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

Ancient Proverb

Contents

List of Figures

List of Tables

List of Algorithms

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 (starting at 1) in vector or sequence x.
$\mathbb{R}^m_n$	The space of $m \times n$ real valued matrices.
$x[1:t]$	Prefix $(x_1,\ ...,\ x_t)$ of vector or sequence $x = (x_1,\ ...,\ x_T)$.
X^t	The cartesian product of t copies of the set X.
$M^\top$	Transpose of matrix M.
M^+	The More-Pennrose pseudoinverse 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

Teemu Ruokolainen statistical methods, friendship, writing.

Måns Huldén finite-state, statistical methods, inspiration, courage, IPA.

Krister Lindén ideas, letting me do useless stuff once in a while, pushing me forward, encouragement, realism.

Anssi Yli-Jyrä inspiration, finite-state, positive attitude, friendship.

Kimmo Koskenniemi intellectual rigorosity, how to think of stuff, how to be legible, being critical.

Arto Voutilainen

Erik Axelson

Tommi Pirinen

Senka Drobac

Sam Hardwick

Graham Wilcock

Juliette Kennedy

Panu Kalliokoski

Coursera

Reetta Vuokko-Syrjänen and Kaj Syrjänen + Elsa

Family

Veera Nykänen

Jarmo Widemann

Antti, Sami, Ville, Jarmo, Rod, Marko and many more friends who have brought a lot of happiness through the time I’ve been working on this thesis project.

Chapter 1

Introduction

Chapter 2

Morphology

2.1 The Structure of Words

Many linguistc units are rather controversial. What is the definition of a sentence? Do sentences even exist in spoken language? Is there such a thing as a phoneme?

The concept of a word is, however, among the among the more universally accepted. The reason for this may be that there is in fact a fairly easy and readily applicable test for what is a word: X is a word if X can be used as an answer to a question. For example

> - You said you saw a what yesterday?
> - Dog. I saw a dog.

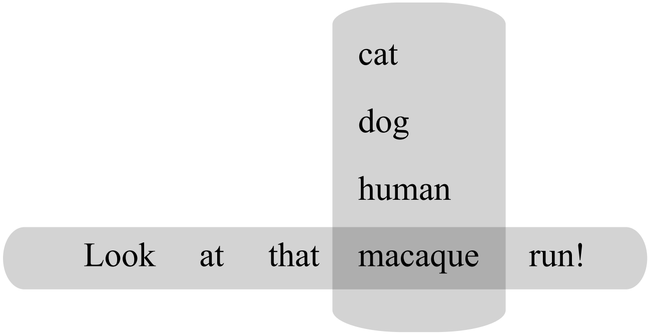
Although, singular nouns in English usually take a compulsory article, exceptional contexts where they are uttered without an article are acceptable for many language users. Therefore, it is plausible to stipulate that singular nouns are words in English. For most languages, this criterion can be used to find most words that language users intuitively group under the concept word. Even this criterion, however, fails for polysemous languages. Nevertheless, it very useful for many languages.

Of course, orthographic words are much easier to identify than spoken language words in many writing systems. Simply look for entities separated by punctuation and white-space. This, however, does not work for some languages such as Chinese where word spaces are not indicated in written text. Because this thesis exclusively deals with written language and because I have performed experiments on European languages whose orthographies mark word boundaries, I will gloss over the difficulties of locating word boundaries although this is an interesting problem from the points of view of both linguistics and engineering.

Morphology is the sub-fields of linguistics that investigates the structure of words. For example the knowledge that appending a suffix "s" to a singular English nouns makes it plural is morphological knowledge. Of course this rule is not entirely correct because some nouns form plurals in some other way (e.g. "foot/feet") and yet other nouns have no plural form (e.g. "music").

Because words consist of phonemes or orthographic symbols, the structure of words cannot be investigated without considering the phonological system of a language. If we had to form the singular inessive

Figure 2.1: The syntagmatic and paradigmatic axes of language.



of the the non-existent Finnish noun “looka”, we would say “lookassa”, not “lookassä” because vowel harmony, a phonological co-occurrence restriction, prohibits the latter form.

2.2 Languages with Rich Morphology

- Typological classification of languages.
- “Large label sets”.

2.3 Morphological Analyzers

- Finite-state morphology (Koskenniemi, 1984), (Kaplan and Kay, 1994).

Chapter 3

Machine Learning

This section outlines the basic methodology followed in machine learning approaches to 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.

3.0.1 Overview

Statistical and Rule-Based NLP Approaches to Natural Language Processing can be broadly divided into two groups: rule-based and machine learning approaches. Rule-based approaches utilize hand crafted rules for tasks such as text labeling or machine translation. Machine learning approaches solve the same set of problems using data driven (usually statistically motivated) models and samples from the problem domain, which are used to estimate model parameters.

This thesis focuses on extending machine learning techniques to the domain of morphologically complex languages, where they have been applied less frequent than for morphologically more straightforward languages like English. I want to emphasize that the thesis should not be seen as an attack against the rule-based paradigm for Natural Language Processing which has proven to be very successful in treatment of morphologically complex languages . This thesis is simply an investigation into extension of machine learning techniques. Truly successful language processing systems should probably combine rule-based and statistical techniques.

Machine learning and hand crafted rules have their respective merits and short-comings and these are dependent on the task to some extent. For example, in the domain of machine translation, rule-based methods can give syntactically correct results . However, statistical machine translation systems are in general better at lexical choice .

Supervised and Unsupervised ML Machine learning is can be divided into two fields supervised and unsupervised. Supervised machine learning uses a training material of labeled training examples. An example would be a set of images with associated key words, where the task is to build a system which can associate previously unseen images with appropriate key words.

Supervised machine learning is typically employed for tasks such as labeling (that is classification) and translation. Typical examples of tasks include Part-of-Speech labeling and speech recognition where annotations consist of POS labels for the words in some text and written sentences corresponding to an acoustic speech signal respectively.

In contrast to supervised machine translation, unsupervised approaches exclusively utilize unannotated data. Unsupervised machine learning is most often used for various kinds of clustering tasks, that is grouping sets of examples into subsets of similar examples.

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

In Natural Language processing, supervised learning approaches are most often used (for example ...). Unsupervised approaches, however, can also be useful for e.g. exploratory data analysis (...).

This thesis focuses on supervised machine learning.

3.1 Supervised Learning

In this section, I will illustrate the key concepts and techniques in supervised Machine Learning using the 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 chose linear regression as example because it is a simple model yet can be used to illustrate data driven techniques. 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

3.1.1 Basic Concepts

Linear Regression As a simple example, imagine a person called Jill who is a real estate agent. 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 but there are a few very robust quantifiable predictors of price that are easy to measure.

Jill 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* 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 average prospective client.

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

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

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

Inference The first question concerns *inference*, that is finding the most likely value for the dependent variable given 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 model such as structured graphical models used in this thesis, it can however be an algorithmically and computationally more challenging problem. The principle is still the same: Find the y which is most likely given the input parameters.

Estimation 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. She also needs to decide upon an *estimator*, that is, a method for estimating the parameters θ.

[1] 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 **?**.

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 $x_t = (x_t[1]...x_t[n])$ 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. 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, however, this has bearing on how the parameters need to be estimated. In any case, the number of data points in the training data should ideally be higher than the number of parameters that need to be estimated. When one cannot accomplish this, one encounters the so called *data sparsity problem*.

Data Sparsity Whereas getting sufficient training data is fairly easy in the case of housing prices, it is vastly more difficult to accomplish with more complicated models in natural language processing. Therefore, one central question in this thesis is how to counteract *data sparsity*.

Cost 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.

$$\theta = \underset{\theta' \in \mathbb{R}^n}{\arg\min} \sum_{x_t \in \mathcal{D}} |y_t - y(x_t; \theta')|^2 \tag{3.2}$$

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

The Exact Solution In the case of linear regression, there is a well known exact solution for θ which utilizes linear algebra. The solution is given in Equation 3.3. The matrix $X \in \mathbb{R}^T_n$ is defined by $X_{t,i} = x_t[i]$ and its More-Pennrose pseudoinverse $X^+ \in \mathbb{R}^n_T$ is defined as $X^+ = (X^\top X)^{-1} X^\top$. The vector $Y \in R^T$ is simply the vector of realized house prices y_t. The solution θ exists only when none of the independent variables are linear combinations of each other in the training data.

$$\theta = X^+ Y \tag{3.3}$$

Iterative Estimation Although the linear regression model is simple enough so that it can be estimated exactly, the same does not hold for most more complex models such as the Conditional Random Field investigated in this thesis. Moreover, the exact solution might often not be the one that is desired. Often the training data is quite sparse, that is there is too little of it compared to the amount of parameters that need to be estimated. Therefore, the model may *over-fit* the training data and fail to generalize well to examples not included in the training data.

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

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

The zero solution to a linear regression problem is simple but also totally 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 cost 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. These are general regularization methods that are employed in many models in machine learning.

The L_1 regularized cost function for linear regression is given in Equation 3.4. L_1 regularization, also called LASSO regularization **?**, enforces solutions where many of the parameter values are 0. These are also called sparse parameters. 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 expression is a sum

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

The L_2 regularized cost function is given in 3.5. 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.

$$\theta = \operatorname*{arg\,min}_{\theta' \in \mathbb{R}^n} \sum_{x_t \in \mathcal{D}} |y_t - y(x_t;\theta')|^2 + \lambda \|\theta\|^2 = \operatorname*{arg\,min}_{\theta' \in \mathbb{R}^n} \sum_{x_t \in \mathcal{D}} |y_t - y(x_t;\theta')|^2 + \lambda \sum_i |\theta[i]|^2 \tag{3.5}$$

The coefficient $\lambda \in \mathbb{R}^+$ is called the *regularizer*. The regularizer determines the degree to which model fit and simplicity affect the cost. A higher λ will increase the cost 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.

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 cost. Therefore, there is no direct way to estimate its correct magnitude simply using the training data. Instead *held-out data* can be used. Held-out data is labeled data that is not used directly for estimating model parameters. If the model over-fits the training data, that is generalizes poorly to unseen examples, the held-out data will have a high cost. However, it will also have a high cost 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 cost of the held-out data. Usually, one uses the unregularized cost function for the held-out data.

Iterative Estimation Regularization is an additional reason for introducing iterative estimation methods instead of exact estimation. There are several choices of regularization methods and some of them might not result in optimization problems that have closed form solutions[2]. When an exact solution cannot be computed, or it is undesirable, numerical methods can be used to estimate model parameters.

Because the cost 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 **?**, 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 only once differentiable.[3] 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 cost function. For the models used in this thesis, differentiability holds. Gradient based methods work in the following general manner. Let $L_{\mathcal{D}} : \mathbb{R}^n \to \mathbb{R}$ be the cost 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 cost function. This is the negative gradient $-\nabla L_{\mathcal{D}}(\theta_t)$ at point θ_t.
3. Determine a suitable step size $\alpha_t \in \mathbb{R}_+$.
4. Take a step of length α_t in direction v_t to get to the next point in the parameter space θ_{t+1}, that is $\theta_{t+1} = \theta_t - \alpha_t \nabla L_{\mathcal{D}}(\theta_t)$.

[2] In the case of regression, non-linear *kernel functions* also result in optimization problems that don't have a closed form solution. I will elaborate this when discussing logistic regression.

[3] Essentially the same procedure can also be applied to functions that are not differentiable but are only guaranteed to have a sub-gradient **?**.

5. If the difference in cost $|L_{\mathcal{D}}(\theta_{t+1}) - L_{\mathcal{D}}(\theta_t)|$ 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 cost 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 $-\nabla L_{\mathcal{D}}(\theta_t)$ **?**. For example $\alpha_t = t^{-1}$ may work[4].

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 cost function that is toward a local optimum. Therefore, they will in general not find the global optimum of the cost function. By choosing a *convex* cost function it is possible to avoid getting stuck at local optima. All local optima of convex functions are in fact global optima.

Convexity is, however, not enough to guarantee convergence to a global optimum. First of all, a global optimum might not exist[5]. Convergence could also be too slow thus leading 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 cost 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 cost 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. It has vastly superior convergence properties when compared to 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 cost over the single training example $\mathcal{D}[i]$ to approximate the gradient $\nabla L_{\mathcal{D}}(\theta)$ for the entire training data $\mathcal{D}$.

3.2 Classifiers

The main topic of this thesis is morphological tagging. It can be seen as a structured *classification task* where each word in a sentence is assigned one morphological label[6]. Classification is a fundamental supervised machine learning task where the objective is to learn a mapping from objects like words, sentences, documents or images onto discrete classes such as morphological labels (Noun+Sg+Nom) or sentiments (Positive sentiment versus Negative Sentiment).

[4] 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 cost function (if it exists) for a wide variety of functions **?**

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

[6] which may have internal structure as is seen in Section 8.13.

Classification resembles regression. The output of a classifier is, however, not a single real value y but a distribution over the set of nominal classes.

Probabilistic classifiers can broadly be divided into two groups, namely generative and discriminative. Generative classifiers learn a joint distribution $p(x, y)$ over labels y and input examples x. Given an example x, a distribution over classes y can be computed using the marginal probability of example x, which is $p(x) = \sum_{y' \in Y} p(y', x)$. The probability distribution over classes is defined by $p(y|x) = p(y, x)/p(x)$.

3.3 Structured Classifiers

There are tasks in natural language processing that cannot be adequately formulated as supervised classification tasks in the simple way that has been discussed earlier. Examples of tasks that require more sophisticated methods are syntactic parsing and translation between languages. One could think of parsing as a labeling task where the objective is to label a sentence with the appropriate syntactic tree. Likewise, the translation of a sentence could be seen as its label.

It is, however, easy to see that these approaches are deeply flawed. Firstly, there is an infinite number of syntactic trees and indeed an infinite number of sentences of finite length. Therefore, the label set would have to be infinite as well. Secondly, the data sparsity problem would be unmanageable. Just in order to see most relatively frequent syntax trees for moderately long English sentences, say twenty words long, we would need an unfathomable amount of training data.

Instead of a simplistic classification approach, *structured* classifiers can be employed. Structured classifiers relate parts of the complex label such as a translation, syntax tree or sequential labeling with the input. They also learn how to piece together complex labels from simple constituents. A very simple (and probably very bad) translation software could learn how to translate isolated English words into French words and then learn how to combine French words into sentence as shown in Figure 3.1. Note that although both “Le” and “La” are perfectly good translations for “The”, only “Le” is usually possible before “chien”, which is a masculine French word.

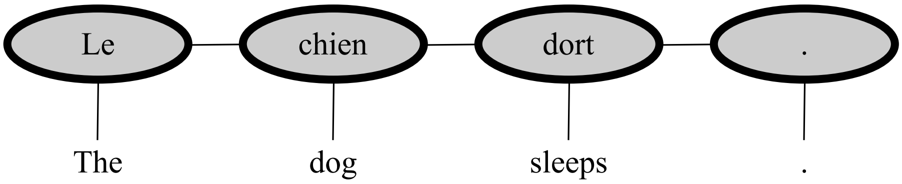
In general, structured classifiers rely on two models, the unstructured and structured model. The unstructured model learns to relate simple labels such as words in a translation or nodes in a syntax tree with the input. In contrast, the structured model learns how to combine simple labels into complex labels such as entire translations or syntax trees. In practice, some models make a stricter division then other models.

Structured models are the main subject matter of this thesis and I will present two structured models, the Hidden Markov model and the Conditional random field in detail in Chapters **??** and **??**.

3.4 The Aim of Research – Improving the State of the Art

Like most language technological research, the work documented in this thesis aims at improving existing solutions for language processing. Specifically, my work is targeted at improving morphological taggers for morphologically rich languages. It is not entirely easy to define what constitutes an improvement to

Figure 3.1: A translation from English to French



the field of morphological tagging or indeed any sub-field of language technology. Nevertheless, most researchers would probably agree that the following kinds of changes are improvements compared to existing approaches:

1. Improving labeling accuracy.
2. Speeding up estimation.
3. Speeding up tagging.
4. Reducing model size.
5. Clarifying the underlying theoretical foundations.
6. Simplifying implementation of taggers.
7. Uncovering best practices for building taggers.

Items 1 though 4 in the list above are *quantifiable* improvements. It is possible to perform experiments to measure the labeling accuracy, tagging speed, estimation speed and model size given by different morphological taggers and derive conclusions about the respective merits of the taggers. In contrast, the rest of the items in the list cannot be measured as easily.

Although, most probably would agree that clarifying theoretical foundations of a field is a substantial contribution, it may not be as easy to agree upon what constitutes a clarification. This could for example be dependent on the background of individual researchers. Similarly, one model might be more straightforward to implement than another model using some programming language, however, this can very well depend on the specific programming language and available libraries to some degree.

Because quantifiable improvements are easier to ascertain, this thesis focuses on demonstrating such improvements compared to other state of the art approaches. Nevertheless, I will also aim to show that the model demonstrated in the thesis is conceptually simpler than other state of the art approaches.

Although quantifiable improvements are easier to demonstrate than other improvements, there are still caveats. First of all, it is impossible to compare machine learning models directly. One can only compare implementations of the models. Because of differences between platforms even different implementations of the exact same model can have radically different run time on the same data. Moreover, bugs in the implementation of different models can reduce the accuracy or have sporadic effects on run time.

Besides practical concerns like dependence on implementation, there are also theoretical issues that interfere with measuring performance of different models. It is easy to say that the accuracy of one system is better than another system on *a specific data set*. This does, however, not imply that the accuracy is better on *all data sets*. In formal terms, we can only conduct experiments on samples of the distribution of all texts in a given language. Therefore, our experiments will yield results only in a probabilistic sense: it is likely that the labeling accuracy of one system is better than the accuracy of another, if the accuracy was better on the sample used for testing.

Using large test sets and test sets compared from a variety of different genres will probably give more reliable results. This is, however, also to some extent a debatable matter. Some methods are very accurate on the same genre that they were trained on but perform worse on other genres. Other methods instead perform well on average but, as a trade-off, cannot reach as high accuracy on a specific text domain. It is not easy to say which kind of system is preferable. This trade-off is called the bias variance trade-off **?** and it has bearing on measuring the performance difference of systems.

When using a high variance system, accuracy on different texts varies a lot. When instead using a system with high bias, the accuracy tends to vary much less.

When comparing two systems, it is not sufficient to simply look at the performance of the systems on a test set. If we measure the performance of the systems on a particular test, one of them will almost certainly perform better than the other regardless of whether there is an actual difference in the performances of the systems. The probability of exactly equal performance is simply very small.

The larger the test sets are, the more accurately the results of experiments will on average reflect the true performance of the systems. This is easy to see, because ultimately the test set will represent the entire domain. Unfortunately, the amount of test material is usually restricted and producing more test data might not be feasible[7].

An approach that is often used is splitting the test data into segments and performing several experiments – one for each segment. If one system outperforms the other system on most segments with a large margin, then we are more confident in saying that there is in fact a difference in the performance of the systems. If on the hand each system outperforms the other on roughly half of the segments, we might be inclined to doubt whether there is any real difference in performance between the systems. The higher the variance between the results, the larger the margins between the systems need to be in order for us to be able to conclude that there is a difference in performance. This argument can be made rigorous using statistical tests which measure the significance of the difference in performance of two systems.

The usual set up of statistical significance testing is to make a null hypothesis H_0 that the average performance of two systems is the same. After this a test statistic is computed. The test statistic is simply a real number whose value indicates the significance of the difference in performances between the systems. If the statistic is high enough, the null hypothesis can be discarded in favor of the alternative hypothesis that there is a genuine difference in performance between the systems. The test statistic incorporates information about the difference in performance of the systems on test data all segments as well as the variance of the performances.

[7] Especially when using standard data sets, one is restricted to the given amount of test data

In the work conducted presented in this thesis, I have used the Wilcoxon signed-rank test **?** to ascertain the significance of results. I use it instead of the T-test because it is not dependent on the fact that the distribution of differences in performance are normally distributed[8]. In practice it is less sensitive than the T-test.

[8] In practice it might be a fairly accurate assumption that the differences are normally distributed. This often holds for measurements **?**.

Chapter 4

Part-of-Speech Tagging

4.1 Background

4.2 Statistical Part-of-Speech Taggers

Morphological disambiguation and part-of-speech tagging are interesting tasks from the perspective of machine learning because they represent labeling tasks where both the set of inputs and outputs are unfathomably large. Since each word in a sentence $x = (x_1, \ldots, x_T)$ of length T receives one label, the complete sentence has n^T possible labels $y = (y_1, \ldots, y_T)$ when the POS label set has size n.

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[1]. 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 is likely to ever produce. Direct ML-estimation of the conditional distributions $p(y \,|\, x)$, 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”.

The disambiguation of the POS label for “dog” utilizes both so called *unstructured* and *structured* information. The information that “police dog” is a frequent nominal compound is unstructured information, because it refers only to the POS label (the prediction) of the word “dog”. The information that verbs

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

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. Structured information refers to the combination of predictions. 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 useful, structure is probably mostly useful in a limited way. For example the labels of “dog” and “anything” in the example are not especially helpful for disambiguating each other. It is probably 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 introduce long range dependencies inside sentences.

It is said that machine learning is sophisticated counting of co-occurrences. This statement applies extremely well to POS tagging. Counting is an adequate approach to capturing correlations between the labels of words inside a small window (in the league of five words), because most adjacent words indeed do depend on each other is some way. However, sophisticated counting fails for larger windows, because the number of meaningful dependencies in large windows is negligible in comparison to the space of possibilities.

4.3 Morphological Disambiguation

Chapter 5

Hidden Markov Models

AN INTRO

5.1 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 5.1 summarizes these probabilities.[2]

ι	
CLEAR	0.75
RAIN	0.25

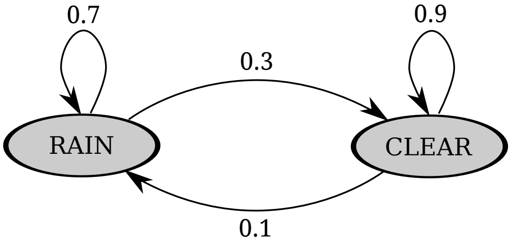
T	CLEAR	RAIN
CLEAR	0.9	0.1
RAIN	0.3	0.7

E	~~☂~~	☂
CLEAR	0.8	0.2
RAIN	0.2	0.8

Figure 5.1: Foo FIXME

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 5.2: 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 5.2 shows a graph representing the transition structure of the HMM outlined in Figure 5.1.

5.2 Formal Definition

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

[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.

Since the succession of days is infinite for all practical purposes, there was no need to consider termination in the example presented in Figure 5.2. 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), 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 5.1).

2. X is the set of emissions, also called observations ($X = \{$☂̸, ☂$\}$ in the example in Figure 5.1).

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 5.1).

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 5.1).

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 5.1).

6. $f \in Y$ is the final state. The state f emits no observations and there are no transitions from f.

The definition of HMMs in this thesis differs slightly from Rabiner (1989) since I utilize final states. Final states were used in for example **?**. Absorbing states **?** could also be used. FIXME.

Figure 5.3 is a more accurate description of the HMM in Figure 5.1 than Figure 5.2. The image has been augmented with initial distribution and a final state.

Because the progression of days is infinite for all practical purposes, the probability of transitioning to the final state f in example 5.2 is 0 regardless of the current state. Hence, the probability of any single sequence of states and emissions is 0. Still, the probability of an initial segment of a state sequence may still be non-zero[3].

An HMM models a number of useful quantities:

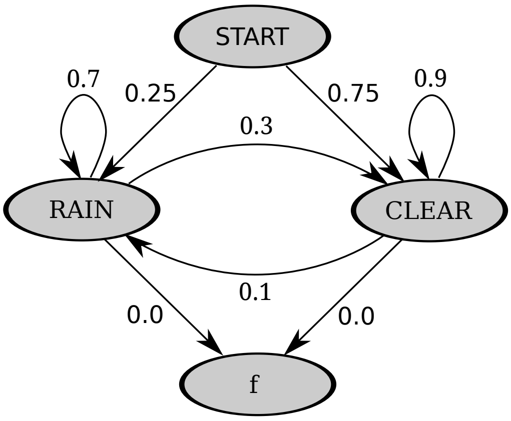
1. The *joint probability* $p(x, y; \theta)$ of a observation sequence x and state sequence y. This is the probability that an HMM with parameters θ will generate the state sequence y and generate the observation x_t in every state y_t.

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 probability of an initial segment up to position t can be computed using the forward algorithm, which is presented in Section 5.3.1.

Figure 5.3: 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, $x = (x_1,\ ...,\ x_T) \in X^T$ be a sequence of observations and $y = (y_1,\ ...,\ y_T, y_{t+1} = f) \in Y^{T+1}$ a sequence of hidden states. Note, that the last state 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 (5.1).

$$p(x, y; \theta) = p(y; \theta) \cdot p(x \mid y; \theta) = \left(\iota(y_1) \cdot \prod_{t=1}^{T} \tau_{y_t}(y_{t+1}) \right) \cdot \prod_{t=1}^{T} \varepsilon_{y_t}(x_t) \tag{5.1}$$

Equation (5.1) 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.

FIXME: Talk about language models and stuff.

There is no limit on the number of hidden states in Y that can emit a given observation. Therefore, it is quite possible that 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 (5.2).

$$p(x; \theta) = \sum_{y \in Y^{T+1},\ y_{T+1} = f} p(x, y; \theta) \tag{5.2}$$

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 (5.3).

$$p(y \mid x; \theta) = \frac{p(x, y; \theta)}{p(x; \theta)} \tag{5.3}$$

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.

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 (5.4)

$$p(z, t,\ x; \theta) = \sum_{y' \in Y^{T+1},\ y'_t = z,\ y'_{T+1} = f} p(x, y'; \theta) \tag{5.4}$$

5.3 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 (5.5) and depicted in Figure 5.4a.

$$y_{MAP} = \arg\max_{y \in Y^T} p(y \mid x; \theta) \tag{5.5}$$

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 (5.6). Figure 5.4c shows the marginal for one position and state.

$$y_{MM} = \arg\max_{y \in Y^T} \prod_{t=1}^{T} p(y_t, t \mid x; \theta) \tag{5.6}$$

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.

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 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 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.

5.3.1 The Forward Algorithm

Equations (5.5) and (5.6) 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 $x[1 : t]$ 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 (5.7).

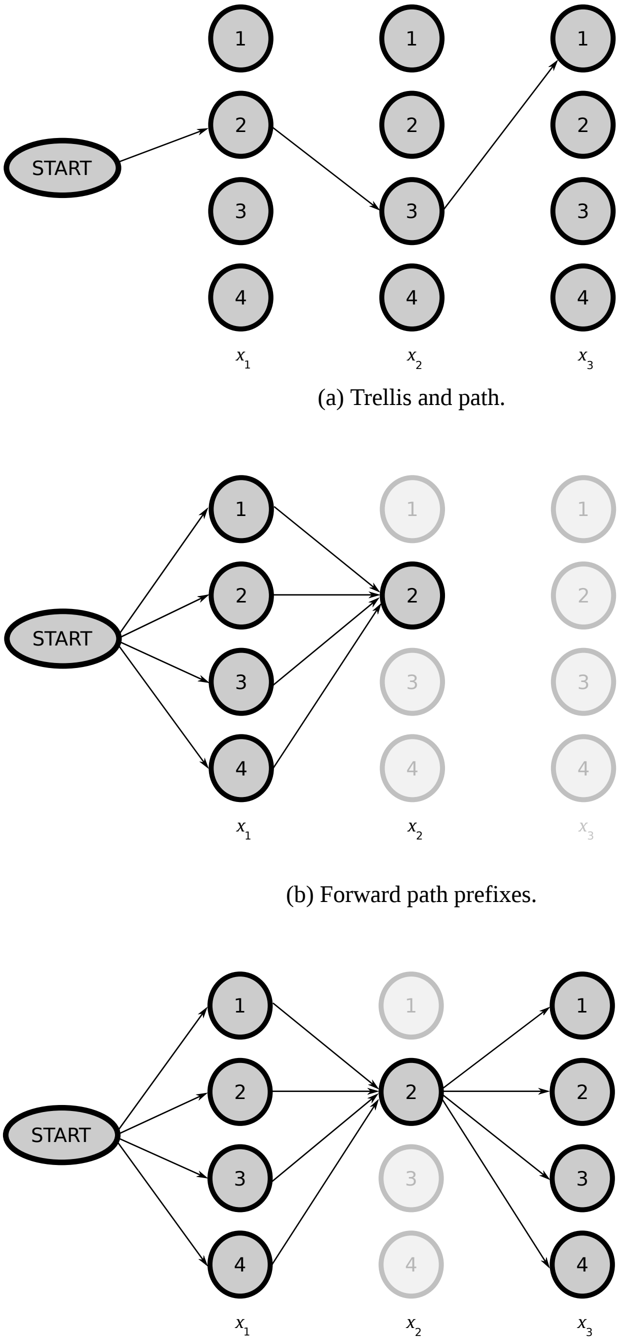
$$p(x,\ y;\theta) = \left(\iota(y_1)\cdot\left(\prod_{u=1}^{t-1}\tau_{y_u}(y_{u+1})\right)\cdot\prod_{u=1}^{t}\varepsilon_{y_u}(x_u)\right),\ t \le T \tag{5.7}$$

When $t = T$, this is very similar to 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 5.4b. Formally, the forward probability is defined by Equation (5.8).

$$\mathrm{fw}_{t,z} = \sum_{y\in Y^t,\ y_t=z} p(x,\ y;\theta) \tag{5.8}$$

Figure 5.4: foo.



(a) Trellis and path.

(b) Forward path prefixes.

(c) Marginal paths.

Figure 5.5: 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	$(0.25 \cdot 0.2 \cdot 0.7 \cdot 0.8) \cdot 0.3 \cdot 0.8 \approx 0.007$
☂	☂	$\cancel{☂}$	
			≈ 0.098

Comparing Equations (5.8) and (5.1) 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 5.5 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, $f_{20,\text{RAIN}}(x)$ requires us to sum approximately 20 million probabilities and $f_{30,\text{RAIN}}(x)$ 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 5.5. By grouping rows one and two, as well as three and four into pairs, it is easy to see that

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

Generalizing, we get the recursion in Equation (5.9).

$$\mathrm{fw}_{t,z} = \begin{cases} \iota(z) \cdot \varepsilon_z(x_1) & , t = 1 \\ \left(\sum_{z' \in Y} \mathrm{fw}_{t-1,z'} \cdot \tau_{z'}(z) \right) \cdot \varepsilon_z(x_t) & , 1 < t \leq T \\ \sum_{z' \in Y} \mathrm{fw}_{T,z'} \cdot \tau_{z'}(f) & , t = T+1, \ z = f. \end{cases} \tag{5.9}$$

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 5.1. 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.

5.3.2 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 (5.10) utilizing the joint prefix probability of x and a state sequence y of length t. Intuitively, it is the sequence of hidden states $y_{t,z}$ which maximizes the joint probability and ends at state z.

$$y_{t,z} = \operatorname*{arg\,max}_{y \in Y^t, \ y_t = z} p(x, \ y; \theta) \tag{5.10}$$

Comparing this equation with the definition of the forward probability $f_{t,z}$ in Equation 5.8, 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

Algorithm 5.1: 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
```

the forward probability $f_{t,z}$. Suppose that $y_{t+1,z'} = (y_1, \ldots, y_t = z, y_{t+1} = z')$. 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 (5.10)[4].

We now get Equation (5.11), which gives us a recursion.

$$y_{t+1,z} = \arg\max_{z \in Y} \begin{cases} \iota(z) \cdot \varepsilon_z(x_1) & , t = 1 \\ y_{t-1,z'} \cdot \tau_{z'}(z) \cdot \varepsilon_z(x_t) & , 1 < t \leq T \\ y_{T,z'} \cdot \tau_{z'}(f) & , t = T+1,\ z = f. \end{cases} \tag{5.11}$$

5.3.3 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 (5.11) 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.[5]

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 5.6.

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

[5] 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.

5.3.4 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 5.4c), 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 (5.12).

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.

$$b_{t,z} = \begin{cases} \left(\sum_{z' \in Y} t_z(z') \cdot b_{t+1,z'} \right) \cdot e_z(x_{t+1}) & , 1 < t < T \\ t_z(f) & , t = T + 1,\ z = f. \end{cases} \tag{5.12}$$

5.3.5 Sparse Forward-Backward Algorithms

FIXME (Pal et al., 2006).

5.4 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[6]. 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).

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

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.

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[7]. Nevertheless, “dog” can be a verb, for example in the sentence “Fans may dog Muschamp, but one thing’s for certain: he did things the right way off the field.”. To avoid this kind of problems caused by data sparsity, both emission and transition counts need to be smoothed.

5.4.1 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 5.6: foo

Figure 5.6 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

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

the emission probability $e_{\texttt{NNP}}(\texttt{Dutch})$ in the Penn Treebank are simply:

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

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

Simple computation of co-occurrences is insufficient because of data-sparsity. Words do not occur with all POS tags in the training corpus and all 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 by Silfverberg and Lindén (2011) 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.

5.4.2 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).

5.5 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%.

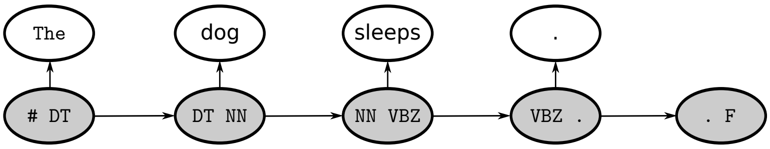


Figure 5.7: foo

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 5.7 illustrates a path with legal transitions.

Figure 5.7 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 $e_{t_i,t_{i-1}}(w_i) = p(w_i \,|\, t_i)$ instead of $e_{t_i,t_{i-1}}(w_i) = p(w_i \,|\, t_{i-1},\, t_i)$. 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 Silfverberg and Lindén (2011) 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.

5.6 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 6

Finite-State Machines

This thesis deals primarily with machine learning but Silfverberg and Lindén (2011) presents a weighted finite-state implementation of Hidden Markov Models. Therefore, a quick overview of finite-state calculus is required.

6.1 Weighted Finite-State Automata

Automata can be seen as a formalization of the concept of algorithm. They represent a restricted type of algorithm in the sense that they can only operate on symbol strings and they only operation they perform is to accept or reject a string. Although, this might sound quite restrictive, it turns out that most computational problems can be formalized as such *decision problems* involving only acceptance or rejection of symbol strings.

Every automaton is associated with a *formal language* which is a (possible infinite) set of finite strings from the set of all strings of some finite alphabet. The automaton solves the decision problem of this language, i.e. accepts exactly those strings that belong to the language. *Weighted formal languages* are an extension to the concept of formal language, where each string should be assigned a weight. If the weights can be interpreted as probabilities, a weighted formal language over an alphabet Σ can be seen as a probability distribution over the set of strings of the alphabet. However, this is not always possible because all weights cannot be interpreted as probabilities.

Several well know classes of algorithms exist. The most famous ones are Turing machines which, in a sense, subsume all existing automata **?**. In this thesis I have, however, utilized another class of automata, Finite-state machines. They are more restricted than Turing machines or another well know class, stack automata, because they utilize only a finite amount of memory. All real world implementations of automata of course utilize only a finite amount of memory.

The restriction on memory limits the set of problems that finite-state machines can solve. However, at the same time it allows for a rich algebra which allows one to combine finite-state machines to produce new finite-state machines.

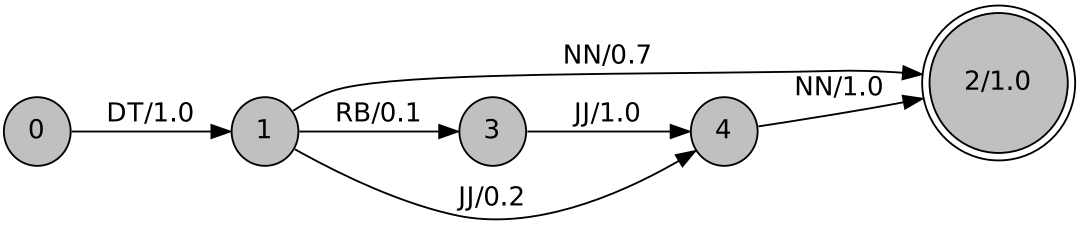


Figure 6.1: A finite-state machine accepting a subset of the singular noun phrases in Penn Treebank.

The machine in Figure 6.1 accepts sequences of Penn Treebank labels which correspond to singular noun phrases that can have an adjective attribute (JJ) with an optional adverb (RB) expressing degree.

Typically finite-state machines are represented as state transition graphs. An example is given in Figure 6.1. The machine has a set of numbered *states* (0, 1, 2, 3 and 4) and *transitions* from one state to another. The transitions are labeled using symbols from a finite alphabet (e.g. NN) and weights from a *weight semiring*, when the machine is weighted. The weight semiring in this example is the *probability semiring* Allauzen et al. (2007) where weights are probabilities that can be added and multiplied in the usual fashion.

Operation starts in the designated *start state* (0 in Figure 6.1) and it has to end in a *final state* (2 in Figure 6.1 marked with a double ring). A string is accepted, if and only if there is a sequence of transitions leading from the start state to a final state where the symbols in the transition labels spell out the input string and the product of the weights along the path is non-zero. For example the machine in Figure **??** accepts the string "DT NN" with weight 0.7 but rejects "DT JJ" because the string inds when the machine is in state 4 but that is not a finite state.

There may be several or no *accepting paths* for a given string. If there are several paths for some string, the machine is called *ambiguous*. More generally, a machine is called *non-deterministic* if some state has several out-going transitions with the same label. Trivially, ambiguous machines are non-deterministic. In the case of ambiguous machines, the weight of a string is the sum of the weights of all of its accepting paths or zero if there are no accepting paths.

Weighted formal languages Formally, weighted languages are mappings $M : \Sigma^* \to \mathbb{K}$ from the set of arbitrary strings Σ^* of some finite alphabet Σ into some *weight semiring* $(\mathbb{K}, \oplus, \otimes, \mathbb{0}, \mathbb{1})$ where $\mathbb{K}$ is a set of weights, $\oplus$ and $\otimes$ are addition and product operators for weights and $\mathbb{0} \in \mathbb{K}$ and $\mathbb{1} \in \mathbb{K}$ are the additive and multiplicative identity.

Weight semirings As mentioned above, the weight semiring in Figure 6.1 is the probability semiring, where weights belong to the set of non-negative reals $[0, \infty)$ and addition and product are given by the regular real addition and product operations. Even though probabilities reside in $[0, 1]$, the addition of probabilities can result in quantities that are larger than 1. Because finite-state algebra does include operations that add probabilities, the weight set needs to include all non-negative reals.

The general weight semiring $(\mathbb{K}, \oplus, \otimes, \mathbb{0}, \mathbb{1})$ is an abstraction of the probability semiring. The set $\mathbb{K}$ is an arbitrary set but the operations $\oplus \to \mathbb{K} \times \mathbb{K} \to \mathbb{K}$ and $\oplus \to \mathbb{K} \times \mathbb{K} \to \mathbb{K}$ need to satify the following requirements Allauzen et al. (2007)

1. $(\mathbb{K}, \oplus, \mathbb{0})$ is a commutative and associative monoid.

2. $(\mathbb{K}, \otimes, \mathbb{1})$ is an associative monoid.

3. $\otimes$ distributes with respect to $\oplus$.

4. $\mathbb{0}$ is the annihilator of $\otimes$.

If $\oplus$ is a group operation, the semiring is in fact a ring. Many useful weight semirings are, however, not rings. Often the semiring $(\mathbb{K}, \oplus, \otimes, \mathbb{0}, \mathbb{1})$ is denoted simply as $\mathbb{K}$. Paralleling the regular sum notation $\Sigma_{i \in I} x_i$ and product notation $\prod_{i \in I} x_i$, I use the notations $\bigoplus_{i \in I} x_i$ and $\bigotimes_{i \in I} x_i$, where I is a denumerable set.[1] Specifically $\oplus_{i \in \emptyset} x_i = \mathbb{0}$ and $\otimes_{i \in \emptyset} x_i = \mathbb{1}$.

Weights in the probability semiring and the addition and product operations have an intuitive interpretation but from a practical point of view the probability semiring is sub optimal. The biggest drawback is that repeated multiplication of probabilities gives rise to very small quantities and under-flow becomes a problem. Therefore, the so called *logarithmic semiring* is more commonly used. It can be derived by applying a logarithmic transformation $x \mapsto -\log(x)$ to the weights and operations in the probability semiring.

In the logarithmic semiring, numerical under-flow is not a problem but the addition operation given by $x \oplus y = -\log(\exp(-x) + \exp(-y))$ is resource intensive. Therefore, another semiring, the *tropical semiring*, is often used. The addition operation in the tropical semiring is given by $x \oplus y = \min(x, y)$. Computationally, this operation is light. A theoretical motivation is given by the fact that $\min$ preserves the magnitude of the sum although not its exact value. In practice, this is often sufficient **?**. All machines in this thesis use tropical weights.

Weighted finite-state automata Formally, a weighted finite-state automaton of weight semiring $\mathbb{K}$ is a structure $M = (\Sigma, Q, q_0, \rho, E)$ where

1. Σ is a finite symbol set (also called an alphabet).

2. Q is a finite set of states.

3. $q_0 \in Q$ is the unique start state.

4. $\rho : Q \to \mathbb{K}$ is the final weight map.

5. $T \subset Q \times (\Sigma \cup \{\varepsilon\}) \times Q \times \mathbb{K}$ is a finite set of transitions.

[1] Infinite sums and products are not defined in all semi rings, for example the probability semiring. In this case, it is often possible to augment the semirings with an infinity element ∞. This problem does not often cause problems in practice and will not be examined further.

Operation	Symbol	Definition
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	$M_1 \otimes M_2$	$\mathrm{w}_{M_1 \otimes M_2}(s) = \bigoplus_{s_1 s_2 = s} \mathrm{w}_{M_1}(s_1) \otimes \mathrm{w}_{M_2}(s_2)$
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	$\bigoplus_{n=0}^{\infty} M^n$	
Intersection	$M_1 \cap M_2$	$\mathrm{w}_{M_1 \cap M_2}(s) = \mathrm{w}_{M_1}(s) \otimes \mathrm{w}_{M_2}(s)$

Table 6.1: A selection of operators from the finite-state algebra.

The final weight map ρ associates each state $q \in Q$ with a final weight. The final states of M are $\{q \in Q \,|\rho(q) \neq \mathbb{0}\}$.

Each transition $x \in T$ consists of a source state $\mathrm{s}(x)$, an input symbol $\mathrm{i}(x)$, a target state $\mathrm{t}(x)$ and a weight $\mathrm{w}(x)$. The input symbol may be ε which denotes the empty symbol. For the single outgoing transition x in state 4 in Figure 6.1, the source state $\mathrm{s}(x) = 4$, the input symbol $\mathrm{i}(x) = $ "NN", the target state $\mathrm{t}(x) = 2$ and the weight $\mathrm{w}(x) = 1.0$.

A sequence of transitions $x_1, ..., x_n \in T$ is a *path*, if $\mathrm{s}(x_1) = q_0$ and $\mathrm{t}(x_{k-1}) = \mathrm{s}(x_k)$ for all k. As a special case, the empty sequence is also a path. The weight of path $p = (x_1, ..., x_n)$ is $\mathrm{w}_M(p) = (\prod_{i=1}^{n} \mathrm{w}(x_i)) \otimes \rho(x_n)$ and the weight of the empty sequence is $\rho(q_0)$. A path p is called successful when $\mathrm{w}_M(p) \neq \mathbb{0}$.

Each path $p = (x_1, ..., x_n)$ is associated with a unique, possibly empty, string $s \in \Sigma^*$ such that there is a subsequence $j_i, ..., j_m$ of $1, ..., n$ for which $i(x_{j_1}), ..., i(x_{j_m}) = s$ and $i(x_l) = \varepsilon$ whenever $l \notin \{j_1, ..., j_m\}$. If the path p is associated with the string s, it is called a *path of* s.

The set of paths of a string s is denoted P_s and the weight assigned to s by the automaton M is $\mathrm{w}_M(s) = \bigoplus_{p \in P_s} \mathrm{w}_M(p)$. When $P_s = \emptyset$, $\mathrm{w}_M(s) = \mathbb{0}$. The string s belongs to the weighted language $L(M)$ accepted by M if $L(M)(s) = \mathrm{w}_M(s) \neq \mathbb{0}$.

6.2 Finite-State Algebra

Finite-state machines use only a finite fixed amount of memory. Therefore, they cannot solve all computational problems. There is, for example, no finite-state machine which accepts only strings of prime number length. This can be proved using the pumping lemma of regular language (Sipser, 1996) which are the languages whose decision problems finite-state automata can solve.

Although the computational power of finite-state machines is limited, they are closed under a number of very useful operations which correspond to set theoretical operations of the languages recognized by the machines. The collection of these operations is called the *finite-state algebra*.

Table 6.1 gives an overview of a number of well known finite-state operations.

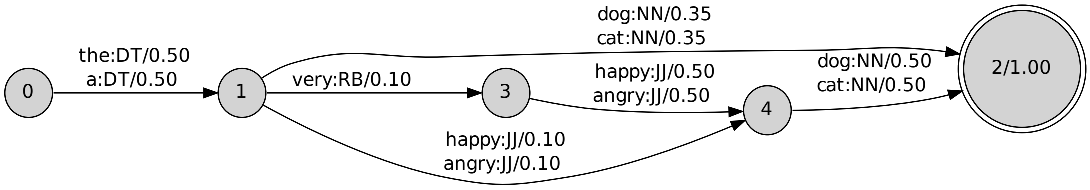


Figure 6.2: A finite-state transducer that analyzes a subset of the singular noun phrases in Penn Treebank.

6.3 Finite-State Transducers

Language processing frequently requires translation of one signal such as speech or text into another signal, for example text in another language in the case of machine translation or annotated text in the case of part of speech tagging. These do not seem like the kinds of processing tasks that finite-state machines can solve because, as we saw above, finite-state machines solve decision problems.

Weighted Finite-state transducers are an extension of weighted finite-state automata. Whereas finite-state automata simply accept or reject a symbol sequence, finite-state transducers simultaneously output another symbol sequence.

Formally, a weighted finite-state transducer of weight semiring $\mathbb{K}$ is a structure $M = (\Sigma, \Omega, Q, q_0, \rho, E)$ where

1. Σ is a finite input symbol set (also called an input alphabet).
2. Ω is a finite output symbol set (also called an output alphabet).
3. Q is a finite set of states.
4. $q_0 \in Q$ is the unique start state.
5. $\rho : Q \rightarrow \mathbb{K}$ is the final weight map.
6. $T \subset Q \times (\Sigma \cup \{\varepsilon\}) \times (\Omega \cup \{\varepsilon\}) \times Q \times \mathbb{K}$ is a finite set of transitions.

Transitions $e \in T$ are similar to transitions in automata but in addition to the input symbol $\mathrm{i}(e)$, they also contain an output symbol (e). Paths of transitions are defined in the same way as paths for automata, however, paths of transducer transitions are associated with two strings – an input and an output string. The input string of path $p = (x_1, ..., x_n)$ is the string $s_i = \mathrm{i}(x_1)...\mathrm{i}(x_n)$ disregarding epsilons and the output string is $s_o = (x_1)...(x_n)$ also disregarding epsilons. Path p itself is associated with the string pair $s = s_i{:}s_o$. The set of paths associated with the input string of s is $\mathrm{P}_i(s)$ and the set paths associated with of output string is $\mathrm{P}_o(s)$.

The weight of path p is defined in exaclty the same way as for automata

$$\mathrm{w}_M(p) = \Big(\prod_{i=1}^{n} \mathrm{w}(x_i)\Big) \otimes \rho(x_n).$$

A transducer M can be viewed as a machine that maps input strings to sets of output strings with weights. Equivalently, it can be seen as a machine accepting string pairs $s = s_i{:}s_o$ with some weight $\mathrm{w}_M(s)$. The weight $\mathrm{w}_M(s)$ given to string pair s by transducer M is defined

$$\mathrm{w}_M(s) = \bigoplus_{p \in \mathrm{P}_i(s) \cap \mathrm{P}_o(s)} \mathrm{w}_M(p)$$

The defition takes into account the fact that there may be several alignments of input and output string giving the exact same string pairs. For example, $(\varepsilon{:}b, a{:}b)$ and $(a{:}b, \varepsilon{:}b)$ represent the same string pair $a{:}bb$.

Chapter 7

Generative Taggers using Finite-State Transducers

This section presents a finite-state implementation of granarative taggers such as HMMs using finite-state algebra. This exapnds on Silfverberg and Lindén (2010) and Silfverberg and Lindén (2011).

7.1 Enriching the emission and transition models

- Halácsy et al. (2007) and Silfverberg and Lindén (2011).

7.2 Problems

In the standard first order HMM an observation depends only on the current hidden state. If the hidden state is given, the observation cannot be influenced by the hidden states at other positions in the input or the other observations. This facilitates estimation and makes the system resistant to over-fitting, but at the same time it severly limits the set phenomena tht the model can capture.

An example from POS tagging the Penn Treebank illustrates this problem. The Penn Treebank has two labels for common nouns: `NN` for singular nouns and `NNS` for plural nouns.

- Local normalization.

- Inability to use word context.

- Label bias.

- The inability to use rich features, causes problems in domains other than POS tagging and in POS tagging for MR languages.

- Smoothing is difficult.

- Differences in performance between generative HMMs and discriminative approaches are even greater for morphologically complex languages and small training sets.

- OOV words may require different mechanisms depending on language again causing problems for MR lanuages.

7.3 Finite-State Implementation of Hidden Markov Models

As seen in Chapter **?**, an HMM can be decomposed into a emission model, which models the conditional distribution $p(y|x)$ of all labels y given a state x and a transition model, which models the conditional distribution $p(y_{n+1}|y_1, ..., y_n)$ of states y_{n+1} given a state history $y_1, ..., y_n$.

Both the emission and transition models can be compiled into finite-state machines. The emission model is compiled into one finite-state transducer but the transition model is made up from a number of component models. All the models are combined using a run-time variant of composition and intersection.

7.3.1 Interpreting HMMs as Finite-State Machines

Given a sequence of observations $x = (x_1, ..., x_n)$ and states $y = (y_1, ..., y_n)$, the joined probability for x and y given by an HMM with parameters θ is

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

where ι, τ and ε are initial, transition and emission distributions respectively.

In an HMM the states y_i correspond to actual states of the model. When interpreting an HMM as a finite-state transducer, they will instead correspond to output symbols. The transducer will then map the input sequence x to the states sequence y with weight $p(x, y; \theta)$.

Mohri et al. (2002) describe an implementation of HMMs for speech recognition where each individual phoneme can be recognized by a small HMM. Their construction, however, requires $O(|Y|)^n$ states where Y is the set of states of the HMM and n its order. It is easy to see that this is infeasible. For example when using a second order model with a morphological label set of 1000 labels the resulting transducer has around one billion states.

7.3.2 The Emission Model

7.3.3 The Transition Model

7.3.4 Weighted Intersecting Composition

Chapter 8

Conditional Random Fields

8.1 Discriminative modeling

As seen in Chapter 5, the HMM POS tagger can be viewed as a state machine which alternates between sampling observations, that is words, from a state specific observation distributions, and sampling successor states, that is the morphological labels, from state specific transition distributions. Each emission and each successor state is conditioned *solely* on the current state. These independence assumptions are harsh. For example the Finnish cannot be adequately modeled, because the model does not include direct information about neighboring words in a sentence.

Although information about for example word collocations and orthography is quite useful in morphological labeling, it is often difficult to incorporate more elaborate contextual information in a generative model. As Sutton and McCallum (2012) note, there are two principal methods of doing this are improving the emissions model, which may be intractable because the model needs to account for complex dependencies between words in a sentence and their orthographic features, and replacing the usual emission model with for example a Naive Bayes' model.

In a domain closely related to morphological labeling, namely biomedical entity extraction,Ruokolainen and Silfverberg (2013) show that the Naive Bayes approach fails. In fact, adding richer context modeling such as adjacent words, worsens the performance of the modeling. One reason for this may be that overlapping information sources tend to cause the Naive Bayes model to give overly confident probability estimates (Sutton and McCallum, 2012). Combining them in a structured manner can therefore be problematic.

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

Discriminative models assign probabilities $p(y \mid x)$ for label sequences y and word sequences x by extracting local features from the input sentence and label sequence. Examples of local features include x_t *is "dog" and* y_t *is "NN"* and y_{t-1} *is "DT" and* y_t *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. Contrastingly, in discriminative models, there are no emissions or transitions. Instead, it is customary to speak about unstructured features relating exclusively 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. This is of course the famous bias-variance trade-off (Geman et al., 1992).

8.2 Maximum Entropy Modeling

8.2.1 Example

8.3 Basics

I will now 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. Whereas inference in CRFs is very similar to inference in HMMs, we did not discuss feature extraction in association to HMMs. This is because HMM taggers use a restricted set of features (the current word and preceding labels).

y	DT	NN	VBZ	.
x	The	dog	eats	.

Features are true logical propositions, which connect aspects of the input sentence with labels or parts of the label sequence. Given sentence x and label sequence y of length n, we extract features at every position t in the sentence. For example at position 2 in sentence x, we could extract *The current word* x_t *is "dog" and the label is "NN"* and *The previous label is "DT" and the current label is "NN"*. We could, however, not extract the same feature *The current word is "dog" and the label is "NN"* at position 1, because this proposition is false when $t = 1$ (the word at position 1 is "The" and the label is "VBZ").

Features are conjunctions of two parts: a feature template, for example *The current word is "dog"* and a label *the label is "NN"*. The set of features recognized by a CRF POS tagger contains all conjunctions $f\&y$ of feature templates f present in the training data and labels y. For example, the tagger would know *The current word is "dog" and the label is "DT"* although it is unlikely that this feature would ever be observed in actual training data.

Ratnaparkhi (1996) introduced a rather rudimentary feature set and variations of this feature set are commonly used in the literature (for example Collins (2002) and Lafferty et al. (2001)). 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 8.1 and the structured feature templates in Table 8.2.

Feature template	Example
The current word is w	The current word is “dog”
The current word has prefix p	The current word has prefix “d-”
The current word has suffix s	The current word has suffix “-og”
The current word contains a digit	
The current word contains a hyphen	
The current word contains a upper case letter	
The previous word is w	The previous word is “The”.
The next word is w	The next word is “eats”.
The word before the previous word is w	
The word after the next word is w	

Table 8.1: foo

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 8.2: foo

These feature templates are then combined with all labels occurring in the training set. It is instructive to try to estimate the number of features when using a realistic training set of size around a million words. The number of features is be $O(\max(|Y|^2,\ |W|) \cdot |Y|)$. For small label sets and large training data, the bulk of the feature set tends to consist 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 false and true truth values, but other activations in $\mathbb{R}$ can also be used, if the truth values of the feature propositions exist on a non-binary scale.

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 be an element in ϕ_t. In a concrete implementations, the index function I could 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.

Lafferty et al. (2001)

8.4 Logistic Regression

The simplest Conditional Random Field is the *Logistic Regression Model* (LRM). It is an unstructured probabilistic discriminative model. In this section, I will present a formal treatment of the LRM because it aids in understanding more general CRFs.

8.4.1 The Model

Regular linear regression models a *real valued quantity* y based on independent variables x_1, ..., x_n. In contrast the LRM is a regression model which models *the probability* that an observation x belongs to a class y in a finite class set Y. For example, the 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 and the degrees of expression of different genes in the tumor cells. These quantifiable information sources are the *feature templates* of the logistic classifier and combined with class labels they make up 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 focus on the binary valued case. When using binary features, we can equate the example x with the set of feature templates $F_x \subset F$ that it *activates*, that is *Tumor diameter* $\geq$ *5 cm*, *Preceded by “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 8.3, let F be a finite set of feature templates and Y a finite set of classes. Each combination of feature template $f \in F$ and class $y \in Y$ corresponds to a unique feature. Therefore, the model will have $|F \times Y|$ features in total. Let θ be a real valued parameter vector in $\mathbb{R}^{|F \times Y|}$ and let I be a 1-to-1 index function which maps each combination of feature template and class onto the indexes of θ, that is $1 \leq \mathrm{I}(f, y) \leq |F \times Y|$.

For each example x, let F_x be the set of feature templates that x activates and let $y \in Y$ be a class. Then the feature vector associated with x and y is $\phi(x, y) = \{0, 1\}^{|F \times Y|}$ 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 (8.1). The equation defines a probability distribution over the set of classes Y because each quantity $p(y \,|\, x; \theta)$ is a positive real and the quantities sum to 1.

$$p(y \,|\, x; \theta) = \frac{\exp(\theta^\top \phi(x, y))}{\sum_{z \in Y} \exp(\theta^\top \phi(x, z))} \tag{8.1}$$

Often a special bias term is used in Equation 8.1. It is

The Logistic Regression Model is log-linear as demonstrated by Equation 8.2, which represents the model using a cost function.

$$-\log p(y \,|\, x; \theta) = log(\mathrm{Z}(x; \theta)) - \theta^\top F_y(x), \text{ where } Z(x; \theta) = \sum_{z \in Y} \exp(\theta^\top F_z(x)) \tag{8.2}$$

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 the maximum likelihood, or equivalently minimum cost, estimation. The minimum cost estimate for the parameters θ using $\mathcal{D}$ is given by equation (8.3), where $\mathrm{Z}(\mathcal{D}; \theta) = \sum_{(x,\ y) \in \mathcal{D}} \mathrm{Z}(x; \theta)$ is the partition function of the entire data set $\mathcal{D}$.

$$\theta = \arg\min_{\theta'} \Big(\log(\mathrm{Z}(\mathcal{D};\theta)) - \sum_{(x,\,y)\in\mathcal{D}} {\theta'}^{\top} F_y(x) \Big) \tag{8.3}$$

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

1. The function $\theta \mapsto p(y \mid x; \theta)$ is smooth,
2. it is convex,
3. there exists a *unique* θ maximizing the likelihood of the training data $\mathcal{D}$, and
4. the model $p(y \mid x; \theta)$ 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 and Property 4 follows from the discussion in Section 8.2.

The log-linear form of the logistic classifier is motivated by the concept of maximum entropy, which was introduced in Subsection 8.2.1.

Although the maximization in Equation 8.1 cannot be solved for exactly in general, the convexity and smoothness of $p(y \mid x; \theta)$ mean that efficient numerical methods can be used for approximating the maximum to an arbitrary precision.

8.5 The Logistic Regression Model as a Classifier

Inference for a Logistic Regression Model means finding the probability of each class label $y \in Y$ given example x. The full computation of the probability is, however, not needed when the model is used as a classifier. For simply finding the class y_{max} which maximizes the conditional likelihood in equation 8.4 given fixed parameters θ it is sufficient to maximize the numerator of $p(y \mid x; \theta)$.

$$y_{max} = \arg\max_{y\in Y} p(y \mid x; \theta) = \arg\max_{y\in Y} \frac{\exp(\theta^{\top}\phi(x,y))}{\mathrm{Z}(x;\theta)} = \arg\max_{y\in Y} \exp(\theta^{\top}\phi(x,y)) \tag{8.4}$$

To avoid underflow when using finite precision real numbers (such as floating-point numbers), the maximization is usually rephrased as the minimization of a cost-function in Equation 8.5

$$y_{max} = \arg\min_{y\in Y} \theta^{\top}\phi(x,y) \tag{8.5}$$

From a practical implementation perspective, the minimization in Equation 8.5 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 $\mathrm{O}(|Y||F|)$.

8.6 Estimation

As seen in Section **??**, the Logistic Regression Model is smooth, that is the likelihood $p(y_1 \mid x_1; \theta) ... p(y_n \mid x_n; \theta)$ of a labeled training data set $\mathcal{D} = \{(x_1, y_1), ..., (x_n, y_n)\}$ is a smooth function of the parameter vector θ. This allows formulation of the parameter estimation task for the Logistic Regression Model as a optimization task whose solution can be approximated using numerical techniques.

8.7 CRF – A Structured Logistic Classifier

8.8 Note on Terminology

- The term perceptron tagger is commonly found in the litterature, e.g. Collins (2002).
- The model is called a CRF.
- The estimator can be a ML, Perceptron, pseudo likelihood, pseudo perceptron...

8.9 Inference

- Inference using Viterbi.

8.10 Estimation

- Iterative process: label data, adjust parameters and repeat.
- Use SGD or L-BFGS (Vishwanathan et al., 2006) for ML estimation.
- Use averaged perceptron (Collins, 2002).
- Dev data for adjusting hyper parameters: number of iterations and regularization hyper-parameters, model order.
- Problems with exact estimation.
- Layered CRF Mueller et al. (2013), Weiss and Taskar (2010) and Charniak and Johnson (2005).

8.11 Approximate Estimation

- Beam search.
- Violation fixing.
- Label guessing.

8.12 CRF taggers and Morphological Analyzers

8.13 Enriching the Structured Model

- Utilizing sub-label structure.

Chapter 9

Lemmatization

9.1 Introduction

9.2 Data-Driven Lemmatization

- Classifier based: Grzegorz Chrupala and van Genabith (2008).
- Finite-state based: Lindén (2009).
- Combination: Hulden et al. (2014).

9.3 Combining Morphological Analyzers and Data-Driven Lemmatization

Chapter 10

Experiments

This section resents experiments on morphological tagging using the FinnPos tagger toolkit developed by the authors of Silfverberg et al. (2015).

10.1 Using a Cascaded Model

Adaptive Guess Count

Guess Mass	Tagging Accuracy (%)	Training time (min)	Tagging Speed (word/s)
0.9	xx.xx	xx	xx
0.99	xx.xx	xx	xx
0.999	xx.xx	xx	xx
0.9999	xx.xx	xx	xx

Fixed Guess Count

Guess Count	Tagging Accuracy (%)	Training time (min)	Tagging Speed (word/s)
1	xx.xx	xx	xx
5	xx.xx	xx	xx
10	xx.xx	xx	xx
15	xx.xx	xx	xx
20	xx.xx	xx	xx

Table 10.1: Different label guesser settings for FinnTreeBank

Adaptive Guess Count

Guess Mass	Tagging Accuracy (%)	Training time (min)	Tagging Speed (word/s)
0.9	xx.xx	xx	xx
0.99	xx.xx	xx	xx
0.999	xx.xx	xx	xx
0.9999	xx.xx	xx	xx

Fixed Guess Count

Guess Count	Tagging Accuracy (%)	Training time (min)	Tagging Speed (word/s)
1	xx.xx	xx	xx
5	xx.xx	xx	xx
10	xx.xx	xx	xx
15	xx.xx	xx	xx
20	xx.xx	xx	xx

Table 10.2: Different label guesser settings for Turku Dependency Treebank

Adaptive Beam

Beam Width	Tagging Accuracy (%)	Training time (min)	Tagging Speed (word/s)
0.9	xx.xx	xx	xx
0.99	xx.xx	xx	xx
0.999	xx.xx	xx	xx
0.9999	xx.xx	xx	xx

Fixed Beam

Beam Width	Tagging Accuracy (%)	Training time (min)	Tagging Speed (word/s)
1	xx.xx	xx	xx
5	xx.xx	xx	xx
10	xx.xx	xx	xx
15	xx.xx	xx	xx
20	xx.xx	xx	xx

Table 10.3: Different beam settings for FinnTreeBank

10.2 Beam Search

10.3 Model Order

10.4 Using A Morphological Analyzer

10.5 Utilizing Sub-Label Dependencies

10.6 Pruning the Model

Adaptive Beam			
Beam Width	Tagging Accuracy (%)	Training time (min)	Tagging Speed (word/s)
0.9	xx.xx	xx	xx
0.99	xx.xx	xx	xx
0.999	xx.xx	xx	xx
0.9999	xx.xx	xx	xx

Fixed Beam			
Beam Width	Tagging Accuracy (%)	Training time (min)	Tagging Speed (word/s)
1	xx.xx	xx	xx
5	xx.xx	xx	xx
10	xx.xx	xx	xx
15	xx.xx	xx	xx
20	xx.xx	xx	xx

Table 10.4: Different beam settings for Turku Dependency Treebank

Model Order	Tagging Accuracy (%)	Training time (min)	Tagging Speed (word/s)
0	xx.xx	xx	xx
1	xx.xx	xx	xx
2	xx.xx	xx	xx

Table 10.5: Different Model Orders for FinnTreeBank

Model Order	Tagging Accuracy (%)	Training time (min)	Tagging Speed (word/s)
0	xx.xx	xx	xx
1	xx.xx	xx	xx
2	xx.xx	xx	xx

Table 10.6: Different Model Orders for Turku Dependency Treebank

Without a Morphological Analyzer

Sub-Label Order	Tagging Accuracy (%)	Training time	Tagging Speed (KTokens/s)
None	92.61 (OOV: 75.25)	3 min, 5 epochs	6
0	93.05 (OOV: 77.35)	1 min, 2 epochs	5
1	93.32 (OOV: 78.58)	1 min, 2 epochs	4

Using a Morphological Analyzer

Sub-Label Order	Tagging Accuracy (%)	Training time	Tagging Speed (KTokens/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

Table 10.7: Different Sub-Label Orders for FinnTreeBank

Without a Morphological Analyzer

Sub-Label Order	Tagging Accuracy (%)	Training time	Tagging Speed (KTokens/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

Using a Morphological Analyzer

Sub-Label Order	Tagging Accuracy (%)	Training time	Tagging Speed (KTokens/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

Table 10.8: Different Sub-Label Orders for Turku Dependency Treebank

	Pruning Strategy			
	None	Zero Param	<5 updates	<7 updates
Sub-Label Order				
None	xx.x%, yy MB	xx.x%, yy MB	xx.x%, yy MB	xx.x%, yy MB
0	xx.x%, yy MB	xx.x%, yy MB	xx.x%, yy MB	xx.x%, yy MB
1	xx.x%, yy MB	xx.x%, yy MB	xx.x%, yy MB	xx.x%, yy MB

Table 10.9: Result of applying different pruning strategies on FinnTreeBank models.

	Pruning Strategy			
	None	Zero Param	<5 updates	<7 updates
Sub-Label Order				
None	xx.x%, yy MB	xx.x%, yy MB	xx.x%, yy MB	xx.x%, yy MB
0	xx.x%, yy MB	xx.x%, yy MB	xx.x%, yy MB	xx.x%, yy MB
1	xx.x%, yy MB	xx.x%, yy MB	xx.x%, yy MB	xx.x%, yy MB

Table 10.10: Result of applying different pruning strategies on Turku Dependency Treebank models.

Chapter 11

Conclusions

References

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

Bilmes, J. (1997). A Gentle Tutorial on the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models.

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.

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.

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.

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.

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

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

Grzegorz Chrupala, G. D. 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/.

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.

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.

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.

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.

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.

Mueller, T., Schmid, H., and Schütze, H. (2013). Efficient higher-order CRFs for morphological tagging. Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 322–332. Association for Computational Linguistics.

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 InternationalConference on Intelligent Text Processing and Computational Linguistics CICLing 2014*.

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. (1998). *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, Philadelphia, PA, USA. AAI9840230.

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.

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, *Advances in Natural Language Processing*, volume 6233 of *Lecture Notes in Computer Science*, 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: Pos tagging toolkit for morphologically rich languages. *Language Resources and Evaluation*, VOL(NUM):XX–YY.

Sipser, M. (1996). *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition.

Sutton, C. and McCallum, A. (2012). An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4):267–373.

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.

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.

Contributions

Articles

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 $t = (t_1 \ ... \ t_n)$ given the sentence $s = (s_1 \ ... \ s_n)$ 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 $t = t_1 \ ... \ t_{2k}$ and a sentence $s = s_1 \ ... \ s_{2k}$ 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 $t = t_1 \ ... \ t_{2k+1}$ and a sentence $s = s_1 \ ... \ s_{2k+1}$ 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 $s = s_1 \ldots s_n$, we form a WFST W_i which is a disjunction of its five best analyzes $t_1 \ldots t_5$ 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

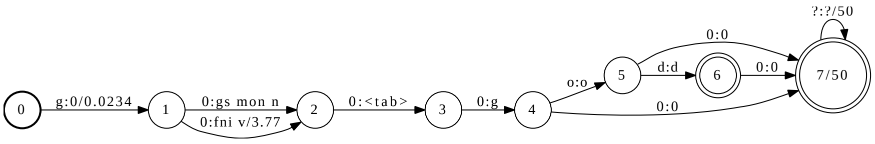


Fig. 1. Guesser constructed from a weighted lexicon. Guessing starts at the end of a word. Skipping letters gives a high penalty and analyzes, where equally many letters are skipped, are weighted according to the frequency of the analyzes.

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.

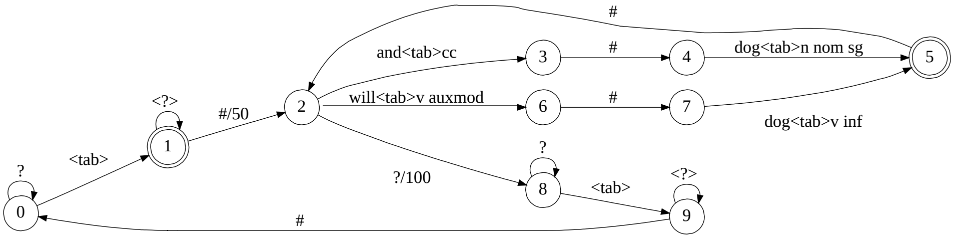


Fig. 2. A small example of an even bigram model B_e. ? signifies an arbitrary symbol and <?> signifies an arbitrary POS analysis symbol.

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.

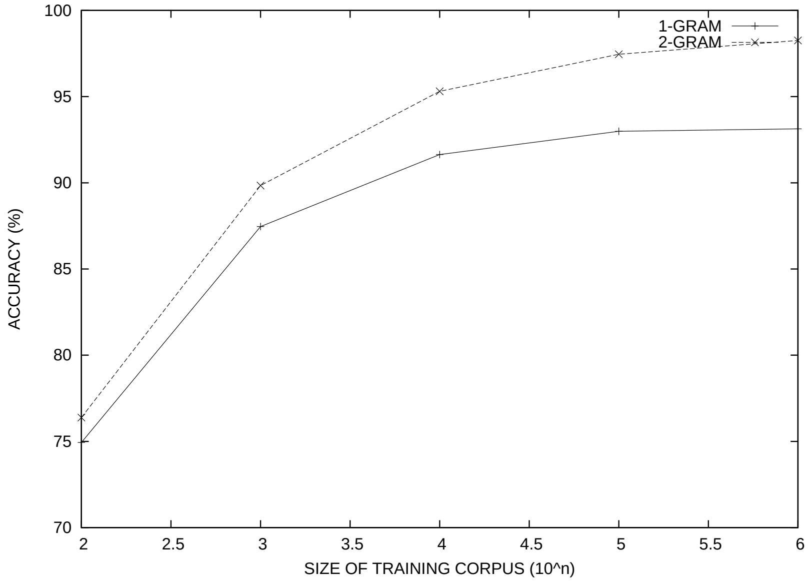


Fig. 3. The accuracy for the English POS tagger as a function of the size of training data. We used between 10^2 and 10^6 sentences for training. The lower curve displays the accuracy using only the unigram model, whilst the upper curve displays the accuracy of the combined unigram and bigram model.

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

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	English	Finnish
Algorithm		
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 word-form 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 $x = (x_1, x_2, \ldots, x_{|x|})$ 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.

$y = (y_1, y_2, \ldots, y_{|x|})$, 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.

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.

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.

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 $x = (x_1, \ldots, x_{|x|})$ and a POS sequence $y = (y_1, \ldots, y_{|x|})$ is defined as

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

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 $i \in 1 \ldots |x|$.

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.