on Acoustics, Speech and Signal Processing, vol 5, pp V–V

Pirinen T (2008) Automatic finite state morphological analysis of Finnish language using open source resources (in Finnish). Master's thesis, University of Helsinki

Ratnaparkhi A (1996) A maximum entropy model for part-of-speech tagging. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, vol 1, pp 133–142

Rush AM, Petrov S (2012) Vine pruning for efficient multi-pass dependency parsing. In: Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, pp 498–507

Silfverberg M, Linden K, et al (2011) Combining statistical models for pos tagging using finite-state calculus. In: Proceedings of the 18th Conference of Computational Linguistics

Silfverberg M, Ruokolainen T, Lindén K, Kurimo M (2014) Part-of-speech tagging using conditional random fields: Exploiting sub-label dependencies for improved accuracy. Association for Computational Linguistics, Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pp 259–264

Sutton C, McCallum A (2011) An introduction to conditional random fields. Machine Learning 4(4):267–373

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

Weiss D, Taskar B (2010) Structured prediction cascades. In: International Conference on Artificial Intelligence and Statistics, pp 916–923