An Approach to the Morphological Disambiguation Problem Using Conditional Random Fields

by

Bülent Burak Arslan

Submitted to the Graduate School of Sabancı University
in partial fulfillment of the requirements for the degree of
Master of Science

Sabancı University

August, 2009

An Approach to the Morphological Disambiguation Problem using Conditional Random Fields

APPROVED BY

Prof. Dr. Kemal Oflazer (Thesis Supervisor)	.....
Assoc. Prof. Dr. Berrin Yanıkoğlu	.....
Dr. Dilek Hakkani-Tür	.....
Assist. Prof. Dr. Hakan Erdoğan	.....
Assist. Prof. Dr. Murat Saraçlar	.....

DATE OF APPROVAL: .....

© 2009, Bülent Burak Arslan
All Rights Reserved

To the perseverant...

An Approach To The Morphological Disambiguation Prolem Using Conditional Random Fields

B. Burak Arslan

FENS, M.S. Thesis, 2009

Thesis Supervisor: Kemal Oflazer

Abstract

Morphology is the subfield of linguistics that studies the internal structures of words. *Morphological analysis* is the first step in revealing this structure by enumerating possible underlying morphological unit combinations that describe the surface form of a given word. The given surface form is said to be *morphologically ambiguous*, when more than one analysis corresponds to the given surface form.

While words in every natural language may manifest morphological ambiguity, solving the problem of morphological disambiguation presents different challenges for different languages.

In this work, we present an approach to this problem using Conditional Random Fields, a statistical framework that elegantly avoids data sparseness problems arising from the large vocabulary and tag set sizes, a characteristic of Turkish language. CRFs are used to build statistical models that rely on simple functions of easily testable properties of the training data at hand. Thanks to higher expressiveness gained by using tests on individual morphological markers, our results are in line with the state-of-the-art, using only a simple one-dimensional bigram chain model.

Biçimbirimsel Denkleştirme Sorununa Koşullu Rassal Alan Algoritması ile Bir Çözüm Önerisi

B. Burak Arslan
MDBF, Lisansüstü Tezi, 2009
Tez Danışmanı: Kemal Oflazer

Özet

Biçimbilim, dilbilimin kelimelerin dahili yapısıyla ilgilenen alt dalıdır. *Biçimbirimsel denkleştirme* işlemi ise, bir kelimenin biçimbirimsel yapısını belirleme işinin, eldeki kelimeye karşılık gelen biçimbirim kombinasyonlarının listelenmesi işleminin yapıldığı ilk adımıdır. Eğer eldeki kelimeye karşılık gelen böyle birden fazla biçimbirimsel analiz varsa, kelimenin *biçimbirimsel belirsizliğinden* söz edebiliriz.

Bütün dillerin kelimelerinde biçimbirimsel belirsizlik gözlemlenebilmesine rağmen, bu sorun farklı dillerde farklı zorluklar barındırmaktadır.

Bu çalışmada, koşullu rassal alan algoritması ile türkçe kelimelerin biçimbirimsel belirsizliğinin kaldırılması sorununa bir çözüm önerisi tanıtılmaktadır. Koşullu rassal alan algoritması, türkçenin kelime ve etiket kümelerinin boyutundan kaynaklanan veri seyrekliği sorunlarını, yapısal özelliklerinden dolayı geçersiz kılan bir istatistiksel analiz yöntemidir. Bu yöntem ile, eldeki verinin kolayca doğrulanabilen özelliklerini kullanan basit fonksiyonlara dayanan istatistiksel modeller elde edilmektedir. Biçimbirimlerin tek tek doğrulanabilmesiyle kazanılan yüksek ifade gücü sayesinde, basit ikili bir model kullanarak aldığımız sonuçlar en modern çalışmalarla aynı seviyededir.

Acknowledgments

If this work was at all possible, it's thanks to my family. They believed in me and put their infinite support behind me.

I would like to express my gratitude to Kemal Oflazer. His tolerance, as well as his wisdom, knowledge and guidance were invaluable.

I'd like to thank Taku Kudo, who was kind enough to offer key insights about his work. I'd also like to thank Zoubin Ghahramani, and his students Iain Murray and Simon Lacoste-Julien who have offered valuable advice.

Lastly, I'd like to express my love to my labmates. Working inside a considerate and naturally cheerful bunch who's always willing to endure my ramblings, that's what I call good fortune! Thanks guys!

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Outline 3

2 Morphological Disambiguation **5**

2.1 Preliminaries 5

2.2 Previous Work 8

3 The Conditional Random Field Framework **12**

3.1 Inference 16

3.2 Parameter Estimation 18

4 Implementation **22**

4.1 Representing Input Turkish Words 22

4.1.1 The Inflectional Group 23

4.1.2 The Ambiguity Tree 27

4.2 Morphological Feature Templates 31

4.3 A Sample Run 35

4.4 Factors that Influence Resource Usage 37

4.4.1 The Training Part 37

4.4.2 The Testing Part 38

4.5 Results 38
4.5.1 Unigram Tests 39
4.5.2 Bigram Tests 39
4.5.3 Other Combinations 39
4.5.4 Error Analysis 41

5 Conclusions and Future Work **43**

List of Tables

2.1 List of possible analyses of the words of the sentence 2.2. . . . 8

4.1 Analyses of the word *yıllarıydı* 24
4.2 Random variables and the markers they may emit. 26
4.3 Tags for the word *için*. 27
4.4 List of possible analyses of the words of the sentence 4.1. . . . 35
4.5 The generated features for lattice in Figure 4.2 36
4.6 Unigram test summaries on 10 random data sets. 39
4.7 Bigram test summaries on 10 random data sets. 40
4.8 Miscellaneous test summaries on 10 random data sets. 41

List of Figures

2.1 The disambiguation lattice for sentence 2.2 7

3.1 The topology of the chain CRF used in this work, very similar to traditional HMM. 14

4.1 Analysis tree of the word *için* 29

4.2 The disambiguation lattice for sentence 4.1 35

Chapter 1

Introduction

1.1 Motivation

The discipline of natural language processing gathers under its umbrella computational methods that have their roots in a large spectrum of disciplines in order to attain one single aim: To make sense of the samples of the human natural language – the essential tool for any kind of social interaction.

While making sense of natural language is acknowledged as one of the most difficult problems in computer science, the motivation for building intelligent systems that can communicate naturally with humans has only increased. This is in part due to the advent of the Internet that made vast amounts of digitized text instantly and freely accessible at a very small cost. Nowadays, an important part of human knowledge, one way or the other, is stored, edited, managed and published with the help of computers.

Processing natural language is a daunting prospect, mainly because every natural language utterance has ambiguities that prevent it from being easily processed.

The aim of this work is to present an attempt at solving a specific problem

in natural language processing: *Morphological disambiguation.* While it is a common problem to every natural language, solving it for different language families present different challenges.

While the methods presented here are known to be applied to other natural languages, the implementation of those methods that was used during the course of this project are highly Turkish specific, with an integrated pipeline of three components:

1. **The Morphological analyzer**, which is an implementation of a finite-state approach to Turkish morphology using the two level formalism [Oflazer, 1994, Oflazer et al., 1999]. It parses the word surface and gives out candidate parses. There may be more than one parse that describes a given surface form, hence the need for disambiguation.

2. **Ambiguity tree generator**, which is an implementation of a novel algorithm for listing minimally distinguishing markers of each morphological analysis. This method renders the problem to a one-dimensional sequential classification problem.

3. **Disambiguator**, which is an implementation of the conditional random field framework – a probabilistic method that unifies best of Markovian and maximum entropy models. It has two distinct components:

   (a) **The training component**, which computes the gradient for a L2-Regularized conditional random field in a one-dimensional chain topology, and runs L-BFGS optimization iteratively, until some termination criteria is satisfied.

   (b) **The testing component**, which computes every probability for every possible path in the CRF chain, and applies Viterbi algorithm in order to obtain the most probable path.

The work presented here can be used as a component in many language processing tasks such as parsing, machine translation, text-to-speech, document classification, etc.

1.2 Outline

The rest of this thesis is organized as follows: In Chapter 2, we present the morphological disambiguation problem in a more detailed yet intuitive way along with a review of previous work.

In Chapter 3, we formally define the problem from a mathematical standpoint, and frame it in the context of the conditional random field framework. We then explain the conditional random field framework in detail and present where methods for parameter estimation and inference used in this work.

In Chapter 4, we apply the model to morphological disambiguation of Turkish and present results from our experiment.

In Chapter 5, we present a summary and conclusions along with a discussion of future work.

A note about the terminology...

The literature of Turkish morphology and conditional random fields are not compatible in some cases, so a few small points merit emphasis:

- The term *morphological marker* is used to refer to the components that form the output of our morphological analyzer, listed in Table 4.2.
- The term *feature* is never used to refer to morphological markers. They are instead used to refer to boolean functions that are the basic components of a language model that uses the conditional random field

framework.

- Similarly, the term *morphological feature* is used to refer to boolean functions that test a specific morphological phenomenon in a given analysis to a given word.

- The terms *state* or *tag* are used to refer to the possible analyses of a given word, distinguished by a minimal morphological representation.

Chapter 2

Morphological Disambiguation

2.1 Preliminaries

Morphological disambiguation is the task of determining the correct morpheme structure of a word, given its context. Candidate morphological structures of a word are obtained via *morphological analysis*. It is assumed that, if there is more than one morpheme structure that describes the given surface form, the speaker meant to use only one of them.

While morphological ambiguity is a problem common to every natural language, performing morphological disambiguation presents varying levels of difficulty for various languages.

Consider the sentence:

He can can the can. (2.1)

In this example, the three occurences of the word "can" has three different parts-of-speech:

He	can	can	the	can
	Modal	Infinitive		Noun

It is possible to perform morphological disambiguation on indo-european languages like English, French etc. by using a small number of predefined tags (mostly part-of-speech tags).

However, determining just the part-of-speech tags is not enough for languages like Turkish, that have a complex word structure. Consider the following example that illustrates the morphological disambiguation problem in Turkish:

Gelemediği için çok üzgündü. (2.2)

As can be seen in Figure 2.1, in this sentence, specifying just the part of speech of a word does not help us to fully disambiguate it. Words in this sentence are ambiguous in root, possesive and miscellaneous markers, as well as part-of-speech markers. The tags that distinguish the morphological analyses are highly word specific, thus can't be defined prior to modeling. This is not the case for English, simply because its simple morphotactics can't encode this much information in a single word. So, solving this problem for Turkish, or Turkic languages in general, present different challenges, when compared to other language families.

The meanings of the states in the lattice are detailed in Table 2.1

The state markers shown in Figure 2.1 are obtained from the output of the morphological analyzer. The details of this process are given in section 4.1.2

The correct analysis of a given word can only be determined in its *context* – it is not possible to make this decision just by looking at the word itself. However, there is not much of a consensus among linguists about what con-

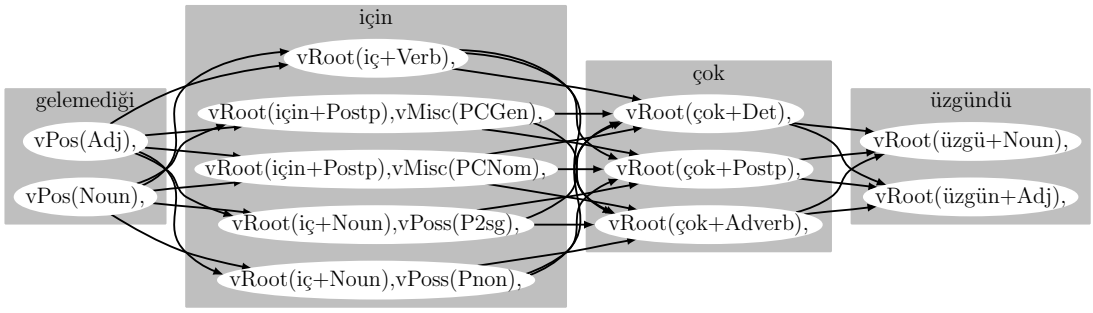

Figure 2.1: The disambiguation lattice for sentence 2.2

stitutes a correct analysis, nor which words can be considered as the context of a given word.

The simplest possible approximation to the context of a word is limiting it to the word before the given word, without going beyond the sentence boundaries. Such a model is called 2-gram (bigram) language model. As this model is both simple and requires a manageable amount of data to be trained optimally, it is a very good starting point.

While a simple approximation can yield promising results for the context determination problem, no simple approximation exists when it comes to defining what a correct morpheme sequence is, given its context. As humans, we *feel* that there just seems to be such a concept of correctness which is defined intuitively and sometimes subconciously.

Two approaches exist for modeling this black box behaviour:

1. **Rule-based morphological disambiguation**, which seeks to approximately define what a correct morpheme structure is via the use of hand-crafted constraints.

2. **Statistical morphological disambiguation**, which uses probability models to select the most likely interpretation of a word in a given context.

Root	Inflectional groups	Sample Usage
	gelemediği	
gel	+Verb ^DB+Verb+Able+Neg ^DB+Adj+PastPart+P3sg	**Gelemediği** yer burası.
gel	+Verb ^DB+Verb+Able+Neg ^DB+Noun+PastPart+A3sg +P3sg+Nom	**Gelemediği** için çok üzgündü.
	için	
iç	+Noun+A3sg+P2sg+Nom	Senin **için** kararmış
iç	+Noun+A3sg+Pnon+Gen	**İçin** görünümü dıştan iyi.
iç	+Verb+Pos+Imp+A2pl	Bunu **için** lütfen.
için	+Postp+PCGen	Benim **için** bunu yap.
için	+Postp+PCNom	Gelemediği **için** çok üzgündü
	çok	
çok	+Adverb	Gelemediği için **çok** üzgündü.
çok	+Det	Burada **çok** at var.
çok	+Postp+PCAbl	Senden daha **çok** yedim.
	üzgündü	
üzgü	+Noun+A3sg+P2sg+Nom ^DB+Verb+Zero+Past+A3sg	Bu şehir senin **üzgündü**. (– eziyetindi)
üzgün	+Adj ^DB+Verb+Zero+Past+A3sg	Gelemediği için çok **üzgündü**.

Table 2.1: List of possible analyses of the words of the sentence 2.2.

2.2 Previous Work

Early work on any sort of disambiguation task on natural languages adopted the rule-based approach. While initial attempts gave promising results, it was realized that the effort required to construct hand-crafted rules that would cover all uses of language was monumental. What's worse, natural languages being in constant evolution meant that this goal was a moving target – impossible to reach.

Especially in the context of Turkish and Turkic languages, performing

morphological disambiguation on input text promises to prune a considerable number of candidates. Turkish words have been actively studied since the seminal work from Oflazer [1994] that presented a robust method of representing the morphological phenomena observed in the Turkish word, as well as an efficient algorithm in order to obtain this representation.

One of the earlier works that attacked the morphological disambiguation problem [Oflazer and Kuruöz, 1994] adopted a rule based approach, reporting up to 98% accuracy within the (reportedly) rather limited test set. Judging by the numbers, the initial results seemed certainly promising.

In a successor work by Oflazer and Tur [1996], the authors observed that manually writing rules was "a serious impediment to the generality and the improvement of the system". So they designed and presented a scheme to extract constraints in an unsupervised way from Turkish corpora.

As the number of rules increased both in volume and complexity, organizing them in order to prevent contradictions and issues arising from application order became increasingly more difficult. The successor work from Oflazer and Tür [1997] sought to fix this problem by delegating the work of arbitration between constraints to a statistical (voting) algorithm. While this method seems quite similar to how the modern maximum entropy approach works, the voting weights of constraints were semi-automatically assigned by using a separate, much simpler hand-crafted rule set, differing substantially from the fully automated optimization techniques used by maximum entropy schemes.

The first work that used mostly statistical methods from Hakkani-Tür et al. [2000] used n-gram language models for disambiguation. While there was a preprocessor that eliminated some of the "obviously" wrong parses by applying constraints from a relatively simple hand-crafted rule set, the major

chunk of disambiguation work was done by the stochastic pipeline.

While this work was a huge step forward in terms of the volume of processed data, the rich expresiveness of the hand-crafted rule sets was lost, because the statistical approach employed in this work can only use equality tests both on the input words and output tags. That's why the depicted methods suffered from serious data-sparseness problems, due to the high number of morphological tags and surface forms, the characteristic challenge presented by languages like Turkish, which rely on their rich productive morphology mechanisms to sustain expresiveness.

With works from Yuret and Türe [2006] and Sak et al. [2007] the expresiveness gap was overcome. The former work uses the Greedy Prepend Algoritm to obtain a decision tree for every possible morphological marker, while the latter uses the averaged perceptron algorithm, very similar to the conditional random field algorithm in the sense that both seek to model the data at hand using a weighted combination of simple boolean tests, or combinations of those tests where the said weights are obtained by optimization techniques that approximate the maximum likelihood estimator of the training data.

Disambiguation tasks for other languages than Turkish solve problems with very similar settings. Kudo et al. [2004] present a framework that attempts to solve the morphological disambiguation problem for Japanese text using the conditional random field at its heart. The resulting implementation was kindly made open-source by the original authors of which the software written to conduct the work presented in this thesis is a derivative[1].

The conditional random field framework has also been reported to give promising results for similar tasks in to other languages. These include part-

[1]Accessible at `http://crfpp.sf.net`

of-speech tagging [Lafferty et al., 2001], shallow parsing [Sha and Pereira, 2003] and named entity recognition [Sarawagi and Cohen, 2004].

Chapter 3

The Conditional Random Field Framework

In this Chapter, we present an overview of the conditional random field framework.[1]

For the rest of this document, The following conventions will be used unless otherwise noted.

- Let $n(x)$ be the cardinality function for input set or vector x.
- Let $M = \{m_1, \dots\}$ the set of morphological markers. For Turkish, this set is assumed to be infinite, in general.
- Let $W = \{w_1, \dots\}$ the set of Turkish words. This set is assumed to be infinite.
- Let $T = \{t_1, \dots\}$ the set of tags[2]. $T = \bigcup_w T_w$ where T_w is the set of possible tags for word w. This set is assumed to be infinite. It is

[1] We assume familiarity with basic concepts of Hidden Markov Models.

[2] Sequences of morphological markers, reduced by the ambiguity tree algorithm

also assumed that tags T_w can be predetermined by the morphological analyzer for every possible input word.

- Let $\mathbf{c}$ be the sentence training corpus which contains sequences of (w, y_w) pairs and sentence delimiters. We assume there are $n(\mathbf{c})$ sentences in $\mathbf{c}$.

- Let X be a discrete random variable that emits a Turkish word, $x \in W$. Let $\mathbf{x}_k$, emitted by random process $\mathbf{X}_k$ be a sequence of words, or the k^{th} sentence of a corpus $\mathbf{c}$.

- Let $Y_i(x_i)$ be a discrete random variable that emits a tag $y_{i,j} \in T_{x_i}$ associated with word x_i. $y_{i,j}$ is said to be the j^{th} possible tag for the i^{th} word in the sentence. $\mathbf{y}_k$, emitted by random process $\mathbf{Y}_k(\mathbf{x})$, is a tag sequence for the k^{th} sentence in a corpus $\mathbf{c}$. Also, for simplicity of the notation, we let y_x denote *one of the tags* for word x

The set of morphological markers also include word roots. While both marker and root groups are pre-defined, the morphological analyzer tries to analyze words with unknown roots as well, so we can't conclude that the set of morphological markers is finite, nor pre-defined.

The set of tags is also infinite only because unknown word roots are processed. However, none of its elements are enumerated in the scope of this work. They are instead dynamically obtained from observed words. They are word-specific and represent a marker sequence for one of the possible analyses of a given word. If the given word is not ambiguous, it is said to have the `NoAmb` tag that stands for *no ambiguity*.

It is observed that more than 50.000 tags exist, but only a small amount (rarely above 10) is applicable to a given word, as other possibilities are ruled out by the morphological analyzer.

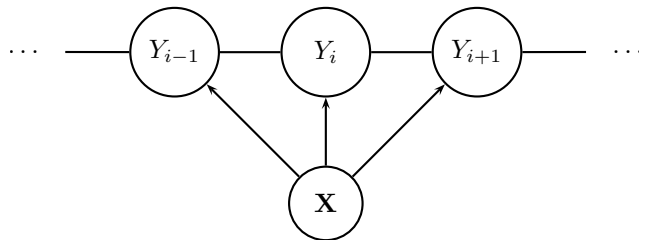

Figure 3.1: The topology of the chain CRF used in this work, very similar to traditional HMM.

We seek to compute the probabilities for every possible tag $y_{i,j}$ of observation x_i, given its context and the training data $\mathbf{c}$.

Formally, that is:

$$P(y_i|y_{j<i}, \mathbf{x}, \mathbf{c}) \tag{3.1}$$

As ours is a bigram model, this reduces to:

$$P(y_i|y_{i-1}, \mathbf{x}, \mathbf{c}) \tag{3.2}$$

where P is a probability distribution function that has its paramaters optimized to maximize the likelihood of the training data $\mathbf{c}$.

When a Markov random field is also globally conditioned on the observed symbols, it is called a *Conditional Random Field* [Lafferty et al., 2001, Sutton and Mccallum, 2006].

The conditional random field framework can model arbitrarily-shaped graph topologies[Bakir et al., 2007]. As the morphological disambiguation problem can be defined as a sequential classification problem, the 1-D chain template that can be seen in Figure 3.1 is judged to be sufficient.

The conditional random field framework aims to make use of a number

of simple tests, in order to explain complex phenomena. The said tests are either used alone or combined with each other, in order to obtain real-valued functions that somehow convert the observations into numbers. In our case, we use boolean features that return either `true` or `false`, testing whether a given word, or a given analysis of a given word has a property defined in the test itself.

The conditional random field framework offers no theoretical guidance on the method of generating features. This is one of the notable strengths of the conditional random field framework where the statistical modeler is free to use any number and types of features the resources at hand allow. However, we observed that models with relatively less number of features are more successful on the same task. So, even with very liberal resources, trying to keep the number of features at a minimum improves model performance.

Let f be a feature, and $\mathbf{f}$ be the vector of generated features. Every feature is assigned a coefficient, denoted λ. The λ coefficients are stored in a vector denoted Λ.

Features are boolean functions that have the following general signature:

$$f(y_i, y_{i-1}, \mathbf{x}, i) \tag{3.3}$$

The following is the probability distribution function of the conditional random field:

$$P(y_i|y_{i-1}, \mathbf{x}, \Lambda) = \frac{1}{Z_\Lambda(\mathbf{x})} \exp\left(\sum_a \lambda_a f_a(y_i, y_{i-1}, \mathbf{x}, i)\right) \tag{3.4}$$

where a ranges over all distinct feature functions, and $Z_\Lambda(\mathbf{x})$ is a normalization constant over all candidate paths:

$$Z_\Lambda(\mathbf{x}) = \sum_{\mathbf{y}} \exp\left(\sum_a \lambda_a f_a(y, y', \mathbf{x}, i)\right) \tag{3.5}$$

where $\mathbf{y}$ ranges over all tag pairs (y, y'). When the likelihood of a label sequence is considered, we have:

$$P(\mathbf{y}|\mathbf{x}, \Lambda) = \prod_{i=1}^{n(\mathbf{x})} P(y_i|y_{i-1}, \mathbf{x}, \Lambda) \tag{3.6}$$

which is equal to:

$$P(\mathbf{y}|\mathbf{x}, \Lambda) = \frac{1}{Z_\Lambda(\mathbf{x})} \exp\left(\sum_{i=1}^{n(\mathbf{x})} \sum_a \lambda_a f_a(y_i, y_{i-1}, \mathbf{x}, i)\right) \tag{3.7}$$

3.1 Inference

Despite subtle differences, the inference process of the conditional random field is quite similar to the hidden markov model: A lattice is built between candidate tags with computed transition probabilities, and Viterbi algorithm (explained in detail in work from Rabiner [1990]) is used to determine the most probable path.

The Viterbi Algorithm

The problem of determination of the most probable state sequence $\mathbf{y} = \{y_1, \ldots, y_m\}$ may be stated as follows:

$$\underset{\mathbf{y}}{\operatorname{argmax}} P(\mathbf{y}|\mathbf{x}) \tag{3.8}$$

Because our model is a bigram model, the equation 3.8 becomes:

$$\underset{\mathbf{y}}{\operatorname{argmax}} \prod_{i=1}^{n(\mathbf{x})} P(y_i|y_{i-1}, \mathbf{x}) \tag{3.9}$$

Let $k \leq n(T_{x_{i-1}}), j \leq n(T_{x_i})$ be positive integers, for all word pairs. Let $(y_{i-1,k}, y_{i,j})$ a candidate path between words x_{i-1} and x_i. Let $C(y_{i-1,k}, y_{i,j})$ for each candidate path be defined as follows:

$$C(y_{i-1,k}, y_{i,j}) = \sum_{a}^{n(\Lambda)} \lambda_a f_a(y_{i-1,k}, y_{i,j}, \mathbf{x}, i) \tag{3.10}$$

Also let:

$$B(y_{i,j}) = \underset{k}{\operatorname{argmax}} C(y_{i-1,k}, y_{i,j}) \tag{3.11}$$

be the best cost for the given tag.

Viterbi algorithm works in the following steps:

1. For all positive integers $i < n(\mathbf{x}), j < n(T_{x_i})$, compute:

   $$\gamma = B(y_{i-1,j})C(y_{i-1,k}, y_{i,j})$$

   for all i, store the j that results in the highest γ as the best previous tag.

2. Choose the tag of the last word with the highest cost to be the inferred tag for that word.

3. For every word before the last one, output the best previous tag that

was stored in step 1 for every tag is chosen to be the inferred tag for the previous word.

3.2 Parameter Estimation

Parameter estimation is the process of finding the parameter vector $\Lambda = \{\lambda_1, \ldots, \lambda_n\}$ that maximizes the likelihood of the training data, given features to be extracted from the training data.

In the seminal work that presented conditional random fields Lafferty et al. [2001] present an iterative scaling algorithm. While this algorithm is quite simple and guaranteed to converge, for rather large feature sets, it turns out be a suboptimal method. In a successor work, Sha and Pereira [2003] show that a quasi-newton method, namely the L-BFGS[3] algorithm [Liu and Nocedal, 1989] works faster than the iterative scaling method. L-BFGS is a widely adopted algorithm for nonlinear optimization tasks with software libraries available for popular computing platforms.

Training the CRF is to find the set Λ that maximizes the likelihood of the training data.

$$\hat{\Lambda} = \underset{\Lambda}{\operatorname{argmax}}\, \mathcal{L}_\Lambda \tag{3.12}$$

where:

[3]short for "Limited-memory Broyden-Fletcher-Goldfarb-Shanno"

$$\mathcal{L}_\Lambda = \sum_{k=1}^{n(\mathbf{c})} \log(P_\Lambda(\mathbf{y}_k|\mathbf{x}_k)) \tag{3.13}$$

$$= \sum_{k=1}^{n(\mathbf{c})} \left(\sum_{i=1}^{n(\mathbf{x}_k)} \sum_a \lambda_a f_a(y_i, y_{i-1}, \mathbf{x}_k, i) - \log(Z_\Lambda(\mathbf{x}_k)) \right) \tag{3.14}$$

where y_i is the tag to the i^{th} word of $\mathbf{x}_k$. This quantity is maximized, when its first derivative:

$$\frac{\partial \mathcal{L}_\Lambda}{\partial \lambda_a} = \sum_{k=1}^{n(\mathbf{c})} \sum_{i=1}^{n(\mathbf{x}_k)} \left(f_a(y_i, y_{i-1}, \mathbf{x}_k, i) - E_{P_\Lambda(\mathbf{y}_k|\mathbf{x}_k)}[f_a] \right) \tag{3.15}$$

is zero, where have:

$$E_{P_\Lambda(\mathbf{y}|\mathbf{x})}[f] = \sum_{y,y'} f(y, y', \mathbf{x}, i) P_\Lambda(y, y'|\mathbf{x}) \tag{3.16}$$

$$= \sum_{y,y'} f(y, y', \mathbf{x}, i) \frac{\alpha_{i-1,y'}(\mathbf{x}) \exp(\sum_a \lambda_a f_a(y, y', \mathbf{x}, i)) \beta_{i,y}(\mathbf{x})}{Z_\Lambda(\mathbf{x})} \tag{3.17}$$

where y, y' iterates over all bigram combinations.

Let: $\forall y \in T : \alpha_{0,y} = 1$ and $\beta_{n(\mathbf{x})+1,y} = 1$. We define:

$$\forall i \in \mathbb{N}, \mathbf{x} \in \mathbf{c} \quad | \quad 1 \leq i \leq n(\mathbf{x})$$

$$\alpha_{i,y}(\mathbf{x}) = \sum_{j=1}^{n(T_{x_{i-1}})} \left(\alpha_{i-1} \exp(\sum_a \lambda_a f_a(y, y_{i-1,j}, \mathbf{x}, i)) \right) \tag{3.18}$$

$$\beta_{i,y}(\mathbf{x}) = \sum_{j=1}^{n(T_{x_{i+1}})} \left(\beta_{i+1} \exp(\sum_a \lambda_a f_a(y, y_{i+1,j}, \mathbf{x}, i)) \right) \tag{3.19}$$

To avoid overfitting, the likelihood function is penalized with a L2-norm of the Λ vector, and weighted with a predefined scalar C which makes equation 3.14 as follows:

$$\mathcal{L}_\Lambda = \sum_{k=1}^{n(\mathbf{c})} \left(\sum_{i=1}^{n(\mathbf{x}_k)} \sum_a \lambda_a f_a(y_i, y_{i-1}, \mathbf{x}_k, i) - \log(Z_{\mathbf{x}_k}) \right) - \frac{1}{2C} \sum_a \lambda_a^2 \tag{3.20}$$

and thus its first derivative:

$$\frac{\partial \mathcal{L}_\Lambda}{\partial \lambda_a} = \sum_{k=1}^{n(\mathbf{c})} \sum_{i=1}^{n(\mathbf{x}_k)} \left(f_a(y_i, y_{i-1}, \mathbf{x}_k, i) - E_{P_\Lambda(\mathbf{y}_k|\mathbf{x}_k)}[f_a] \right) - \frac{1}{C} \lambda_a$$

The alpha values are computed during a forward pass, and the beta values are computed during a backward pass, for each candidate tag of a given word. Once the expectations for features are also computed using these values, the feature expectations and current feature coefficients are passed to the L-BFGS algorithm, along with the penalized derivative.

Once the L-BFGS algorithm has updated the feature coefficients, the

derivatives are re-computed. if the change is significant ($> 0.01\%$) compared to previous derivative values, the values are re-updated with the L-BFGS algorithm. If this is not the case four times in a row, the training process is terminated.

Chapter 4

Implementation

We will use a one-dimensional conditional random field chain as the probability distribution that will model the training data. The implementation of the algorithm has two distinct parts: The training part and the testing part. They operate on the same data structures, but the operations they perform on the said data structures are completely different.

4.1 Representing Input Turkish Words

The Turkish word is the common input to both the training and the testing parts of the algorithm. The raw surface of a Turkish word is processed by the morphological analyzer[Oflazer, 1994, Oflazer et al., 1999], which results in the kind of output that can be observed in Table 4.1.

As a data structure, the Turkish word can be said to have one surface and one or more analyses. Every analysis is made of one word root and one or more inflectional groups. We start by defining the inflectional group.

4.1.1 The Inflectional Group

Work on the morphological analysis of Turkish prior to 1999 was not using inflectional groups [Oflazer, 1994]. It instead returned a chain of morphological markers computed from the input word. During the work of designing a Turkish treebank Oflazer et al. [1999] decided that grouping morphological markers under inflectional groups contributed to both the expresiveness and the tractability of the morphological analyses.

Quoting [Çetinoğlu and Oflazer, 2006]:

> Inflectional groups represent the inflectional properties of segments of a complex word structure separated by derivational boundaries. An inflectional group is typically larger than a morpheme but smaller than a word (except when the word has no derivational morphology in which case the inflectional group corresponds to the word).

In other words, a new inflectional group is "started" whenever a new derivation is detected. A "derivation" is not necessarily a suffix that substantially changes the semantics of a word. In fact, in cases of zero-derivations, there may not even be an explicitly observable morpheme that makes this derivation. The word *yıllarıydı* is an example of this phenomenon, whose analyses can be seen in Table 4.1. As can be seen, the noun root *yıl* is inflected to become *yılları*, which is then inflected with the tense marker `+Past` to become *yıllarıydı*[1] which, by definifiton, can only inflect verbs.

It should be noted that the derivation from the noun *yılları* to the verb *yıllarıydı* happened without any explicit notation.

[1] whose infinitive form can only be expressed with the help of an auxiliary verb *yılları olmak*

```
yıl  +Noun+A3pl+P3pl+Nom  ^DB +Verb+Zero+Past+A3sg
yıl  +Noun+A3pl+P3sg+Nom  ^DB +Verb+Zero+Past+A3sg
yıl  +Noun+A3sg+P3pl+Nom  ^DB +Verb+Zero+Past+A3sg
```

Table 4.1: Analyses of the word *yıllarıydı*

Representation

Implementations of the published research on morphological disambiguation examined so far represent the inflectional group as a regular string of characters. While this approach is quite easy to implement, it is suboptimal to store the inflectional group data inside a variable-length delimited and unordered data structure. Having a fixed-size data structure that has "slots" for morphological markers conserves both space and time when storing and manipulating the data inside an inflectional group.

Fixing the size of the inflectional group also has a theoretical advantage: It will be possible to create features that test individual morphological markers, without testing for impossible morphological marker combinations.

For above reasons, a fixed-size inflectional group is a desirable data structure.

But as can be seen in *e.g.* Tables 2.1 and 4.4, the maximum size of the inflectional group is not obvious. In an effort to explore the possibilty of a fixed-size data structure that can accomodate every Turkish inflectional group, a 330-million-word corpus was scanned in order to gather a list of morphological marker pairs that are observed in the same inflectional group. This list of pairs can be interpreted as a list of constraints that depict which morphological markers should be put in separate slots.

When this list of pairs is interpreted as a graph, one can obtain the optimal grouping of the morphological markers by choosing one of the graph partititionings that has the least number of partitions. For the following

reasons, this operation was carried out semi-automatically, instead of automatically:

- The graph partitioning problem is NP-complete.
- The number of partitions that fit into this definition is likely to be greater than one.
- The resulting partitioning should linguistically make sense as well.

The automatic part of the algorithm consisted of an iterative scheme that created new partitions for morphological markers that appear in the same inflectional group, as the file was being scanned. The result was then adjusted manually, heeding linguistic properties of the partitioned morphological markers. The resulting 9-partition that was deemed more appropriate from a linguistic perspective can be seen in table 4.2. During the course of this work, these partitions were given special names and are termed *variables*.

Among those variables, `vTense` is special, as it can appear up to three times in an inflectional group[2]. Because of this fact, while not shown in the table 4.2, there are additional variables that are termed `vTense2` and `vTense3` in the implementation.

As for the `vRoot` variable, it is not considered to be part of the inflectional group, but of the analysis altogether. The word roots are are assigned an integer identifier on first occurence, contrary to being pre-defined like the other variables.

[2]Kemal Oflazer, private communication.

Variable	Morphological markers
vRoot	Every possible word root.
vAgr	**Agreement markers:** +A1sg, +A2sg, +A3sg, +A1pl, +A2pl, +A3pl, +NullAgr
vMisc	**Miscellaneous markers:** +Able, +Acquire, +ActOf, +Adamantly, +AfterDoingSo, +Agt, +Almost, +AsIf, +AsLongAs, +As, +Become, +ByDoingSo, +Card, +Caus, +Continue, +DemonsP, +Dim, +Distrib, +EverSince, +FitFor, +FutPart, +HastilyFeelLike, +InBetween, +Inf1, +Inf2, +Inf3 +Inf, +JustLike, +Ly, +Ness, +NotAbleState, +NotState, +Ord, +PCAbl, +PCAcc, +PCDat, +PCGen, +PCIns, +PCNom, +Part, +Pass, +PastPart, +Percent, +PersP, +QuantP, +QuesP, +Range, +Ratio, +Real, +Recip, +ReflexP, +Related, +Repeat, +SinceDoingSo, +Since, +Start, +Stay, +Time, +When, +While, +WithoutBeingAbleToHaveDoneSo, +WithoutHavingDoneSo, +Without, +With, +NullMisc
vPol	**Polarization markers:** +Pos, +Neg, +NullPol
vPos	**Part-of-speech markers:** +Adj, +Adverb, +Verb, +Num, +Punc, +Conj, +Dup, +Det, +Postp, +Interj, +Pron, +Noun, +NullPos
vPoss	**Possessive markers:** +P1sg, +P2sg, +P3sg, +P1pl, +P2pl, +P3pl, +Pnon, +NullPoss
vProp	**The proper noun marker:** +Prop, +NullProp
vTense	**Tense markers:** +Aor, +Cond, +Desr, +Fut, +Imp, +Narr, +Neces, +Opt, +Past, +Pres +Prog1, +Prog2, +NullTense1, +NullTense2, +NullTense3
vZero	**The Zero derivation marker:** +Zero, +NullZero

Table 4.2: Random variables and the markers they may emit.

```
1  vRoot(iç+Verb),
2  vRoot(iç+Noun),vPoss(P2sg),
3  vRoot(iç+Noun),vPoss(Pnon),
4  vRoot(için+Postp),vMisc(PCGen),
5  vRoot(için+Postp),vMisc(PCNom),
```

Table 4.3: Tags for the word *için*.

So, we conclude that the inflectional groups that form Turkish words are made of at most 11 morphological markers. There are 129 morphological markers emitted by the current implementation of the morphological analyzer. Every variable also said to have its own `Null*` value to be able to distinguish the case of the non-existance of a marker for the given variable.

4.1.2 The Ambiguity Tree

The ambiguity tree is a way of obtaining *distinguishing morphological markers* of a word's morphological analysis, compared to other analyses of the same word. It's a decision tree that is $\max_{1 \leq i \leq a}(n_i \times 11)$ levels deep (where a is the number of analyses for a given word and n is the number of inflectional groups in the given analysis) which has variables as its nodes and the values to those variables assigned to the transitions between these nodes.

In order for it to be compatible with other words, an insertion order must be fixed before running the algorithm. While the choice of the insertion order is arbitrary, it can be used to discover which morphological markers have more distinguishing power compared to others.

It is constructed as follows:

1. All morphological analyses of a word are obtained using the morphological analyzer.

2. For every analysis:

(a) Every morphological marker is inserted to the ambiguity tree using the fixed insertion order.

(b) The resulting tree is searched depth-first for nodes with more than one children, and these are pushed in a stack.

(c) For every encountered leaf node, the variables in the stack are printed as the distinguishing morpholgical markers for the given analysis.

The ambiguity tree for the word *için*, and the distinguishing morphological markers that result from this ambiguity tree can be seen in Figure 4.1 and Table 4.3 respectively.

Applicability

Initially, the ambiguity tree approach was an attempt to fix the number of possible tags in the morphological disambiguation problem. This had the potential to express the morphological disambiguation problem as a regular one dimensional markov chain *with known tags*. This in turn had the advantage of letting us use one of the existing implementations of the conditional random field, as no known free implementation supports tagging with incomplete information. However, initial tests showed that the information that the ambiguity tree algorithm was throwing away was crucial, and thus can't just be discarded. Also, according to the ambiguity tree algorithm, words with no ambiguities don't have any tags, which effectively ignores them during the training process, another important drawback of the algorithm.

However, as mentioned before, it's an important tool that shows the influence of the morphological markers. It was especially useful in a specific case, where the decision about which null values to include in the model had to be taken.

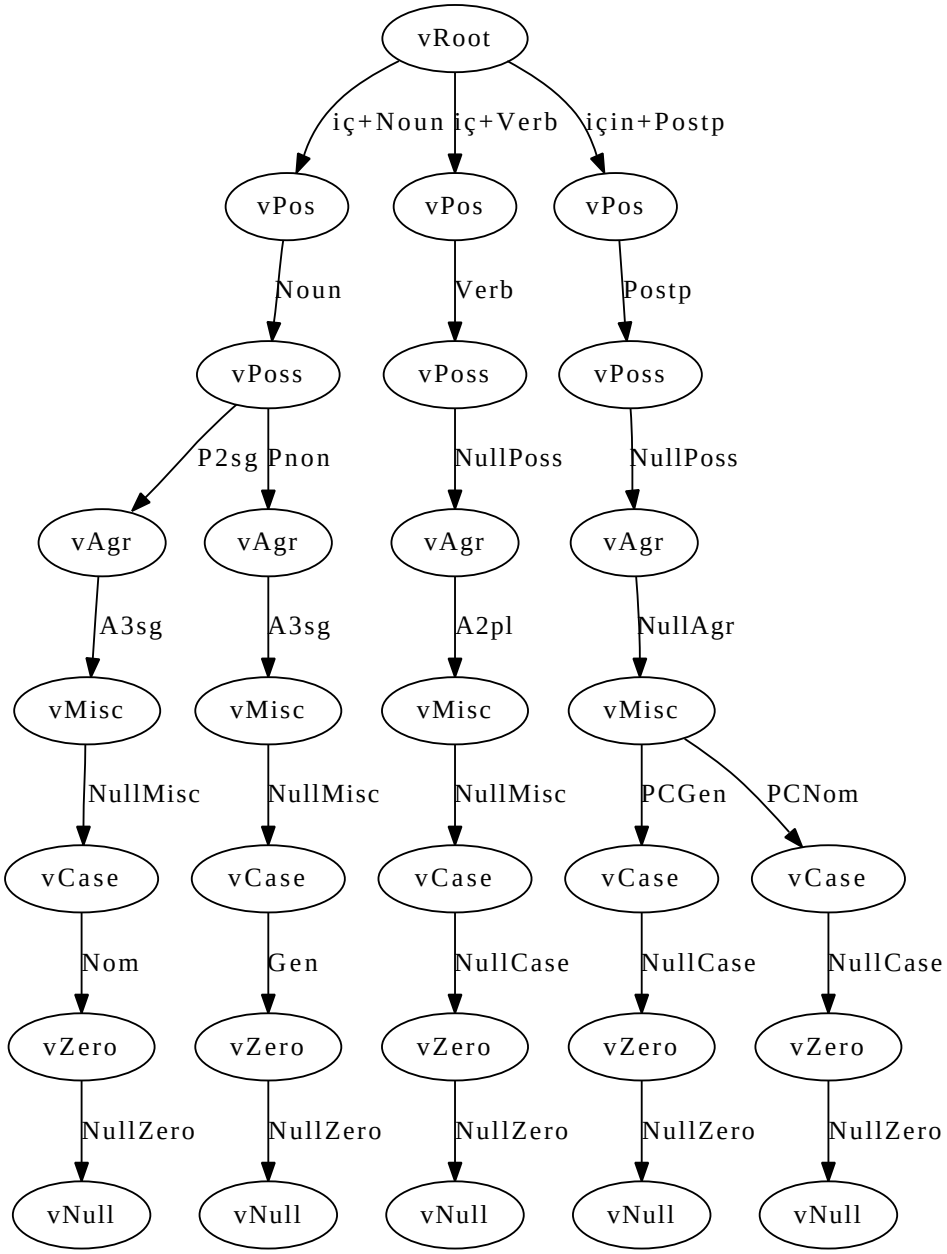

Figure 4.1: Analysis tree of the word *için*

Null values are the artifacts of the fixed-size inflectional group paradigm that had been adopted. Features that test, for example, the `vTense` slot to be null don't contribute to the quality of the model because it's an ambiguous information: `vTense` slots may be null for a number of reasons.

The distinguishing null variables are caused by the difference in the number of inflectional groups between the analyses of a word. That in turn is caused by three factors:

1. `+Zero` derivations
2. Different word roots
3. Different parts-of-speech of the word root.

The minimal branching scheme that avoids null branches was found to be the following:

`vRoot+vPos`[3]`, vPos, vPoss, vAgr, vMisc, vCase, vZero`

The morphological markers are inserted starting from the last inflectional group, and going left.

The variables that are left out, namely `vTense{1,2,3}, vProp and vPol` are seen to have no distinctive power for morphological disambiguation, and `vCase` has distinctive power in rare cases; e.g. only for words like *öte*, *yukarı* etc. which can be used as a noun in both nominative or dative case in their given forms. Additionally, the only the null marker to have distinctive power was found to be `+NullZero`.

Using this scheme, we discovered that, in our training data, which is a 837,293-word corpus, where 410,124 (approx. 49%) tokens are ambiguous, there are 53,332 distinct word classes, of which 17,785 (approx. 33%) distinct cases are seen as correct answers.

[3]Part-of-speech of the first inflectional group

4.2 Morphological Feature Templates

As the usage of the distinguishing morphological markers as tags had catastrophic results, a more fine-grained approach to testing was adopted. That's why functions that can test individual morphological markers (and combinations of those morphological markers) are implemented. They work on the analysis represented by the tag produced by the ambiguity tree algorithm, not the tag itself.

Two main types of feature templates exist: Simple and complex. Simple features are individual tests on the training data, whereas complex features are combinations of those tests.

The following are simple test templates:

`S`: Surface test

The surface test returns 1 when the given surface in the feature is equal to the surface of the given word. It has no distinctive power by itself, as it's true for every possible tag of a given word – It's just a way of specializing produced features to n-grams, which gives way to modeling the behaviour of n-grams separately, in hopes of obtaining more specific but successful models.

It should be noted that, as this feature template is independent of the tags, it can be used to incorporate information from any part of the sentence.

`SC`: Case test

The case test returns 1 when the given surface's given initial number of letters is uppercase or lowercase, depending on the given value of the feature. Same comments about the distinction power of the Surface feature template (above) apply to this feature as well.

It should be noted that, similarly to the surface test, this feature template can be used to incorporate information from any part of the sentence.

This test can have a custom uppercase/lowercase parameter and initial number of letters.

`XU`: eXtended Unigram test

The extended unigram test returns 1 when the given variable in a given inflectional group has a given morphological feature as value when tested against a given analysis of a given word.

This test can be customized to include a certain range of inflectional groups, and / or to include or exclude certain list of variables.

`FreqU`: Unigram Frequency test

The extended bigram test returns 1 when the given variable in a given inflectional group has the given morphological feature as value *more frequently* than another given morphological feature as value in the training data, when tested against a given analysis of the *current* word, and 0 otherwise or there's no such inflectional group. It's a slower test because it has to look up frequency ranks for all other values in all other analyses of a given word.

This test can be customized to include a certain range of inflectional groups, and / or to include or exclude certain list of variables.

`XB`: eXtended Bigram test

The extended bigram test returns 1 when the given variable in a given inflectional group has a given morphological feature as value when tested against a given analysis of the *current* word *and* when the given variable in a given

3 has a given morphological feature as value when tested against a given analysis of the *previous* word, and 0 otherwise.

This test can be customized to include a certain range of inflectional groups, and / or to include or exclude certain list of variables for both sides of the edge.

`FreqB`: Bigram Frequency test

The extended bigram test returns 1 when: the given variable in a given inflectional group has a given morphological feature as value when tested against a given analysis of the *current* word *and* when the given variable in a given inflectional group has a given morphological feature as value when tested against a given analysis of the *previous* word, ***more frequently than*** the given morphological markers for the same variable pair, when tested against the current analysis of the current word, the current analysis of the previous word, and other every other analysis combination of the current and the previous word.

Obviously, this is the slowest test, as it makes a test against every possible analysis pair. It is not feasible to use this feature without implementing some caching mechanism, which in turn is observed to have a huge penalty in memory usage. So, unless a more elaborate caching mechanism that properly balances space and time costs is implemented, this template should not be used in the conducted tests.

This test can be customized to include a certain range of inflectional groups, and / or to include or exclude certain list of variables for both sides of the edge.

`AndB`: Bigram *And* Operator

The bigram *and* operator is a tool for combining feature templates. It combines one feature group per template, according to the combination mode given in definition-time. Three such grouping modes exist:

Per Analysis: It combines features for every analysis pair. This mode generates the least number of `AndB` features, which are in turn more complex and specific.

Per IG: It combines features for inflectional group pairs in every analysis.

Per Variable: It combines features for variable pairs in inflectional groups in every analysis. In this mode, exactly one feature is combined for every feature template, but the number of individual `AndB:` features is highest.

This test is formed of other tests, which are combined using three different slots:

1. **Left slot** operates on the previous word and accepts only unigram tests.
2. **Right slot** operates on the current word and accepts only unigram tests.
3. **Middle slot** operates on both the current word and the previous word, and accepts only bigram tests.

The unigram and bigram tests are separated for performance reasons.

4.3 A Sample Run

Consider the following sentence:

Bence geldiğini duymasınlar. (4.1)

The inference lattice can be seen in Figure 4.2.

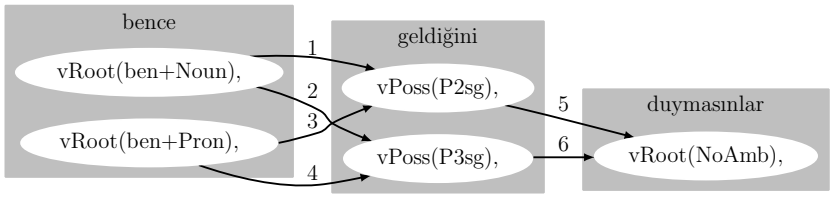

Figure 4.2: The disambiguation lattice for sentence 4.1

We use feature template from Test 9 as an example, where we generate bigram tests including every variable, and later grouping them as separate features by inflectional group pair.

Root	**Inflectional groups**
bence	
ben	`+Noun+A3sg+Pnon+Equ`
ben	`+Pron+Pers+A1sg+Pnon+Equ`
geldiğini	
gel	`+Verb+Pos^DB+Noun+PastPart+A3sg+P2sg+Acc`
gel	`+Verb+Pos^DB+Noun+PastPart+A3sg+P3sg+Acc`
duymasınlar	
duy	`+Verb+Neg+Imp+A3pl`

Table 4.4: List of possible analyses of the words of the sentence 4.1.

Once generated, the features are looked up in the database, in order to retrieve their λ coefficients. Unseen features are ignored, as their coefficients are assumed to be 0. These coefficients are later summed to obtain the weight for each path. From now on, the process is the same as any other statistical model that does inference on a one dimensional lattice: The Viterbi algorithm

Path No	Features that return `true` for the given path			
	Left word		Right word	
	IG	Vars	IG	Vars
1	-1	`ben+Noun+Pnon+A3sg+Equ`	0	`+Verb`
	-1	`ben+Noun+Pnon+A3sg+Equ`	1	`gel+Noun+A3sg+Acc+P2sg +PastPart`
2	-1	`ben+Noun+Pnon+A3sg+Equ`	0	`+Verb`
	-1	`ben+Noun+Pnon+A3sg+Equ`	1	`gel+Noun+A3sg+Acc+P3sg +PastPart`
3	-1	`ben+Pnon+A1sg+Pron+Equ +Pers`	0	`+Verb`
	-1	`ben+Pnon+A1sg+Pron+Equ +Pers`	1	`gel+Noun+A3sg+Acc+P2sg +PastPart`
4	-1	`ben+Pnon+A1sg+Pron+Equ +Pers`	0	`+Verb`
	-1	`ben+Pnon+A1sg+Pron+Equ +Pers`	1	`gel+Noun+A3sg+Acc+P3sg +PastPart`
5	-2	`+Verb`	0	`duy+Verb+Neg+Imp+A3pl`
	-1	`gel+Noun+A3sg+Acc+P3sg +PastPart`	0	`duy+Verb+Neg+Imp+A3pl`
6	-2	`+Verb`	0	`duy+Verb+Neg+Imp+A3pl`
	-1	`gel+Noun+A3sg+Acc+P3sg +PastPart`	0	`duy+Verb+Neg+Imp+A3pl`

Table 4.5: The generated features for lattice in Figure 4.2

is used to combine these weights with their neighbors, and the most likely path is printed to the output stream.

4.4 Factors that Influence Resource Usage

The two distinct parts of the conditional random field implementation have different resource requirement characteristics.

4.4.1 The Training Part

The memory usage of the training algorithm depends in large part on the number of features that are generated during the preprocessing stage of the algorithm, where the training data is loaded into memory. The training data and the features themselves are stored in a key/value store, which may or may not store the bulk of the training data in non-volatile memory[4].

The speed of the training process does not depend on the number of features in total, but on the number of features per lattice path. It is observed that, as the number of features per path increase, both time spent per path and the number of iterations required to optimize a model increase. So, templates that group less number of tests per feature are observed to be more costly to train.

As computing the gradient is about computing a vector of sums, this computation can be effectively divided to multiple threads. However, once the gradient is computed, the L-BFGS optimization runs in a single thread, as the one dimensional chain implementation used in this project is inherently atomic. For this reason, increasing the number of threads won't result in the

[4]Current state of our implementation stores everthing in volatile memory, as the speed of execution was of paramount importance when doing numerous training sessions in order to evaluate various feature template schemes.

desired reduction in the execution time, unless the model is big enough to be divided to multiple threads.

4.4.2 The Testing Part

The memory usage and the speed of the testing process depend on the number of paths in the testing process, as well as the number of features per path. It should be noted that a scheme that uses one thread per sentence would extract most out of today's popular multi-core architectures.

4.5 Results

The flexibility of the conditional random field framework is also one of the challenges of working with it. As one can't use every possible test on training data, some sort of feature selection method *prior to* training must be adopted.

The reported results are summaries of 10 tests that are random excerpts of the training data, shuffled in the sentence level. In each set, the training data contains approximately 800.000 words and the testing data contains approximately 35.000 words.

No Tests

As a comparison basis, a tagger that emits random tags (among the possible ones) is implemented and run 10 times. It averages on approximately 70% accuracy.

No	Template	Average	Min	Max
1	All IGs, all variables, grouped by Analysis.	93.21%	93.03%	93.59%
2	All IGs, all variables except **vProp**,**vPol** and **vTense**$\{1,2,3\}$ grouped by analysis.	93.34%	93.17%	93.7%
3	All IGs, all variables, grouped by IG.	93.58%	93.36%	94.01%
4	All IGs, all variables except **vProp**,**vPol** and **vTense**$\{1,2,3\}$ grouped by IG.	93.55%	93.26%	93.96%
5	All IGs, all variables, non-grouped.	92.63%	92.39%	93.01%
6	All IGs, all variables except **vProp**,**vPol** and **vTense**$\{1,2,3\}$ non-grouped.	92.31%	91.88%	92.62%

Table 4.6: Unigram test summaries on 10 random data sets.

4.5.1 Unigram Tests

It is very easy to obtain above-90% scores using only unigram tests. See the results in Table 4.6.

Judging by the numbers, we can conclude that the variables omitted by the ambiguity tree algorithm don't have significant influence on model performance. Small deviations we observe in the accuracy scores are due to the randomness of training and test data sets[5].

4.5.2 Bigram Tests

We observed that bigram tests tend to be more sensitive to overfitting. Grouping tests in smaller groups is observed to increase accuracy, as can be seen from the increasing trend in Tests 8, 9 and 12, in Table 4.7.

4.5.3 Other Combinations

Combining various high-scoring tests increases the overall accuracy of the model. Apparently, the limitations of the bigram model does not allow us

[5]Using other sets of training/testing data pairs, we've observed as small deviations, sometimes in other directions.

No	Template	Average	Min	Max
7	Only **vRoot** and **vPos** pairs from the first IG, from both sides, grouped by analysis pair.	82.05%	77.85%	87.21%
8	All variables from all IGs from both sides, grouped by analysis pair.	85.83%	84.39%	86.44%
9	All variables from all IGs from both sides, grouped by IG pair.	89.09%	88.36%	89.43%
10	All variables from last IG from left and all IGs from right side grouped by IG pair.	87.79%	86.72%	88.39%
11	All variables except **vRoot** from all IGs from both sides, grouped by IG pair.	92.93%	92.63%	93.6%
12	All variables from all IGs from both sides, not grouped.	94.83%	94.67%	94.98%
13	All variables from all IGs from right side and last ig from left side, not grouped.	94.78%	94.61%	94.93%
14	All variables except vRoot from all IGs from both sides, not grouped.	92.41%	92.11%	93.28%
15	All variables except vRoot from all igs from the right side and last ig from left side, not grouped.	92.24%	91.95%	93.17%

Table 4.7: Bigram test summaries on 10 random data sets.

No	Template	Average	Min	Max
16	Combination of templates from Tests 2, 8 and 11.	96.03%	95.74%	96.43%
17	Combination of templates from Tests 4, 7 and 14.	95.78%	95.62%	96.14%

Table 4.8: Miscellaneous test summaries on 10 random data sets.

to go above 96% scores. Two highest scores from numerous tests we did can be found in Table 4.8. As can be seen, the scores from Tests 16[6] and 17 are very close.

4.5.4 Error Analysis

An analysis of the errorneous parses from the highest scoring Test 16 is presented in this section.

We observe that:

- 46.84% of errors are due to the **vRoot** variable being incorrectly tagged.
- 29.32% of errors are due to the **vPos** variable being incorrectly tagged.
- 20.78% of errors are due to the **vPoss** variable being incorrectly tagged.
- Remaining 3.06% of errors are due to **vAgr**, **vMisc**, **vZero** and **vCase** being incorrectly tagged.

We see that it is much harder to disambiguate the **vRoot** variable. The ideal method to attack this problem is to perform deeper semantic analysis within a wider window of context. In cases where this is not an option, two approaches may be adopted:

1. Using a separate language model to disambiguate just the **vRoot** variable,

[6]Test 16 combines the test numbers 4,17 and 3 from Sak et al. [2007].

2. Putting the **vRoot** variable in a lower rank in the ambiguity tree hierarchy, re-generate distinguishing morphemes, and design feature templates that try to be more independent of the root morpheme.

Remaining most significant confusions are between adjectives and nouns, where **vPos** variable was tagged `+Adj` instead of `+Noun` accounted for 8.46% of errors and the reverse case accounted for 6.77% of errors, 15.23% in total. Most of the following ranks are for **vPoss** mistaggings, that account for 18.18% of the errors in total.

Both the `+Adj` vs. `+Noun` and the **vPoss** mistagging cases are due to the weaknesses of the bigram model: Adjectives may naturally be farther than one word to the word they modify. However, this is a more serious problem for possessive markers, which are typically more distant to the words they are in relation with, especially as the sentence gets longer.

Chapter 5

Conclusions and Future Work

In this thesis, we have suggested a slot-based representation of the inflectional group. Basing on this representation, we've suggested a scheme for building feature-based language models using morphological information from candidate analyses of a given surface form in a fine-grained manner. We later used Conditional Random Fields as a demonstration framework for this scheme, where we observed accuracy scores to be in line with the state of the art.

Feature research may try to address the following weak points of the disambiguator presented in this thesis:

- The template generation framework has room for improvement. In an effort to further simplify the language model, adopting more sophisticated feature selection methods that have either linguistic or mathematical foundations is a promising prospect. A decrease in the number of features of a model is desirable because it is observed to increase the accuracy and the robustness of the language model, while decreasing its resource requirements.

- The bigram model is unable to express higher order relations between

words. Adopting a model that can process information from a wider window of words, like a Semi-Markov Conditional Random Field [Sarawagi and Cohen, 2004] would certainly contribute to the overall performance of the disambiguation task.

- Relaxing the assumption of sentences being independent of each other is also a desirable property – sometimes the information in the sentence alone is not enough. For example, it is not possible to disambiguate the possessive marker in the word *geldiğini* in Sentence 4.1 without taking inter-sentential relations into account. For this reason, exploring possible uses of inter-sentential relations is also a promising prospect.

- The L2-Regularizer is *too fair*. Exploring ways to boost the weights to some features is a promising prospect.

When adopting a framework that can tractably express higher order relations between words, the problem of feature explosion will become more serious. Projects that will adopt higher order models will better make use of aggressive feature reduction techniques, compared to bigram models.

Bibliography

Gükhan H. Bakir, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S. V. N. Vishwanathan. *Predicting Structured Data (Neural Information Processing)*. The MIT Press, 2007. ISBN 0262026171.

Özlem Çetinoğlu and Kemal Oflazer. Morphology-syntax interface for turkish lfg. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 153–160, Sydney, Australia, July 2006. Association for Computational Linguistics. doi: 10.3115/1220175.1220195. URL `http://www.aclweb.org/anthology/P06-1020`.

Dilek Z. Hakkani-Tür, Kemal Oflazer, and Gökhan Tür. Statistical morphological disambiguation for agglutinative languages. In *Proceedings of the 18th conference on Computational linguistics*, pages 285–291, Morristown, NJ, USA, 2000. Association for Computational Linguistics. ISBN 1-55860-717-X. doi: http://dx.doi.org/10.3115/990820.990862.

Taku Kudo, Kaoru Yamamoto, and Yuji Matsumoto. Applying conditional random fields to japanese morphological analysis. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 230–237, Barcelona, Spain, July 2004. Association for Computational Linguistics.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Carla E. Brodley and Andrea Pohoreckyj Danyluk, editors, *ICML*, pages 282–289. Morgan Kaufmann, 2001. ISBN 1-55860-778-1.

D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Math. Program.*, 45(3):503–528, 1989. ISSN 0025-5610. doi: http://dx.doi.org/10.1007/BF01589116.

K. Oflazer. Two-level description of turkish morphology, 1994. URL `citeseer.ist.psu.edu/oflazer93twolevel.html`.

Kemal Oflazer and Ilker Kuruöz. Tagging and morphological disambiguation of turkish text. In *Proceedings of the Fourth ACL Conference on Applied Natural Language Processing (13–15 October 1994, Stuttgart)*, 1994. URL `citeseer.ist.psu.edu/oflazer94tagging.html`.

Kemal Oflazer and Gökhan Tür. Morphological disambiguation by voting constraints. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, pages 222–229, Morristown, NJ, USA, 1997. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/979617.979646.

Kemal Oflazer and Gokhan Tur. Combining hand-crafted rules and unsupervised learning in constraint-based morphological disambiguation. In Eric Brill and Kenneth Church, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 69–81. Association for Computational Linguistics, Somerset, New Jersey, 1996. URL `citeseer.ist.psu.edu/article/oflazer96combining.html`.

Kemal Oflazer, Dilek Z. Hakkani-Tür, and Gökhan Tür. Design for a turkish treebank. In *Proceedings of the Workshop on Linguistically Interpreted Corpora, EACL 99*, Bergen, Norway, 1999. URL `/ref/oflazer/ttbank.pdf`.

Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Readings in speech recognition*, pages 267–296, 1990.

Haşim Sak, Tunga Güngör, and Murat Saraçlar. Morphological disambiguation of Turkish text with perceptron algorithm. In *CICLing 2007*, volume LNCS 4394, pages 107–118, 2007. URL `http://www.cmpe.boun.edu.tr/~hasim/papers/CICLing07.pdf`.

Sunita Sarawagi and William W. Cohen. Semi-markov conditional random fields for information extraction. In *In Advances in Neural Information Processing Systems 17*, pages 1185–1192, 2004.

Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proceedings of HLT-NAACL 2003*, pages 213–220. Association for Computational Linguistics, 2003. URL `PDF/shallow.pdf`.

Charles Sutton and Andrew Mccallum. An introduction to conditional random fields for relational learning. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, 2006. URL `http://www.cs.umass.edu/~casutton/publications/crf-tutorial.pdf`.

Douglas L. Vail, John D. Lafferty, and Manuela M. Veloso. Feature selection in conditional random fields for activity recognition. In *IROS* Vail et al. [2007], pages 3379–3384. URL `http://dblp.uni-trier.de/db/conf/iros/iros2007.html#VailLV07`.

Deniz Yuret and Ferhan Türe. Learning morphological disambiguation rules for turkish. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 328–334, Morristown, NJ, USA, 2006. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1220835.1220877.