TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

MASTERARBEIT

Recognizing Structure in Report Transcripts

An Approach Based on Conditional Random Fields (CRFs)

Ausgeführt am Zentrum für Hirnforschung

Institut für Medizinische Kybernetik und Artificial Intelligence

der Medizinischen Universität Wien

unter der Anleitung von

ao.Univ.Prof. Dipl.-Ing. Dr.techn. Harald Trost

durch

Jeremy M. Jancsary, Bakk.techn.

Kalvarienberggasse 18/1/15

A-1170 Wien

Wien, am 5. Februar 2008

Jeremy M. Jancsary

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 5. Februar 2008 Jeremy M. Jancsary

Abstract

Typically, the output of Automatic Speech Recognition (ASR) is a mere sequence of words. This view may be sufficient for some tasks, whereas others require a more structured approach. This thesis presents a framework that allows for identification of deep, underlying structure in report dictations. Identification of structural elements, such as headings, sections and enumerations is an important step towards automatic post-processing of dictated speech. The contributions of this thesis include a generic approach that can be integrated seamlessly with existing ASR solutions and provides structured output, as well as a freely available Conditional Random Field (CRF) toolkit that forms the basis of aforementioned approach and may also be applicable to numerous other problems.

Kurzfassung

Typischerweise gibt automatische Spracherkennung lediglich eine Folge von Wörtern aus. Diese Sichtweise mag für einige Anwendungen ausreichend sein; andere wiederum benötigen eine etwas strukturiertere Vorgehensweise. Diese Diplomarbeit stellt ein Framework vor, das es ermöglicht, zugrundeliegende Strukturen in diktierten Berichten zu erkennen. Die explizite Ausweisung von strukturellen Elementen wie Abschnitten, Überschriften und Aufzählungen ist ein wichtiger Schritt in Richtung automatischer Nachverarbeitung von Diktaten. Der wissenschaftliche Beitrag dieser Diplomarbeit ist einerseits die Entwicklung eines generischen Ansatzes, der bestehende Spracherkennungssysteme dahingehend erweitert, dass strukturierte Ausgabe generiert werden kann; andererseits liegt der Beitrag in der Veröffentlichung eines frei verfügbaren Conditional Random Field (CRF) Toolkits, das dem zuvor genannten Ansatz zugrunde liegt, aber auch für viele andere Problemstellungen einsetzbar ist.

Acknowledgement

I would like to thank my advisor, Harald Trost, for his encouragement throughout the duration of this project and for introducing me to the field of computational linguistics. I am greatly indebted to my colleagues at OFAI[1] for their advice and many fruitful discussions.

Furthermore, I would like to thank Andrew McCallum and Charles A. Sutton[2] for their insightful publications on Conditional Random Fields, which got me interested in the topic in the first place.

[1] http://www.ofai.at
[2] http://cs.umass.edu

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

> *“ The beginning is the most important part of the work. ”*
>
> *– Plato*

Automatic Speech Recognition (ASR) has now reached a state where it can be used successfully for everyday tasks. Its users are as diverse as individuals dictating letters into their word processor software and companies incorporating ASR into their automatic inquiry systems. Yet other companies provide transcription services, facilitating the work of their professional typists via ASR.

Traditionally, utterances are treated by ASR systems as mere sequences of words. Indeed, this view is sufficient for many tasks. However, when it comes to processing dictations of clearly structured text – like reports, for instance – capturing words is only one particular facet of a much wider task.

This thesis presents a means of enhancing existing speech recognition systems in a non-intrusive way such that deep, underlying structure can be identified in dictated reports, rather than a mere sequence of words. The domain of medical reports serves as the application area that will be portrayed throughout this document; however, the techniques and findings should be equally applicable to any domain comprising dictations of highly structured text.

It should be noted that the focus of this thesis are machine learning aspects of structure identification rather than concrete technical integration with available ASR software or speech recognition itself.

1.1 Motivation

The need to dictate reports arises in various domains. While it may still be common for a secretary to transcribe these dictations and create a well-formatted document, it makes sense to start employing ASR at a certain point. Often, the process will then be outsourced to a third party offering professional transcription services.

ASR can help professional typists by handling much of their original work. Yet, some tasks remain that still have to be performed manually. First, the output of the speech recognition software needs to be proof-read, and any recognition errors need to be corrected. Most of these errors are inherently hard to avoid – they can be due to homophones[1], sloppy pronunciation or various other speech-related phenomena such as hesitations and repetitions by the speaker. On a related note, speakers can direct instructions to the transcriptionist which have to be interpreted by a human being. Therefore, *some* amount of manual work will always have to be performed.

Another task that is typically performed by a professional typist is proper formatting, arranging and structuring of a dictated report. Missing headings may have to be inserted, sentences must be grouped into paragraphs in a meaningful way, enumeration lists may have to be introduced and properly indented. These activities require some amount of domain knowledge. However, in some domains, such as medical consultations, most reports bear a striking resemblance. Fairly clear guidelines exist with regards to what has to be dictated, and how it should be arranged. This indicates that part of this task could be automated.

The first step towards achieving that goal is the identification of various structural elements in a dictated report. This forms the basis for later rearrangement, rephrasing and formatting, should it be necessary.

Consider figure 1.1. This is a typical example of a report concerning a medical consultation, **after** being processed by a professional transcriptionist. Obviously, such reports arise in great quantity, since the number of medical consultations required in modern health care systems is enormous. This fortifies the wish for increasing automation.

One solution is the move towards structured data entry, and there is a clear tendency towards that approach. However, dictation will probably remain ubiquitous for years to come, if only for the reason that certain special cases simply cannot be squeezed into a rigid form.

[1] Homophones are words that are pronounced the same way but differ in meaning.

CHIEF COMPLAINT

Dehydration, weakness and diarrhea.

HISTORY OF PRESENT ILLNESS

Mr. Wilson is a 81-year-old Caucasian gentleman who came in here with fever and persistent diarrhea. He was sent to the emergency department by his primary care physician due to him being dehydrated.
...

PHYSICAL EXAMINATION

GENERAL: He is alert and oriented times three, not in acute distress.

VITAL SIGNS: Stable.

...

DIAGNOSIS

1. Chronic diarrhea with dehydration. He also has hypokalemia.
2. Thromboctopenia, probably due to liver cirrhosis.

...

PLAN AND DISCUSSION

The plan was discussed with the patient in detail. Will transfer him to a nursing facility for further care.
...

Figure 1.1: A typical medical report

```
complaint dehydration weakness and diarrhea full stop Mr. Will
Shawn is a 81-year-old cold Asian gentleman who came in with
fever and Persian diaper was sent to the emergency department
by his primary care physician due him being dehydrated period
... neck physical exam general alert and oriented times three
known acute distress vital signs are stable ... diagnosis is one
chronic diarrhea with hydration he also has hypokalemia neck
number thromboctopenia probably duty liver cirrhosis ... a plan
was discussed with patient in detail will transfer him to a nurse
and facility for further care ... end of dictation
```

Figure 1.2: Raw output of speech recognition

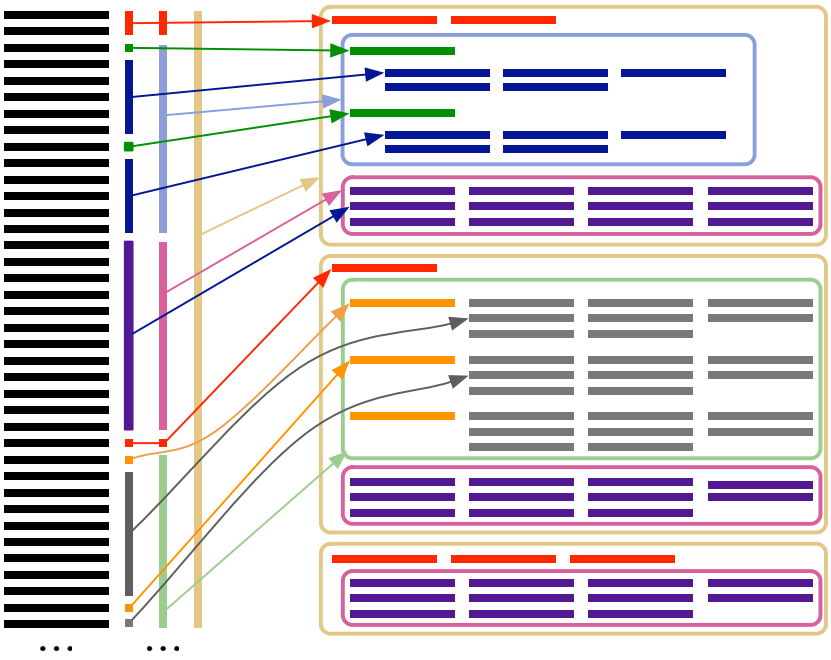

Figure 1.3: Successive multi-level segmentation of a list of tokens

Another possible solution is increasingly intelligent processing of natural language data via statistical methods. This is the approach pursued in this thesis.

Figure 1.2 shows possible output of ASR software for the dictation that eventually ended up as the report in figure 1.1. It should be noted that most ASR software is nowadays capable of detecting sentence boundaries and automatically inserting punctuation (with a varying degree of success), so actual output may in practice be slightly more structured than depicted in 1.2. However, since sentence boundary detection can easily be performed within the holistic framework presented in this thesis, a worst case scenario will be assumed for the available input – a mere sequence of words, that is. In a way, the methods outlined in this document can be considered a natural generalization of already available or emerging features, such as automatic punctuation.

It is fairly obvious that a fair amount of effort is required to transform the input (figure 1.2) into the desired output (figure 1.1). From a schematic point of view, words need to be grouped into sentences, sentences into paragraphs, etc. We can proceed this way for a while, building ever coarser units, until a report is segmented into top-level sections (actually, the report itself may be considered the coarsest unit). Figure 1.3 depicts this process. It can be thought of as *extrapolating structure* out of a sequence. In principle, this is the task that is solved by the mechanisms presented in this thesis.

As mentioned above, identifying underlying structure in a dictated report is only the first, albeit quite intricate, step. Depending on the quality of a dictation, certain transformations may be required (such as re-ordering of sections, insertion of headings, etc.). However, this is outside the scope of this thesis and remains an interesting topic for future work. This thesis aims to provide the *basis* for aforementioned transformations. Once it is known which elements occur in a dictation, and how they relate to each other, operating on the data will be much easier.

What is the benefit of such analysis and transformation of report dictations? By now, one aspect should be fairly clear: one of the goals of the work performed in the context of this thesis is to ultimately free professional typists from the needless burden of repetitive tasks. From a less humanistic point of view, further automation of report transcription increases throughput and allows for cost cutting.

However, there is also another aspect that has not been highlighted thus far. There is hope that the error rate of speech recognition can be improved by identifying sections of limited scope that occur frequently in reports. Some sections in medical reports, for instance, are quite limited with regards to the vocabulary. Examples are sections that typically only contain medication lists and sections about laboratory data[2]. In the end, this is quite similar to the first goal, in that it will ultimately reduce cost or further assist human beings, depending on the point of view.

1.2 Related Work

This topic of this thesis is closely related to the field of *linear text segmentation*. The goal of linear text segmentation is – as its name implies – to partition text into coherent blocks. Now, this is different from the work performed in this thesis in that segmentation is only performed on a single level (thus the name *linear*), whereas this thesis aims to find deeper structure. Indeed, our task may be viewed as a generalization of linear text segmentation.

The recognition of such deep structure has various merits. First, it allows for explicit representation of report structure, including fine-grained elements such as headings and enumerations. This information is required for automatic formatting of reports, for instance. The

[2] A possible strategy might be to first perform automatic speech recognition using a broad coverage language model, then segment the text using the methods presented in this thesis, and finally re-invoke speech recognition using specific language models for the different sections that could be identified. In fact, there is ongoing research that investigates this option [Pro07].

explicit representation also greatly facilitates further processing, such as transformation of report transcripts according to formal and informal requirements of the domain, or rephrasing of utterances to better suit a written style.

Segmenting report dictations into adjacent segments on a single level is one step up from the original representation as a mere sequence of words, but it is not quite sufficient for our scenario. Still, many insights gained in the research area of linear text segmentation also apply to this thesis. In particular, a solution to our structure recognition task could be approached by repeatedly performing linear text segmentation, thereby iteratively splitting segments into more fine-grained items. Section 2.1 demonstrates why this simple approach is not such a good idea after all.

A meanwhile classic approach towards domain-independent linear text segmentation is presented by Choi in [Cho00]. His algorithm *C99* is the baseline which many current algorithms are compared to. Choi's algorithm surpasses previous work by Hearst, who invented the popular *Texttiling* algorithm ([Hea97]). The best results that have been published to date are – to the best of the author's knowledge – those of Lamprier et al. ([LALS07]).

With regards to domain-specific text segmentation, the thesis of Matsuov ([Mat03]) should be noted. In his thesis, Matsuov presents a dynamic programming algorithm capable of segmenting medical reports into adjacent sections. Matsuov's work is similar to this thesis in that he applies his algorithms to medical reports. Furthermore, the task of identifying the sections of a report can be considered a subtask of the problem solved in this thesis. Matsuov also attempts to automatically assign a topic to each section in a report; this is common to both theses. Our goal is different in that Matsuov only performs linear segmentation of report transcripts and is not concerned with more fine-grained elements. Furthermore, the underlying machinery differs from the approach taken in Matusov's thesis.

The automatic detection of section and subsection *types* or *topics* is an important part of our thesis. First, these types provide important clues for further processing. For instance, appropriate headings can easily be derived from the section type. Information extraction tasks are also greatly simplified if the topic of sections and subsections is known beforehand; at the very least, the types show which sections need not be considered. Second, topic detection techniques also provide valuable input for segmentation: If a change of topic is detected, this indicates that a section boundary may have to be introduced.

Topic detection is usually performed using methods similar to those of *text classification* or *text categorization*). Automatic text categorization can be seen as the task of assigning a number of category labels to each document of a given document collection. Sebastiani ([Seb99])

more formally defines the problem of automatic text categorization as that of determining an assignment of a value from $\{0, 1\}$ to each entry of a decision matrix

	d_1	...	d_j	...	d_n
c_1	a_{11}	...	a_{1j}	...	a_{1n}
...	...	...	...	...	...
c_i	a_{i1}	...	a_{ij}	...	a_{in}
...	...	...	...	...	...
c_m	a_{m1}	...	a_{mj}	...	a_{mn}

where $C = \{c_1, ..., c_m\}$ is a set of pre-defined *categories* and $D = \{d_1, ..., d_n\}$ is a set of documents to be categorized. A non-zero value of a_{ij} then indicates that category c_i has been assigned to document d_j.

Numerous solutions to the problem of automatic text classification have been proposed. While the dominant approach towards automatic text categorization used to be that of manual knowledge engineering, this labor-intensive strategy has mostly been superseded by inductive machine learning algorithms since the early to mid-90ies.

Some examples of rather early inductive learning algorithms for text categorization include a mechanism presented by Apté and Damerau ([ADW94]), which employs inductive decision rule learning, as well as the often-cited *RIPPER* algorithm by Cohen and Singer ([CS96]), which can be used to learn simple hypotheses in the form of a disjunction of conjunctions.

Clearly, these algorithms were motivated by earlier, manually constructed expert systems such as *CONSTRUE* ([HANS90]). Meanwhile, with the arrival of thorough understanding of computational learning theory, research seems to be shifting towards application and construction of mathematically well-founded algorithms such as Support Vector Machines (see [Joa98] for an extensive introduction).

This thesis uses transcripts of medical report dictations in order to demonstrate our approach and to assess the efficacy of the presented algorithms. Natural language processing of free text from the medical domain has a long tradition and uses ideas and tools similar to those applied in this thesis. Originally, such processing was usually performed for the sake of automatic information extraction. For instance, an article discussing automated analysis of discharge summaries was published by Gabrieli and Speth as early as 1986 ([GS86]).

The increasing complexity and matureness of medical nomenclatures such as SNOMED also intensified the need for automatic coding of free text documents. To hospitals, such automatic

coding could result in significant cost cutting. Unsurprisingly, this problem is treated extensively in current literature; e.g., Moore and Berman present an automatic SNOMED coding algorithm for pathology reports in [MB94].

Upon further inspection, such information extraction problems are indeed closely related to text segmentation. Both tasks may be approached using similar machinery, since both of them can be represented as a tagging problem. Using that representation, labels which mark concept boundaries or segment boundaries, respectively, have to be assigned to the sequence of tokens that constitutes the text.

As we shall see in section 2.2, the task of finding deep structure in report transcripts can also be represented as a tagging problem. A great number of theoretical frameworks is available for solving such tagging problems; most of them are statistical in nature.

Hidden Markov Models (HMMs) (see [Rab89]) are a mature and efficient framework that has been used ubiquitously for tagging tasks. HMMs have been applied to tasks as diverse as Part-of-Speech (POS) tagging ([Bra00]), text segmentation and topic tracking on broadcast news ([MvG+98]) and emotion recognition ([WVA07]). HMMs are *generative* models, which is a natural representation for many original applications, such as modeling of signal sources. On the other hand, many tagging tasks are better viewed as classification problems. *Discriminative* models can be applied successfully in such cases.

Conditional Random Fields (CRFs) are one particular probabilistic tagging framework that takes a discriminative approach. CRFs were introduced by Lafferty et al. in 2001 (see [LMP01]) and have quickly gained in popularity. They are the main formalism employed in this thesis. Section 2.3 gives further insight regarding the choice of dicriminative versus generative models and motivates why CRFs lend themselves naturally to the structure recognition approach pursued in this thesis.

Major parts of this thesis are dedicated to CRFs. First, a theoretical introduction to this formalism is given in chapter 4. Second, CRFs form the basis of the practical structure recognition implementation that was used for the empirical experiments presented in chapter 6. Therefore, most literature about CRFs is in one or another way relevant to parts of this thesis. The most comprehensive introduction to CRFs available to date is the highly recommended tutorial by Sutton ([SM06]). This tutorial is more recent than the original article by Lafferty et al. and covers current trends.

While not strictly related to the topic of this thesis, the work of Ye and Viola ([YV04]) bears interesting similarities. Ye and Viola apply CRFs to parsing of hierarchical lists and outlines in handwritten notes. Naturally, the features and hints considered for this task are different,

but the goal of finding deep structure and the probabilistic framework are common to both approaches.

CRFs and HMMs are not the only feasible statistical segmentation frameworks. Numerous special purpose algorithms exist. For instance, McDonald et al. present a new model of segmentation based on ideas from multilabel classification ([MCP05]). Their approach allows for natural handling of overlapping or non-contiguous segments and may be an interesting alternative to the CRF-based approach pursued in this thesis.

Finally, it should be noted that options other than statistical analysis of data exist. For instance, Semecký and Zvárová apply regular grammars in order to gather structured data from free-text medical reports; they use a cardiology knowledge base for building the rules of the grammar (see [SZ02]).

Chapter 2

Approach

> *“Difficulties increase the nearer we approach the goal.”*
>
> *– Johann Wolfgang von Goethe*

The structure identification task that was sketched in the introduction leaves many degrees of freedom. At first, it might seem that a myriad of different approaches may have to be considered. Indeed, various different solutions to the problem are conceivable and have been applied successfully to similar tasks.

In this chapter, the problem description will be narrowed. First, a catalog of requirements will be established that has to be met by any feasible approach. In particular, the structural dependencies that are inherent to such a task will be analyzed. Subsequently, the problem representation will be formalized in such a way that these dependencies can be fulfilled.

This still leaves us with a number of options. We will argue that CRFs, a probabilistic framework for segmenting and labeling sequence data, are a natural, if not the best choice for the task at hand. This is illustrated by briefly comparing the relevant properties to those of other similar models, such as HMMs.

Finally, an outline of the approach pursued in this thesis is presented. The outline not only lists all major steps that must be performed, but also serves to give an overview of the remaining chapters in this document.

2.1 Analysis of Requirements

Many feasible approaches towards the identification of structure in report transcripts exist. In section 1.1, a possible strategy was already outlined: Tokens can be grouped into sentences, sentences into paragraphs, and so on. This corresponds to an *iterative* approach, and one can envision at least two variants of this procedure:

- The **bottom-up approach**: Start at the smallest unit of interest (tokens, in our case) and group the items into coarser units. The resulting, coarser items, can in turn be grouped into even coarser units. Iterate this step until the coarsest unit of interest is reached (the whole report, for instance).
- The **top-down approach**: Start at the coarsest unit of interest (e.g., a whole report), and segment it into more fine-grained units. The resulting items can in turn be segmented into even more fine-grained units. This step can be iterated until the smallest unit of interest is reached (tokens, in our case).

Which of these two approaches should be preferred? We will argue that neither of these two variants satisfy our requirements. The problem is one that is quite typical and often encountered in Natural Language Processing (NLP) applications. Such iterative approaches fail to properly incorporate existing *interdependencies*. At each level of segmentation, there is separate knowledge about where a boundary should be introduced, and therefore, it must be possible to provide feedback in *both* directions.

As an example, every sentence boundary strongly influences the possible points for section boundaries (a section boundary will probably not lie in the midst of a sentence), but the opposite is also true: If we have strong knowledge that a section boundary lies at a certain point, this indicates that a sentence boundary needs to be introduced.

If these dependencies are only tracked in one direction, errors will accumulate at each level. A similar situation typically arises in phrase chunking. As a first subtask, a POS tagger will assign the best syntactic label to each token. The main task will then use these labels to segment sentences into phrases. However, if the Part-of-Speech tagger was wrong in the first place, these errors will be carried over. The phrase chunker may have its own idea of where a phrase is likely to start, and it should be able to signal that information back to the Part-of-Speech tagger. This dilemma is well-known and has been studied thoroughly by Sutton and McCallum ([SM05]).

In summary, what is needed is a flexible and non-intrusive mechanism that satisfies at least the following requirements:

- Robustness with regards to noise and variance. Noise is introduced by ASR in the shape of recognition errors. Great variance arises from different speakers, each of whom has his or her own style, different topics (within certain bounds), and various speech-related phenomena. This strongly suggests that a stochastic, corpus-based approach may be appropriate.
- Incorporation of interdependencies between various structural levels. It must be possible for knowledge at one level to propagate to a different level and vice versa.
- Seamless incorporation of various knowledge sources, hints and observations into the decision process. This is mandatory since knowledge about how to best segment a report can come from resources as diverse as word lists, ontologies, grammars and more.

The requirements outlined above suggest that we need to optimize over all possible boundaries on all levels at the same time. Only such a global view allows for proper incorporation of interdependencies between different levels – a requirement that any iterative approach fails to meet.

Some of these requirements may be obvious, and identifying them is not particularly hard. Meeting them can be quite tricky, though. In particular, a formal representation of the structure identification task must be established that is not contradictory to these demands.

2.2 Representing the Problem

In principle, the problem that has to be solved can be considered a segmentation task. One trick that is well-known from chunking and named entity recognition, for instance, is that segmentation problems can be represented as tagging problems. Typically, the so-called `BIO` notation (BEGIN - INSIDE - OUTSIDE) is used. Assigning a `B` label to a token indicates that a segment starts here, whereas `I` indicates that we're inside a segment and `O` indicates that the token is not in any segment of interest.

For our purposes, that notation needs to be adapted slightly: First, since complete segmentation is performed in the scenario of this thesis, there are no `O` labels - any token belongs to a particular segment. Second, we also want to assign *types* to certain segments: the motivation behind this is to not only find the section boundaries in a report, for instance, but to also encode what kinds of sections are present. For this purpose, labels such as $\texttt{B-T}_\texttt{i}$ and $\texttt{I-T}_\texttt{i}$ can be

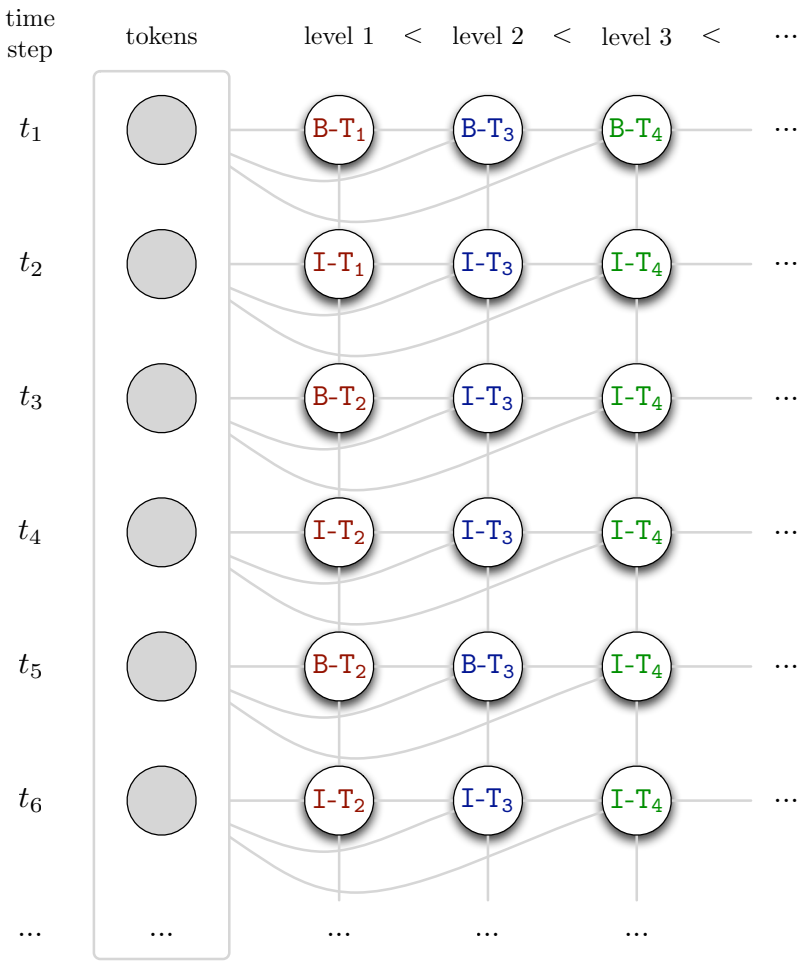

Figure 2.1: Multi-level segmentation represented as a tagging problem

used to indicate that a segment of type `T_i` begins at a certain token, or that a token lies within a segment of type `T_i`, respectively.

Furthermore, since segmentation is to be performed on multiple levels, multiple *label chains* are required. Figure 2.1 illustrates this representation. By adding an additional label chain, it is possible to group the segments of the previous chain into coarser units. Tree-like[1] structures of unlimited depth can be expressed this way. Notice that a tree-like structure is only induced if any segment on a higher level fully contains one or more segments of the adjacent lower level. This means that no `B` label on a higher level may lie between any two `B` labels on a lower level.

The gray lines in figure 2.1 denote dependencies between connected nodes. The horizontal lines indicate that it must be possible for knowledge to propagate between different levels, whereas the vertical lines indicate that any node label depends on the label of its successor node within the same vertical chain. Finally, node labels also depend on the input sequence, which is a sequence of tokens in our case. Ideally, it should be possible for any node to inspect the whole input in order to decide on its label. Typically, only the closer context will be inspected; however, some knowledge sources may require inspection of a wider token window.

A formalism is then needed that is able to properly assign labels to the nodes in figure 2.1, ideally by estimating a stochastic model from a training corpus, which can in turn be applied to new, unknown data. We demand that the requirements identified in section 2.1 must be satisfied by such a formalism.

2.3 On the Choice of CRFs

As it turns out, Conditional Random Fields (CRFs), a relatively recent formalism that was first introduced by Lafferty et al. in 2001 ([LMP01]), are perfectly suitable for this type of problem. Lafferty first applied CRFs to tagging of a single, linear chain. That representation was later generalized in order to allow for more flexible structures by McCallum and Sutton ([SRM04]). An in-depth introduction to CRFs, most notably the variant which will be used in this thesis, namely Factorial Conditional Random Fields (FCRFs), will be given in chapter 4.

[1] In practice, the last chain will still contain multiple segments, since it's redundant to model a "top-level" chain with only one segment. From a theoretic point of view, the data structure is therefore a *hedge* rather than a *tree*. Hedges will be introduced properly in chapter 3.

What makes CRFs so compelling for the purpose of this thesis is that they can directly solve complex multi-chain tagging problems (as presented in figure 2.1), while allowing for natural incorporation of relevant features and dependencies.

This is mostly due to CRFs being discriminative models that describe a conditional probability distribution $p(\boldsymbol{y}|\boldsymbol{x})$ over all possible sequence labelings $\boldsymbol{y}$ given a fixed, arbitrary observation $\boldsymbol{x}$. In contrast, generative models such as HMMs ([Rab89]) describe a joint probability distribution $p(\boldsymbol{y}, \boldsymbol{x})$, which means that the observations $\boldsymbol{x}$ must also be modeled. For many tagging tasks, this is less natural and intuitive than the discriminative view. Furthermore, it seriously restricts the use of $\boldsymbol{x}$ in model features: each observation $x_t \in \boldsymbol{x}$ only depends on the current state in an HMM. In CRFs, all elements of $\boldsymbol{x}$ can be accessed as a feature at any time step of the sequence without additional computational cost.

Both CRFs and HMMs exist in generalized variants that can incorporate dependencies in multi-chain tagging (see [JGJS99] for several generalizations of HMMs, for instance), so the flexible model structure is not an inherent advantage of CRFs. The flexible feature support, on the other hand, is exclusive to discriminative models. It is safe to assume that this is advantageous for the task at hand – in particular for modeling segment topics/types.

The multilabel classification-based segmentation framework by McDonald et al. ([MCP05]) which was mentioned in section 1.2 also might have been an option; however, it lacks the considerable momentum that CRFs have gained in recent years. Meanwhile, there are significant theoretical and empirical results for CRFs. This is an important advantage that should not be disregarded.

In short, CRFs were chosen because of their great flexibility, and because they combine both state-of-the-art performance and sufficient matureness. Furthermore, they can be directly applied to the problem representation established in the previous section. However, it should be expected that comparable results can also be obtained using other problem representations and other probabilistic frameworks.

Regardless of the concrete formalism or method, any machine learning approach requires a substantial amount of training data. Naturally, this also applies to CRFs. Since this thesis aims to provide automatic recognition of structure in medical reports, a large number of such reports need to be available so that a CRF training routine can be used to estimate model parameters.

2.4 Outline

Now that the task has been formalized and all pieces are in place, the approach pursued in this thesis can be outlined as follows:

- First, the available data (medical reports) must be sighted and analyzed. A set of labels and label chains must be determined that allows for modeling of the structure typically found in the available reports. A training corpus consisting of reports that are annotated using the aforementioned set of labels must then be compiled. The course of this work is described in chapter 3.
- Second, actual code for training of CRFs (parameter estimation) and for labeling new data using CRFs must be written or re-used. The software should implement the theoretical framework presented in chapter 4. An overview of the implementation is given in chapter 5.
- Finally, experiments must be conducted that prove the practicability of the presented approach. The experimental setup and the corresponding results are presented in chapter 6.

The thesis is then concluded by giving an outlook on future activities (chapter 7) and a summary of the presented work (chapter 8).

Chapter 3

Data Preparation

> *“ Mathematics may be compared to a mill of exquisite workmanship which grinds your stuff to any degree of fineness; but, nevertheless, what you get out depends on what you put in; and as the grandest mill in the world will not extract wheat flour from peascods, so pages of formulae will not get a definite result out of loose data. ”*
>
> *– Thomas Henry Huxley*

As with any corpus-based approach, the available data plays a major role in this thesis. This chapter aims to give an overview of the available data and its characteristics.

The data consists of a large number of medical reports. These reports will be analyzed regarding their structure. A formal model describing the structure and contents of any well-formed medical report of the corpus will be presented. From that model, a suitable set of labels will be derived that forms the basis for representing reports as a CRF instance.

Data cleansing and corpus annotation will be described in detail. This is a particularly tricky issue: Manual annotation is infeasible due to the sheer size of the corpus. Therefore, a semi-automatic approach based on parallel corpora is presented.

Finally, feature generation will be described in this chapter. This is a key topic since CRFs allow for incorporation of arbitrary knowledge sources. The available resources will be presented, and it will be shown how this knowledge can find its way into CRF features.

While this chapter is tied to the available data (medical reports, that is), it should be possible to pursue a similar approach for any kind of structured documents. The labels used for annotation will differ, but the process should remain the same.

3.1 Available Corpora

For the purpose of this thesis, two parallel corpora consisting of 2,007 manually corrected and formatted reports and the corresponding raw output of ASR, respectively, were compiled. All reports are concerned with medical consultations and were dictated by physicians.

The first part of the corpus will be referred to as C_{COR}. These documents have all been edited by professional typists and are formatted as presented in figure 1.1. The second part of the corpus, which will be called C_{RCG}, consists of the raw output of ASR which was used by the typists to produce the properly formatted reports in C_{COR}. This means, in essence, that every report is available in two forms: a properly formatted and manually corrected version, and an unformatted, error-ridden original version.

Now, the goal of this thesis is to allow for structured output of speech recognition. Since this process should work in an automatic fashion, such a mechanism will be restricted to using whatever speech recognition provides. This may raise the question of what use the documents in C_{COR} are. Actually, these reports play an important role during *training*: In order to train a statistical model that is suitable for identifying structure in raw, unformatted documents, annotation is required which explicitly marks the various structural elements. Obviously, it would be a daunting task to manually annotate 2,000 unformatted reports with regards to their underlying structure.

This is where C_{COR} comes into play. Since the documents in C_{COR} are formatted quite consistently, it is possible to automatically parse their structure. The idea is then to find some way of mapping the structure identified in the documents of C_{COR} onto the corresponding documents of C_{RCG}. This can certainly be done – to a varying degree of accuracy –, since any two parallel reports are in general quite similar. However, the task is still non-trivial, since there may be various discrepancies:

- As mentioned before, documents in C_{RCG} typically contain recognition errors.
- Passages may have been rephrased by a typist to better suit a written style.
- Structural elements that are explicitly marked in C_{COR} may not have been dictated at all. As an example, headings are often introduced by typists to clearly indicate the structure of a document.
- Certain utterances contained in the original dictations, such as instructions directed to the typist, will have been removed and therefore cannot be found in C_{COR}.

The above list is by no means exhaustive. Rather, it serves to give an idea of the obstacles that are to be expected when trying to automatically annotate reports of C_{RCG} using the parsed structure of their counterparts in C_{COR}.

3.2 Required Annotation

One key question, and indeed the first problem to tackle, is the kind of annotation that is required in order to properly capture the structure of all reports contained in the corpus. A set of labels must be determined that allows for encoding of all relevant aspects. This can be achieved by analyzing the reports in C_{COR} with regards to how they were structured by the transcriptionist.

3.2.1 Analysis of Report Structure

As has been mentioned already, the structure of reports in C_{COR} is readily visible due to markup, indenting, etc. It is – for a human reader, anyway – easily possible to identify the various elements a report consists of. A first quick look at figure 1.1 reveals at least the following structural elements:

Headings: These are indicated via a bold font and capital letters (`HEADING`). Conceptually, headings introduce the coarsest structural unit of a report, which we'll call **sections**.

Subheadings: Like top-level headings, these are shown in all capital letters and a bold font, followed by a colon (`GENERAL:`). They are usually indented and introduce a following **subsection**.

Enumerations: Enumerations are usually separated from the rest of the text by one or more blank lines. They contain a number of **enumeration items**, which usually start by a number, followed by a period, and are horizontally indented using one or more blanks.

These are, however, only the most obvious elements. It is possible to go much further than that. Undoubtedly, a medical report also consists of **paragraphs**. We can descent even further, dividing paragraphs into (possibly multiple) **sentences**.

If a medical report is hierarchically dissected into the elements identified above, the outcome is a hierarchic data structure comprising all elements of the report. Compared to the original representation as a plain file, the function of each element is expressed explicitly. A formal

model which describes the structure and contents of any well-formed report in our corpus can be expressed as a Regular Hedge Grammar (RHG) and will be introduced later in this section.

What does this mean in practice? Such a formal model describes which structural elements may be contained in a document, and where these may occur. Obviously, a medical report may contain elements such as headings, sections, subsections, enumerations, etc. A more delicate question is how these elements are related to each other. May an enumeration occur in a subsection? How many paragraphs can be contained in a section? Where may a heading be placed in a medical report? It quickly becomes evident that these questions can best be solved by providing a formal description of the model. In order to do so, some theory is required, which will be treated next.

3.2.2 Formal Description of Report Structure

Hedge Theory

Informally, a hedge can be considered a sequence of trees. It is important to note that a hedge is unequal to a forest: forests are unordered sets of trees. We will adopt the formal definition of Murata [Mur00] for the purpose of this thesis:

A ***hedge*** over a finite set Σ of symbols and a finite set X of variables is:

- $\Rightarrow$ ε , *(null hedge)*
- $\Rightarrow$ x, where x is a variable in X, *(variable data)*
- $\Rightarrow$ $a\langle u\rangle$, where a is a symbol in Σ and u is a hedge, or *(addition of a symbol as root node)*
- $\Rightarrow$ uv, where u and v are hedges. *(concatenation of two hedges)*

An example of a hedge is $heading\langle token\langle\varepsilon\rangle\ token\langle\varepsilon\rangle\rangle paragraph\langle sentence\langle token\langle\varepsilon\rangle\ token\langle\varepsilon\rangle\rangle\rangle$. It is depicted in figure 3.1. As the figure demonstrates, a hedge is not in general a tree. However, a hedge can be a tree if there is only one root node.

The set of variables X deserves some explanation: it is a special property of hedges which can be traced back to the history of hedges as a formal description language for XML documents. There, X can for instance be used to describe elements containing character data by adding #PCDATA to X. Our model of report structure currently does not use character data, hence X will be empty. Now that we can describe single hedges, we would like to have some mechanism to describe classes of valid hedges (i.e., all valid reports). Regular Hedge Grammars (RHGs) come to the rescue. Again, the definition of Murata will be used:

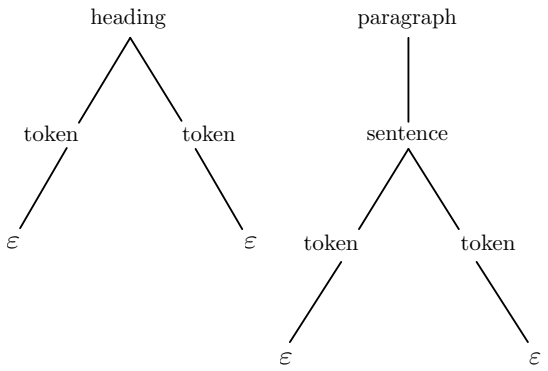

Figure 3.1: A hedge

A ***Regular Hedge Grammar (RHG)*** is a 5-tuple $\langle \Sigma, X, N, P, r_f \rangle$, where Σ is a finite set of symbols, X is a finite set of variables, N is a finite set of non-terminals, r_f is a regular expression comprising non-terminals and P is a finite set of production rules, each of which takes one of the two forms given below:

- $\Rightarrow$ $n \rightarrow x$, where n is a non-terminal in N, and x is a variable in X,
- $\Rightarrow$ $n \rightarrow a\langle r \rangle$, where n is a non-terminal in N, a is a symbol in Σ, and r is a regular expression comprising non-terminals.

We can then go on to define a RHG that describes the permissible structure of reports in C_{COR}.

A RHG for Reports in $\mathrm{C_{COR}}$

The RHG shown in figure 3.2 was determined by looking over all reports in C_{COR} and identifying their structural elements. It should cover most of what is needed for properly arranging a typical medical report.

The regular expression syntax used in figure 3.2 follows the usual conventions, where "$\star$" stands for *zero or more occurrences*, "+" stands for *at least one occurrence*, and "|" is used to express *alternation*. Parentheses are used to disambiguate operator precedence. In order to make the grammar more comprehensible, non-terminals are always capitalized, whereas terminal symbols consistently start with a lower-case letter.

Please note that this grammar only holds for corrected, properly arranged reports as contained in C_{COR}. In particular, most raw dictations in C_{RCG} will not strictly satisfy all constraints imposed by the grammar. For instance, it is quite common that headings or enumeration

$$REP = \langle \Sigma, X, N, P, Section+ \rangle$$

$$\Sigma = \{section, heading, subsection, subheading, enumeration, enumelement, enummarker, paragraph, sentence, token\}$$

$$N = \{Section, Heading, Subsection, Subheading, Enumeration, Enumelement, Enummarker, Paragraph, Sentence, Token\}$$

$$X = \{\}$$

$$\begin{aligned} P = \{ & Section \rightarrow section\langle Heading(Subsection|Enumeration|Paragraph)+\rangle, \\ & Heading \rightarrow heading\langle Token+\rangle, \\ & Subheading \rightarrow subheading\langle Token+\rangle, \\ & Subsection \rightarrow subsection\langle Subheading\, Paragraph\rangle, \\ & Enumeration \rightarrow enumeration\langle Enumelement+\rangle, \\ & Enumelement \rightarrow enumelement\langle Enummarker\, Paragraph+\rangle, \\ & Enummarker \rightarrow enummarker\langle Token\rangle, \\ & Paragraph \rightarrow paragraph\langle Sentence+\rangle, \\ & Sentence \rightarrow sentence\langle Token+\rangle, \\ & Token \rightarrow token\langle \varepsilon\rangle, \end{aligned}$$

Figure 3.2: A RHG describing permissible report structure

markers are not explicitly dictated. Still, the grammar gives a good overview of what can be expected.

Finally, elements (or nodes) of RHGs are typically assigned *attributes*. This is not strictly part of the formalism; however, it is common practice. In our case, it will be useful to assign a *type* attribute to *section* and *subsection* elements. There is only a limited number of types of sections and subsections that are used in medical reports over and over again, and it may be helpful for further processing to know the type of a segment.

3.2.3 Determining Section and Subsection Types

Section and subsection types are certainly dependent on the report domain. For the medical domain, attempts have been made to standardize the contents of reports in the past by defining a set of sections and subsections that are typically needed in such reports and which practitioners should adhere to. An example of such a standard is the "Standard Specification for Healthcare Document Formats", issued by the American Society for Testing and

Materials (ASTM) International under designation number E2184-02 ([Int02]). This specification addresses requirements for the headings, arrangement and appearance of sections and subsections when used in healthcare documents.

However, it should be noted that in spite of the existence of such clear recommendations, actual medical reports still vary greatly with regard to their structure, depending on the dictating practitioner and in-house standards of healthcare providers. The section and subsection types used for the purpose of this thesis were thus identified as follows:

- As an initial set, the section and subsection types listed in E2184-02 were adopted.
- The headings and their respective variants given for each section and subsection type in E2184-2 were used as a starting point for assigning a type to each report segment occurring in C_{COR}.
- If a new heading was found in the corpus that had not been seen so far, the first step was to try to manually assign it to one of the previous section or subsection types; only if this was impossible, a new section or subsection type was added to the previous set.

This process was supported by a script that automatically assigned a type to report segments if their heading was already known. In order to keep the amount of manual work as low as possible, the assumption had to be made that segments with the same heading are of the same type. This may not be true in all cases, but for the vast majority this premise seems to hold. In the end, manually created heading clusters were available that could be used to identify the type of a report segment.

3.2.4 From Hedge to Label Chains

The formal model of report structure (see figure 3.2), along with the set of section and subsection types identified above, provides all information that is required in order to determine a set of labels for annotating reports in C_{COR} and C_{RCG}.

As outlined in the previous chapter, tree-like structures of finite depth can easily be represented via multiple *label chains* containing typed BEGIN and INSIDE labels (figure 2.1). Since the RHG depicted in figure 3.2 is non-recursive, all productions are of limited depth. Each structural element described by the RHG could easily be assigned to one of three *levels* or label chains:

- **Sentence level:** This term will be used to denote the first label chain, which describes the boundaries of sentences, headings, subheadings, and such.

- **Subsection level:** This is the label chain describing boundaries of typed subsections, paragraphs and enumerations. Obviously, each segment in this label chain spans one or more segments at the sentence level.
- **Section level:** The topmost label chain is used to encode boundaries of typed sections. Each section spans one or more subsections, paragraphs, etc., at the subsection level.

For untyped segments such as headings, subheadings, sentences and paragraphs, the identifiers introduced in figure 3.2 were used. For each of these structural elements, one BEGIN label (e.g.: `BeginHeading`) and one INSIDE label (e.g.: `Heading`) were used. One notable exception is `Enummarker`, which does not have a corresponding BEGIN label because it never spans multiple tokens.

For typed segments (sections and subsections), a separate (BEGIN, INSIDE) label pair was introduced for each type (e.g.: `BeginDiagnosis` and `Diagnosis`). Figure 3.1 gives the annotation labels used for each level, excluding the BEGIN variants. The `None` label has a special purpose: If there is a `Heading` segment at the sentence level, there isn't any segment at the subsection level that might contain it; yet, *some* label must be assigned in this formalism, therefore `None` labels are used to encode this situation.

Finally, it should be noted that the division into the three label chains listed above (and the corresponding labels) is somewhat arbitrary. Other divisions (and labels) are conceivable that might be just as useful. However, most plausible representations are probably rather close to the one presented in this thesis, simply because it is quite natural.

3.3 Semi-Automatic Label Annotation

Now that it is clear what kind of annotation is required, the next question is how C_{COR} and C_{RCG} can be annotated without unreasonable manual effort.

The approach presented in this section follows the following outline:

- In the first step, the fact that reports in C_{COR} are consistently formatted and arranged is exploited. Each report is parsed with regards to its underlying structure, and the resulting hedge is then used to annotate the report using three label chains.
- The second step consists of mapping the annotation of each report in C_{COR} onto the corresponding report in C_{RCG}. This involves a smart alignment process that draws on domain knowledge and phonetic similarity.

Sentence Level	Subsection Level	Section Level
Heading	Abdomen	AdvanceDirectives
Subheading	AnoRectal	Allergies
Sentence	Back	ChiefComplaints
Enummarker	Brain	Course
	Breasts	Diagnosis
	Cardiac	DiagnosisAndPlan
	Cardiovascular	DiagnosticStudies
	Coordination	Findings
	CranialNerves	Habits
	Ears	HistoryOfPresentIllness
	Endocrine	Medication
	Enumelement	Neurologic
	Extremities	Notes
	Eyes	PastHistory
	Gastrointestinal	PastSurgicalHistory
	General	PhysicalExamination
	Genitourinary	Plan
	HEENT	Practitioner
	Head	Procedures
	Hematologic	Prognosis
	Immunologic	ReasonForEncounter
	LymphNodes	ReviewOfSystems
	MentalStatus	Time
	Motor	
	MouthAndThroat	
	Musculoskeletal	
	Neck	
	Neurologic	
	None	
	NoseAndSinuses	
	Paragraph	
	Pelvic	
	PeripheralVascular	
	Psychiatric	
	Reflexes	
	Respiratory	
	Sensory	
	Skin	
	Speech	
	Station	
	ThoraxAndLungs	
	VitalSigns	

Table 3.1: Labels used for annotation (excluding “Begin” labels)

3.3.1 Cleansing

To ensure that all reports in C_{COR} could be parsed automatically, some initial cleansing was necessary. This involved manual sighting of all reports.

In particular, proper formatting of section and subsection headings had to be checked, since these are essential clues for subsequent parsing. Furthermore, proper indentation of enumeration items had to be ensured. Finally, some reports contained fragments of instructions to the transcriptionist or other meta markers like "end of dictation". Such passages were removed.

Each report was manually edited until the parser was able to properly determine its structure. Fortunately, only a small percentage of reports required any editing at all.

3.3.2 Parsing of Formatted Reports

The structure of reports in C_{COR} could be parsed using a parsing module that had been developed in a previous practical course. This process involves the following steps:

- First, domain-specific tokenization and POS tagging is performed. The tokenizer uses a broad-coverage dictionary and a domain-specific grammar that covers most common numerical expressions (dates, physical units, dosages, etc.) of the medical domain. Both resources were compiled into finite state automatons for runtime efficiency. The tokenization component is described in detail in [HJK+06].
- In the second step, phrase chunking is performed. The output of this step is not relevant to this thesis, however, since the syntactical structure of sentences need not be analyzed.
- Finally, report structure is identified via section and subsection headings, linebreaks, enumeration tokens ("1.", "A.", etc.) and other simple heuristics.

The output of the parsing component is a hedge data structure following the rules of the RHG shown in figure 3.2. The Part-of-Speech (POS) information associated with each token is not immediately helpful, but can be used during feature generation (section 3.4).

Annotation in the form of three label chains could then be created from the hedge data structure as described in subsection 3.2.4. Each token of a report corresponds to one time step of the label chains.

3.3.3 Mapping Annotations via Alignment

The remaining step is to map the newly created annotation of reports in C_{COR} onto the corresponding reports in C_{RCG}. For this purpose, an existing smart alignment framework could be used (see [HJK+06]) [1].

The basic idea of this approach is that since any two corresponding reports of C_{COR} and C_{RCG} should be very similar (except for formatting, etc.), the automatically created annotation of reports in C_{COR} should – to a large degree – also apply to those in C_{RCG}. The key is to establish proper alignment between any two corresponding tokens of parallel reports. This process is illustrated in figure 3.3. Note that this figure only shows the first label chain (the sentence level); however, the idea can easily be extended to multiple label chains.

Multiple problems need to be dealt with when pursuing the aforementioned approach:

1. Typically, most tokens occurring in the edited and properly formatted version of a report (C_{COR}) are *somehow* contained in the unedited version (C_{RCG}). However, the token may be formatted differently (e.g., "09/07/2007" vs "September 7, 2007"), there may have been a speech recognition error (e.g., "09/07/2007" vs "09/17/2007"), or a more formal synonym may have been used by the transcriptionist ("acute myocardial infarction" or "AMI" vs "heart attack").
2. Quite often, some utterances of the dictation have been removed from the edited report. Typical examples are meta instructions directed to the transcriptionist, or repetitions by the practitioner. Obviously, it is then impossible to properly align the tokens of the dictation with corresponding ones of the formatted report. Mapping annotation onto the unedited dictation is quite tricky in such situations.
3. Sometimes, part of the formatted report has never been dictated (e.g., because punctuation was inserted by the ASR automatically, or because the document was supplemented at a later point). In this case, it is impossible to properly align the newly introduced passage. However, this is rather unproblematic, since it is obviously not necessary to carry over annotation labels for such parts.

Since point three is essentially unproblematic, two problems remain: In order to solve the problem described in point one, the alignment algorithm needs to be able to perform some kind of "sloppy" matching that takes into account phonetic and/or semantic similarity. Such an alignment algorithm will be described shortly.

[1] The author of this thesis helped develop the alignment framework within the SPARC project (http://www.sparc.or.at).

C_{COR}		**OP**	C_{RCG}	
...	...	...	...	...
BeginHeading	CHIEF	del		
Heading	COMPLAINT	sub	complaint	BeginHeading
BeginSentence	Dehydration	sub	dehydration	BeginSentence
Sentence	,	del		
Sentence	weakness	sub	weakness	Sentence
Sentence	and	sub	and	Sentence
Sentence	diarrhea	sub	diarrhea	Sentence
Sentence	.	sub	fullstop	Sentence
BeginSentence	Mr.	sub	Mr.	BeginSentence
Sentence	Wilson	sub	Will	Sentence
		ins	Shawn	Sentence
Sentence	is	sub	is	Sentence
Sentence	a	sub	a	Sentence
Sentence	81-year-old	sub	81-year-old	Sentence
Sentence	Caucasian	sub	cold	Sentence
Sentence		ins	Asian	Sentence
Sentence	gentleman	sub	gentleman	Sentence
Sentence	who	sub	who	Sentence
Sentence	came	sub	came	Sentence
Sentence	in	del		
Sentence	here	sub	here	Sentence
Sentence	with	sub	with	Sentence
Sentence	fever	sub	fever	Sentence
Sentence	and	sub	and	Sentence
Sentence	persistent	sub	Persian	Sentence
Sentence	diarrhea	sub	diaper	Sentence
Sentence	.	del		
...	...	...	...	...

Figure 3.3: Mapping labels via alignment

Finally, the problem described in point two can really only be solved heuristically. In practice, it proved to be the best strategy to simply assume that any passage that only occurs in the dictation belongs to the last segment for which proper alignment could be established (see figure 3.3). This approach (when enhanced with some exceptions) yields satisfactory results for the label chains of section and subsection levels; however, the segment boundaries on the sentence level may be inaccurate at times, which is due to the fact that this level is the most fine-grained one. This is a concession to the time-saving automatic annotation mechanism that could really only be remedied by manually correcting the annotation.

So far, establishing alignment between the corresponding reports of C_{COR} and C_{RCG} has been presented as an abstract process. In the following, a concrete algorithm will be given that can be used to align arbitrary sequences according to user-defined criteria.

One well-known algorithm that is frequently applied to such problems was published by Levenshtein as early as 1966 (see [Lev66]). In its original form, the algorithm is used to compute the *distance* between two character sequences. The Levenshtein distance is defined as the minimum number of operations needed to turn one of these sequences into the other, where each operation must be one of "insertion" (`ins`), "deletion" (`del`) and "substitution" (`sub`). If the minimal path of operations is captured, this information implicitly defines an alignment of the two input sequences.

Algorithm 1 shows a slight generalization of the Levenshtein algorithm, which allows to specify a user-defined cost function that depends on the operation that is performed and the elements of the sequences which the operation is performed for.

The algorithm returns two matrices, the cost matrix $\boldsymbol{C}$ and the back pointer matrix $\boldsymbol{O}$. Finding an optimal alignment for sequences $\boldsymbol{x}$ and $\boldsymbol{y}$ is equivalent to finding the shortest path from $\boldsymbol{C}[0,0]$ to $\boldsymbol{C}[n,m]$. This is achieved through dynamic programming: Each element $\boldsymbol{C}[i,j]$ is iteratively assigned the minimum cumulated cost of reaching it from $\boldsymbol{C}[0,0]$. Therefore, after the algorithm has terminated, $\boldsymbol{C}[n,m]$ contains the cost of the shortest path from $\boldsymbol{C}[0,0]$ to $\boldsymbol{C}[n,m]$.

The path itself can be reconstructed via $\boldsymbol{O}$. Each element $\boldsymbol{O}[i,j]$ stores the operation that minimized the cost $\boldsymbol{C}[i,j]$. This information can be used as a back pointer: If the operation is `sub`, the previous node of the shortest path was $[i\text{-}1, j\text{-}1]$, whereas `del` and `ins` imply $[i\text{-}1, j]$ and $[i, j\text{-}1]$, respectively. The shortest path can thus be found by following the back pointers from $\boldsymbol{O}[n,m]$ to $\boldsymbol{O}[1,1]$. This is quite similar to the Viterbi algorithm.

The complexity of the algorithm is obviously $O(nm)$. Note that Algorithm 1 is just a sketch. Actual implementations may perform several optimizations; for instance, $\boldsymbol{C}$ need not be kept

Algorithm 1 GENERALIZED LEVENSHTEIN DISTANCE

Given:

- Two arbitrary sequences $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)$;
- A cost function $\texttt{cost}(x_i, y_j, op)$ that measures the cost of of operation $op \in \{\texttt{del}, \texttt{ins}, \texttt{sub}\}$ given element $x_i \in \boldsymbol{x}$ and element $y_j \in \boldsymbol{y}$;

1. $\boldsymbol{C} = [0..n, 0..m]$; *# empty matrix of size n+1 $\times$ m+1*
2. $\boldsymbol{O} = [1..n, 1..m]$;
3. $\boldsymbol{C}(0, 0) := 0$;
4. for $i := 1, 2, \ldots, n$:
   (a) $\boldsymbol{C}[i, 0] := \boldsymbol{C}[i\text{-}1, j] + \texttt{cost}(x_i, \epsilon, \texttt{del})$;
5. for $j := 1, 2, \ldots, m$:
   (a) $\boldsymbol{C}[0, j] := \boldsymbol{C}[0, j\text{-}1] + \texttt{cost}(\epsilon, y_j, \texttt{ins})$;
6. for $i := 1, 2, \ldots, n$:
   for $j := 1, 2, \ldots, m$:
   (a) $\texttt{cc}(op) = \begin{cases} \boldsymbol{C}[i\text{-}1, j] + \texttt{cost}(x_i, y_j, \texttt{del}) & \text{if } op = \texttt{del} \\ \boldsymbol{C}[i, j\text{-}1] + \texttt{cost}(x_i, y_j, \texttt{ins}) & \text{if } op = \texttt{ins} \\ \boldsymbol{C}[i\text{-}1, j\text{-}1] + \texttt{cost}(x_i, y_j, \texttt{sub}) & \text{if } op = \texttt{sub} \end{cases}$
   (b) $\boldsymbol{C}[i, j] := \min\{\texttt{cc}(\texttt{del}), \texttt{cc}(\texttt{ins}), \texttt{cc}(\texttt{sub})\}$;
   (c) $\boldsymbol{O}[i, j] := \underset{op}{\operatorname{argmin}}\{\texttt{cc}(\texttt{del}), \texttt{cc}(\texttt{ins}), \texttt{cc}(\texttt{sub})\}$;
7. return (C, O).

		S	**a**	**t**	**u**	**r**	**d**	**a**	**y**
	0	1	2	3	4	5	6	7	8
S	1	0	1	2	3	4	5	6	7
u	2	1	1	2	2	3	4	5	6
n	3	2	2	2	3	3	4	5	6
d	4	3	3	3	3	4	3	4	5
a	5	4	3	4	4	4	4	3	4
y	6	5	4	4	5	5	5	4	3

(a) Matrix $\boldsymbol{C}$

	S	**a**	**t**	**u**	**r**	**d**	**a**	**y**
S	sub	ins	ins	ins	ins	ins	ins	ins
u	del	sub	ins	sub	ins	ins	ins	ins
n	del	del	sub	ins	sub	ins	ins	ins
d	del	del	del	sub	ins	sub	ins	ins
a	del	sub	ins	del	sub	del	sub	ins
y	del	del	sub	ins	del	del	del	sub

(b) Matrix $\boldsymbol{O}$

Figure 3.4: Dynamic programming matrices after aligning two strings

in memory completely – it is sufficient to store two rows at any time.

Traditionally, a cost function might be defined as follows:

$$\texttt{cost}(x_i, y_j, op) = \begin{cases} 0 & \text{if } op = \texttt{sub} \text{ and } x_i = y_j \\ 1 & \text{otherwise} \end{cases} \tag{3.1}$$

If $\boldsymbol{x}$ and $\boldsymbol{y}$ are two strings, then Algorithm 1 returns the well-known *string edit distance* when invoked with this cost function.

Figure 3.4 shows the resulting matrices $\boldsymbol{C}$ and $\boldsymbol{O}$ if Algorithm 1 is invoked with cost function (3.1) for two strings ("Sunday" and "Saturday")[2].

For the purpose of this thesis, a more sophisticated cost function is required:

- Tokens that are similar (either from a semantic or phonetic point of view) should be assigned low cost for substitution, whereas dissimilar tokens should receive a prohibitively expensive score. It is not practical in this scenario to perform substitution of unrelated tokens, since this would result in these two tokens being aligned – and ultimately, the two tokens would therefore receive the same annotation labels.

[2]This example is adopted from Wikipedia (`http://www.wikipedia.org`).

- The cost for deletion and insertion should behave inversely: If phonetic or semantic similarity between two tokens cannot be established, the cost for insertion and deletion should be low; otherwise, the cost should be prohibitively high.

In practice, the cost function applies a few additional heuristics for dealing with punctuation and other subtle issues; however, measuring phonetic and semantic similarity is of primary importance.

The semantic scoring module described in [HJK+06] could be used to check if two tokens are synonymous or have a similar meaning; this works roughly as follows:

- In the first stage, a check is performed if the tokens are actually identical modulo formatting and way of speaking. For this purpose, a large finite state transducer is used that covers a multitude of different formattings and variants of domain-specific expressions. For instance, "09/01/2007" and "September the first, two thousand and seven" can be identified as being identical using the aforementioned finite state automaton.
- In the second stage, resources of the Unified Medical Language System (UMLS) – see [LHM93] – are used to check the semantic relation between two tokens, if any. An ordinal number is returned that describes the degree of semantic similarity.

For phonetic matching, the Metaphone algorithm ([Phi90]) was used. This algorithm is a slightly improved variant of the well-known Soundex algorithm. It converts an input string into a pseudo-phonetic representation; two words that are pronounced the same way should result in a similar representation when Metaphone is invoked on them.

The cost function used for this thesis first applies the Metaphone algorithm to any two tokens that shall be compared, and then computes the string edit distance between the resulting pseudo-phonetic representations. Tokens that are pronounced equally are thus assigned a distance of zero. This is a reasonable way of identifying possible errors of the speech recognizer.

The actual cost for substituting two tokens is then derived from the degree of semantic similarity and the phonetic distance (which is calculated as described above).

3.4 Feature Generation

The annotation discussed in the previous sections is only one part of a training corpus for a CRF-based approach. The other part are features or observations that need to be provided for

each time step (token) of a report. These observations are expected to indicate as strongly as possible which annotation labels need to be assigned to the time step. The CRF training algorithm (or any other discriminative machine learning approach, for that matter) will try to find out which observations typically occur in conjunction with which annotation labels; this is the basis for later tagging of unseen, unlabeled reports.

Part of the appeal of CRFs is that arbitrary features can be computed for any time step, and that the whole input sequence may be inspected in doing so. However, along with great flexibility also comes the need for great discipline; CRFs make it easy to specify useless features.

The following features were computed for each time step of the reports in C_{COR} and C_{RCG}:

- **N-gram features** covering the close local context of the current time step. These features inspect a window of ± 2 tokens. Typical examples are `patient@0`, `the@-1` and `is@1`, where the second part of the feature (after the @) is the offset, and the first part is the token found at that offset. The reasoning behind these features is that they should be suitable for covering local phenomena such as headings, which consist of a (small) number of subsequent tokens.
- **Syntactic features** inspecting the same context as the **N-gram** features above. In contrast, these features use the possible syntactic categories of a token (as determined using a broad-coverage dictionary) rather than the token string itself. Examples are `NN@0`, `JJ@0`, `DT@-1` and `be+VBZ+aux@1`. These features are introduced to provide some kind of redundancy; they encode indications for families of tokens rather than specific tokens.
- **Bag-of-Words (BOW) features** inspecting the wider context of the current time step. In contrast to the N-gram features, the BOW features do not encode any order of tokens; they merely indicate how often a particular concept occurred within a window of ± 10 tokens. UMLS ([LHM93]) concept IDs are used rather than the actual token strings – these features are intended to capture the topic of a text segment, and for that purpose, different words describing the same concept can be considered equal. As an added bonus, this approach reduces the feature space. If a UMLS concept ID cannot be retrieved, stemming is performed on the token. Examples are `C00028734⟨bow⟩(3)` and `polydipsia⟨bow⟩`. The first part of the feature string specifies the concept ID or word stem, whereas the number in parentheses is a feature value that indicates how often that concept or stem occurred in the inspected window. If no number is given, a feature value of 1.0 is assumed. This corresponds to `TF` term weighting, which is

shown to perform competitively for text categorization in a recent study by Lan et al. ([LSLT05]).

- **Semantic type features** inspecting the same context as the BOW features. These features are similar to the BOW features; however, semantic types of the UMLS semantic type hierarchy are given rather than UMLS concept IDs. The semantic type hierarchy is much coarser than the space of concept IDs, so these features are again used to introduce some kind of redundancy (this is similar to what the syntactic features do for local phenomena, but this time for topic detection). Examples are `A1.4.1.2.1.7⟨bow⟩` and `B2.2.1.2.1⟨bow⟩(2)`.
- **Relative position features** give the relative position of the current time step in the report. The report is divided into eight parts corresponding to eight binary features; only one of these features is non-zero, depending on the part into which the current time step falls. An example of such a feature is `relpos=first_eighth`. These features are encoded as multiple binary features rather than a single feature on a scale from, say, $0.0 - 1.0$, because the latter representation fails to encode a *positive* indication for a certain label at the beginning of a report. The idea behind the relative position features is that they can actively support topic detection, because certain report sections are more likely to occur at the end of a report than in the beginning, for instance.

It may seem peculiar at first that most of these features are – in some way – based on the tokens of the report. Indeed, other application domains may possibly draw on more diverse features; however, in a NLP task, the tokens are really the only input that is available. One notable exception is the position indicator described above, which encodes the relative position of a token in the report. This feature models domain knowledge and is not derived from the token itself.

On a related note, it is certainly possible to use slightly different features for the same task; however, since most features have to be based on the tokens occurring in the report anyway, the options are rather limited and mostly equal from a performance point of view. The only thing that can really be done to improve performance – compared to an approach using only plain token N-grams or bags – is to introduce some kind of redundancy, which is what the syntactic and semantic type features provide, and to model further domain knowledge, as the relative position features do.

Chapter 4

Review of Theory

> *“There is nothing so practical as a good theory.”*
>
> *– Kurt Lewin*

This chapter aims to introduce the theoretic framework that forms the basis of the structure identification approach pursued in this thesis.

The theory of Conditional Random Fields (CRFs) lies at the very heart of this framework. First, a formal definition of general CRFs will be given. That definition will then be concretized for a special family of CRFs, so-called Factorial Conditional Random Fields (FCRFs). FCRFs are particularly suitable for modeling document structure using multiple label chains.

Algorithms will be introduced that can be used for performing inference in a CRF. Inference is required in order to predict the most likely outcome of the random variables in a CRF given a set of parameters.

The parameters of a CRF are typically estimated from training data. This chapter will present common methods of parameter estimation. All of these methods involve optimization of an objective function. Several different objective functions are conceivable; the most frequently used variant requires running inference for each instance of the training data at each step of the optimization algorithm. This emphasizes the central role of efficient inference algorithms.

Finally, it should be noted that the notation used in this chapter is similar to that used by Sutton and McCallum in [SM06].

4.1 Introduction to Conditional Random Fields

Let G be an undirected model over a set of random variables $\boldsymbol{y}$ and a fixed, observed entity $\boldsymbol{x}$. A CRF is then a conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$, where:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \prod_{\Psi_A \in G} \Psi_A(\boldsymbol{y}_A, \boldsymbol{x}_A; \boldsymbol{\theta}) \tag{4.1}$$

and the factors Ψ_A of the undirected model G are parameterized as follows:

$$\Psi_A(\boldsymbol{x}_A, \boldsymbol{y}_A; \boldsymbol{\theta}) = \exp\left(\sum_{k=1}^{K(A)} \lambda_{Ak} f_{Ak}(\boldsymbol{x}_A, \boldsymbol{y}_A)\right) \tag{4.2}$$

Here, $K(A)$ denotes the number of feature functions f_{Ak} defining factor Ψ_A, and $\boldsymbol{\theta} \in \mathbb{R}^N = \{\lambda_{Ak}\}$ are the parameters of the CRF. The normalization function $Z(\boldsymbol{x})$ is then defined as

$$Z(\boldsymbol{x}) = \sum_{\boldsymbol{y}} \prod_{\Psi_A \in G} \Psi_A(\boldsymbol{y}_A, \boldsymbol{x}_A; \boldsymbol{\theta}) \tag{4.3}$$

and sums over all possible assignments of $\boldsymbol{y}$.

The factors can be partitioned into a set of *clique templates* $\mathcal{C} = \{C_1, C_2, \ldots C_P\}$, where each clique template C_p is a set of factors whose parameters $\boldsymbol{\theta}_p \in \mathbb{R}^P$ are tied. The CRF can then be rewritten as:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\boldsymbol{y}_c, \boldsymbol{x}_c; \boldsymbol{\theta}_p) \tag{4.4}$$

and the normalization function $Z(\boldsymbol{x})$ is defined accordingly. Some comments may be appropriate at this point:

- A separate random variable $y_i \in \boldsymbol{y}$ is associated with each node of the undirected graphical model G. For our purposes, all random variables will be discrete: the outcomes of these variables correspond to the labels that may be assigned. If variables are adjacent in G, this means that there is a dependency between them.
- The nature of $\boldsymbol{x}$ depends on the task; in sequence tagging, $\boldsymbol{x}$ will typically be an observed sequence of input data.
- Factors Ψ_c are defined over cliques[1] c of G. They assign a *potential* to each assignment of the variable(s) of a clique. If G is a pairwise graph, there are typically univariate and

[1] A clique is a set of pairwise adjacent nodes.

bivariate factors. Bivariate factors define potentials for the joint outcomes of the two variables in a two-node clique.

- The feature functions f_{ck} for a clique template C_p determine the value of the potential for a certain assignment of the variable(s) of factor Ψ_c over clique c. Typically, G has a repetitive structure and the parameters (feature coefficients) $\boldsymbol{\theta_p} = \{\lambda_{pk}\}$ of each clique template are tied across time. Feature functions are often binary; they depend on $\boldsymbol{x}$ (often only a local context $\boldsymbol{x_c}$) and the variable assignment $\boldsymbol{y}_c$ of clique c.
- As an example, a typical feature function in a Part-of-Speech (POS) tagging task might be defined as follows:

$$f_{ck}(\boldsymbol{y}_c, \boldsymbol{x}_c) = \begin{cases} \mathbf{1} & \text{if } \boldsymbol{y}_c = (\texttt{VBZ}, \texttt{DT}) \text{ and } \boldsymbol{x}_c = (\texttt{book}, \ldots) \\ \mathbf{0} & \text{otherwise} \end{cases}$$

  This highlights another property of feature functions: most features functions are ever only non-zero for a particular variable assignment – $(\texttt{VBZ}, \texttt{DT})$ in this case.

As mentioned above, the underlying graphical model G usually has a repetitive structure. Several useful families of CRFs can be distinguished on the basis of that structure and the form of parameter tying.

One well known type are *linear chain* CRFs, which are similar to HMMs as far as their application area is concerned. Linear-chain CRFs consist of only one connected chain of variables $\boldsymbol{y}$; their parameters are tied across time. Factors are typically defined over single-node and two-node cliques. These capture the local probabilities and transition probabilities[2], respectively, given the observed input sequence $\boldsymbol{x}$.

Factorial Conditional Random Fields (FCRFs) are another important type of CRF, which is applied in this thesis. They can be considered a generalization of linear-chain CRFs and a special case of DCRFs (see [SMR07]). FCRFs will be described in detail in the following section.

Other useful types of CRFs exist; the tutorial of Sutton and McCallum ([SM06]) describes these in detail and also gives an overview of how various types of graphical models are related.

[2] It should be noted that from a probabilistic point of view, these are not really probabilities. The term *potentials* may be more appropriate.

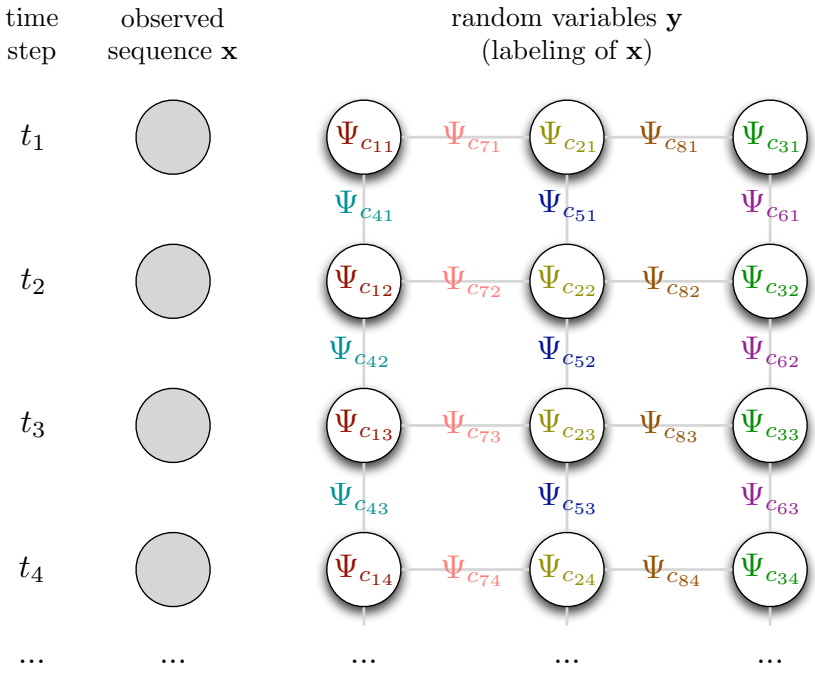

Figure 4.1: A FCRF with 3 label chains (dependencies on $\boldsymbol{x}$ omitted for brevity)

4.2 Factorial Conditional Random Fields

FCRFs consist of multiple chains of equal length. Dependencies exist not only within these chains, but also between co-temporal variables of different chains. As such, FCRFs can be considered a composition of multiple linear-chain CRFs with additional dependencies between chains. Figure 4.1 shows a typical FCRF with 3 chains.

Factors of a FCRF are tied across time. Multiple clique templates exist; the factors defined over cliques of the same clique template are shown in the same color in figure 4.1:

- There is one clique template for the single-node cliques of each chain. Figure 4.1 shows clique templates $C_1 = \{\Psi_{c_{11}}, \Psi_{c_{12}}, \Psi_{c_{13}}, \Psi_{c_{14}}\}$, $C_2 = \{\Psi_{c_{21}}, \Psi_{c_{22}}, \Psi_{c_{23}}, \Psi_{c_{24}}\}$ and $C_3 = \{\Psi_{c_{31}}, \Psi_{c_{32}}, \Psi_{c_{33}}, \Psi_{c_{34}}\}$.
- In addition, there is a clique template for the two-node cliques of each chain (the corresponding factors will be referred to as *in-chain factors* in this thesis): $C_4 = \{\Psi_{c_{41}}, \Psi_{c_{42}}, \Psi_{c_{43}}\}$, $C_5 = \{\Psi_{c_{51}}, \Psi_{c_{52}}, \Psi_{c_{53}}\}$ and $C_6 = \{\Psi_{c_{61}}, \Psi_{c_{62}}, \Psi_{c_{63}}\}$.
- Finally, all two-node cliques between the same two chains have a clique template of their own (the corresponding factors will be called *between-chains factors*): $C_7 = \{\Psi_{c_{71}}, \Psi_{c_{72}}, \Psi_{c_{73}}, \Psi_{c_{74}}\}$ and $C_8 = \{\Psi_{c_{81}}, \Psi_{c_{82}}, \Psi_{c_{83}}, \Psi_{c_{84}}\}$.

Factors that are in the same clique template C_p share the same set of parameters $\{\lambda_{pk}\}$. By inspecting the different clique templates depicted in figure 4.1, one can easily see why this results in the parameters being tied across time.

In general, a FCRF with n label chains will have n single-node clique templates, n in-chain clique templates and $n-1$ between-chains clique templates; this accounts for a total of $3n-1$ clique templates (which are, in effect, separate sets of parameters).

With regards to the observed input sequence $\boldsymbol{x}$, it should be noted that any dependencies of factors $\Psi_{c_{ij}}$ on $\boldsymbol{x}$ are omitted in figure 4.1 for the sake of visual comprehensibility. Do note that all feature functions defining the factors in figure 4.1 can access any element of $\boldsymbol{x}$ at any time step (in fact, this is one of the major advantages of discriminative models like CRFs); however, typically, only the local context of the current time step will be inspected.

Finally, it should be noted that univariate factors are not strictly necessary; the same family of distributions can be defined using bivariate factory only ([SM06]). However, univariate factors may prove to be useful if the training data is rather sparse, since they provide redundancy.

4.3 Inference in Factorial Conditional Random Fields

Inference in a CRF is required for several reasons:

- First, inference is needed to solve the task of finding the most likely labeling for an unseen instance (i.e., the Maximum a Posteriori (MAP) configuration of the variables $\boldsymbol{y}$ of a CRF).
- Second, during training, if Maximum Likelihood Estimation (MLE) is performed, inference is required in order to compute the likelihood $p(\boldsymbol{y}|\boldsymbol{x})$ (most notably the normalization term $Z(\boldsymbol{x})$) and marginal probabilities $p_c(\boldsymbol{y}_c|\boldsymbol{x})$ (where c is a clique)[3].

Multiple algorithms exist that may successfully be used to perform exact inference on trees. Typical examples are the Viterbi (MAP) and Baum-Welch (MLE) algorithms for linear chains, and the max-product (MAP) and sum-product (MLE) variants of belief propagation for any tree structure. In general, every algorithm that is suitable for trees can be applied to arbitrary graphs by clustering nodes in such a way that they form a so-called *junction tree* (see

[3] As we shall see in subsection 4.4.1, Maximum Pseudolikelihood Estimation avoids these costly computations.

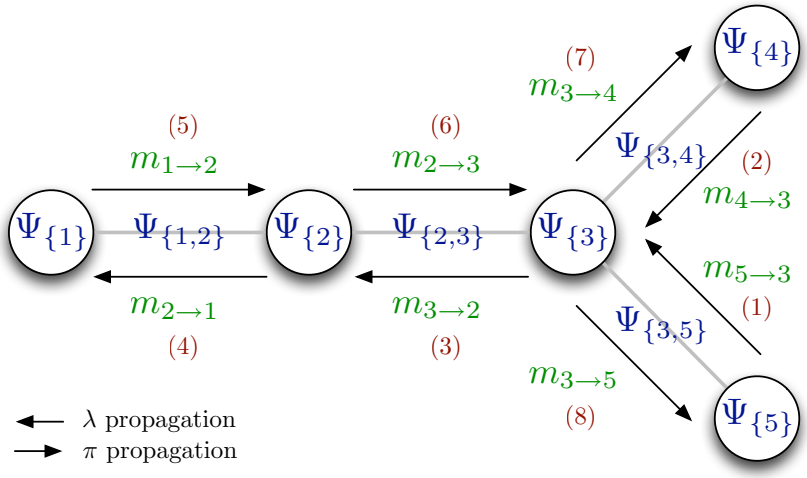

Figure 4.2: Belief propagation on a tree

[WJW02b]). However, this approach yields exact inference results and may often be prohibitively expensive.

Loopy belief propagation generalizes the well-known belief propagation algorithm to arbitrary graphs. As opposed to the junction tree approach, loopy belief propagation is an approximate inference algorithm that allows for reasonable computational cost on many graphs of interest. In particular, loopy belief propagation is suitable for performing inference on FCRFs. The underlying model of FCRFs does not satisfy tree properties, as is evident in figure 4.1.

In the following, the original belief propagation algorithm for inference on trees, and its generalization, loopy belief propagation for approximate inference on arbitrary graphs, will be presented.

4.3.1 Belief Propagation

Belief propagation can perform inference on a tree using two message-passing sweeps: the first sweep (λ-propagation) runs from the leaves up to a designated root; the second sweep (π-propagation) runs from the root to the leaves (see figure 4.2).

In order to proceed, our notation needs to be clarified first. Let c be a clique of an undirected graphical model G. This clique can either be referred to as $\{i\}$ (in the case of a single-node clique) or as $\{i, j\}$ (in the case of a two-node clique), where i and j (with associated variables y_i and y_j) are nodes of G. Analogously, $\Psi_{\{i\}}$ and $\Psi_{\{i,j\}}$ denote the factors defined over a

single-node and a two-node clique, respectively. We will use that notation to describe belief propagation for pairwise graphs as presented by Yedidia ([YFW03]).

The belief propagation messages are then self-consistently determined as follows:

$$m_{i\to j}(y_j) = \sum_{\downarrow y_i} \left(\Psi_{\{i\}}(y_i)\Psi_{\{i,j\}}(y_i, y_j) \prod_{k\in N(i)\setminus j} m_{k\to i}(y_i) \right) \tag{4.5}$$

where $m_{i\to j}$ denotes a message from node $i \in G$ to node $j \in G$. Such a message can be thought of as a vector with one component for each possible outcome of y_j. Each component of $m_{i\to j}$ encodes the current belief of node i about the corresponding outcome of y_j. The message is built by marginalizing the product of all incoming messages (except for the message from j itself), the single-node factor $\Psi_{\{i\}}$ and the two-node factor $\Psi_{\{i,j\}}$ for y_j; this marginalization is performed by summing over all outcomes of y_i (hence $\sum_{\downarrow y_i}$).

In practice, the messages are computed in the order given above (λ-propagation followed by π-propagation). After all messages have been computed, single-node and two-node beliefs are defined as presented in (4.6) and (4.7), respectively:

$$b_{\{i\}}(y_i) = \kappa\Psi_{\{i\}}(y_i) \prod_{j\in N(i)} m_{j\to i}(y_i) \tag{4.6}$$

$$b_{\{i,j\}}(y_i, y_j) = \kappa\Psi_{\{i,j\}}(y_i, y_j)\Psi_{\{i\}}(y_i)\Psi_{\{j\}}(y_j) \prod_{k\in N(i)\setminus j} m_{k\to i}(y_i) \prod_{l\in N(j)\setminus i} m_{l\to j}(x_j) \tag{4.7}$$

Here, κ is a normalization constant that ensures the beliefs sum to 1.

Sum-Product

If belief propagation is performed using the sum-marginalization operator ($\sum_{\downarrow y_i}$), this results in the so-called *sum-product* algorithm. Other operators are feasible, as we shall see shortly.

We note that the following relations hold if G is a tree and messages are computed according to the sum-product algorithm:

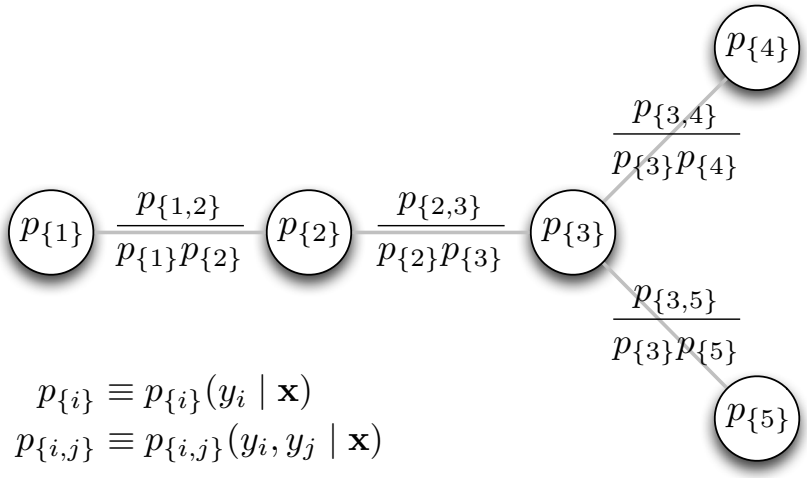

Figure 4.3: Alternative factorization of the tree in figure 4.2

$$b_{\{i\}}(y_i) \equiv p_{\{i\}}(y_i|\boldsymbol{x}) \tag{4.8}$$

$$b_{\{i,j\}}(y_i, y_j) \equiv p_{\{i,j\}}(y_i, y_j|\boldsymbol{x}) \tag{4.9}$$

This means that the single-node and two-node beliefs are equivalent to the marginal probabilities $p_c(\boldsymbol{y}_c|\boldsymbol{x})$, which are required for MLE (see subsection 4.4.1).

Besides the marginals, efficient computation of the normalization term $Z(\boldsymbol{x})$ is of great importance. As (4.3) shows, naive computation requires summing over all assignments of $\boldsymbol{y}$. This is too expensive to be practical. Fortunately, it turns out that belief propagation produces an alternative factorization of $p(\boldsymbol{y}|\boldsymbol{x})$, as follows:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \prod_{\{i\} \in C_s} p_{\{i\}}(y_i|\boldsymbol{x}) \prod_{\{i,j\} \in C_t} \frac{p_{\{i,j\}}(y_i, y_j|\boldsymbol{x})}{p_{\{i\}}(y_i|\boldsymbol{x}) p_{\{j\}}(y_j|\boldsymbol{x})} \tag{4.10}$$

where C_s and C_t are defined to be the sets of all single-node and all two-node cliques, respectively. In other words, the conditional distribution defining the CRF can be expressed in terms of the marginals gained during sum-product belief propagation. This representation does not require any additional normalization, so $Z(\boldsymbol{x})$ need not be computed. The derivation of (4.10) is explained by Wainwright in [WJW02b]; it can be considered a generalization of the well-known factorization of Markov chains. The alternative factorization is depicted in figure 4.3. Further insights on this topic are also discussed by Kschischang et al. (see [KFL01]).

Max-Product

If the max-marginalization operator $\max_{\downarrow y_i}$ is substituted for $\sum_{\downarrow y_i}$ in equation (4.5), this results in the so-called *max-product* algorithm. $\max_{\downarrow y_i}$ marginalizes over all outcomes of y_i by selecting the maximum associated value.

The sum-product variant is useful for MLE, whereas max-product can be used to find the MAP assignment of $\boldsymbol{y}$ ([WJW02a]). The MAP assignment y_i^* of a variable $y_i \in \boldsymbol{y}$ is obtained by computing the belief propagation messages using the max-product algorithm and then selecting the outcome of y_i with the highest belief according to the max-marginals $b_{\{i\}}(y_i)$:

$$y_i^* = \underset{y_i}{\operatorname{argmax}} \left(b_{\{i\}}(y_i) \right) \tag{4.11}$$

Generalizations of the max-product algorithm for finding the M most probable configurations also exist (see [YW04]).

Equation (4.11) is the final piece of a complete inference algorithm for CRFs, which needs to compute marginals and likelihood for MLE and the MAP assignment for labeling unseen instances. However, since any FCRF with > 1 chains and > 1 time steps contains loops (see figure 4.1), we cannot immediately apply the above results (which are valid for trees only) without further consideration.

4.3.2 Loopy Belief Propagation

The good news is that the inference algorithms given above can be used for loopy graphs without extensive adaption. However, a different schedule is needed for message-passing: obviously, λ-propagation and subsequent π-propagation are inapplicable if a graph contains loops.

Typically, loopy belief propagation is performed as follows:

- The messages $m_{i \to j}$ are initialized to 1 (other initializations are possible).
- Messages are sent repeatedly according to some schedule until the process has converged, i.e., newly computed messages $m'_{i \to j}$ do not differ from previous messages $m_{i \to j}$ between the same nodes except for some small ε.
- Another sensible convergence criterion is that at a message must have been sent between any two adjacent nodes at least once (in both directions).

A multitude of schedules are feasible; a simple strategy that seems to fare surprisingly well is to randomly select adjacent nodes. Schedules for belief propagation are analyzed in detail by Sutton and McCallum in [SM07a]. Tree-based Reparameterization (TRP), one particular schedule that has been recommended repeatedly (e.g., [SM06]), will be presented in the next subsection.

So much for the good news; the bad news is that loopy belief propagation is not guaranteed to converge on all graphs, although it has been applied successfully for various tasks in the past ([YFW03], [SM06]). Indeed, examples can be constructed for which loopy belief propagation fails to converge ([YFW03]). This is clearly an undesirable property; still, it is a reasonable trade-off, seeing that exact inference algorithms with guaranteed convergence properties can easily get intractable, whereas loopy belief propagation works mostly well in practice. Practical experience regarding the convergence behavior of loopy belief propagation is discussed in section 6.8.

4.3.3 The TRP Schedule

Tree-based Reparameterization (TRP) is introduced by Wainwright et al. in [WJW02b]. Wainwright's PhD thesis ([Wai02]) contains additional background information.

It should be noted that Wainwright gives two versions of TRP which yield identical results: The first version can be considered a particular schedule for loopy belief propagation, whereas the second version is a message-free algorithm involving a sequence of local reparameterization operations. The former presentation of TRP will be described in this thesis, since it allows to build on the previous results about belief propagation.

Algorithm 2 gives a rough outline of the TRP schedule. With regards to step 1, it should be noted that twice as many messages $m_{i \to j}$ as the number of edges in G are needed; this is due to the fact that messages need to be sent in both directions. The convergence criterion of step 2 also deserves mention: as described in the previous subsection about loopy belief propagation, convergence is usually checked by comparing newly computed messages to their counterparts of the previous iteration; if all messages are equal except for some small ε, the algorithm is considered to have converged (another common constraint is that each message must have been updated at least once).

Step 2a should be fairly clear; each spanning tree τ is an acyclic subgraph that connects all nodes of G. Step 2b is a bit more subtle – consider the message update equation (4.5): it is very important to realize that while step 2b only updates the messages for the edges of τ,

Algorithm 2 TRP SCHEDULE

Given:

- An undirected graphical model G;

1. Initialize the components of all messages $m_{i \to j}$ for edges of G to 1.
2. while `not converged`:
   a) Randomly select a spanning tree $\tau \in G$;
   b) Perform a λ-sweep followed by a π-sweep on τ, thereby updating the messages $m_{i \to j}$ for all edges of τ;
3. return all messages $\{m_{i \to j}\}$.

these updates need to incorporate incoming messages from all nodes that are adjacent in G. In particular, this means that the term $N(i) \setminus j$ of equation (4.5) refers to adjacency in G, rather than τ. If this is disregarded, inference on τ will be performed completely independent of any previous iteration. The messages for τ are then essentially re-computed from scratch at each iteration, and the algorithm will fail to converge.

Once Algorithm 2 has converged, the single-node and two-node beliefs are then defined as per equation (4.6) and (4.7).

Concluding the section on inference, it should be mentioned that the presentation of belief propagation in this thesis is restricted to pairwise graphs; generalizations to arbitrary clique sizes do exist and are also described in detail by Yedidia ([YFW03]). Since we are concerned with pairwise FCRFs, these algorithms exceed the scope of this thesis.

4.4 Parameter Estimation

Parameter estimation is the task of determining the parameters $\boldsymbol{\theta}$ of a CRF from independent and identically distributed (IID) training data $\mathcal{D} = \{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\}_{i=1}^{N}$.

As Sutton notes ([SM06]), the training instances $\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}$ can be considered disconnected components of a single undirected model G. The clique templates $\{C_1, C_2, \ldots, C_P\}$ are then assumed to extend over the factors of all training instances. This spares us from explicitly summing over $\sum_{i=1}^{N}$ in the following.

4.4.1 The Objective Function

Ultimately, the goal is to achieve high prediction accuracy on unseen data $\mathcal{T}$. This is accomplished by choosing the parameters $\boldsymbol{\theta}$ in such a way that they fit the training data $\mathcal{D}$. In order to do so, an objective function is needed that measures how well the parameters fit $\mathcal{D}$; the parameters $\boldsymbol{\theta} = \{\lambda_{pk}\}$ are then adjusted so that the objective function reaches its optimum. Different objective functions are feasible.

Maximum Likelihood Estimation (MLE)

Frequently, the parameters are chosen such that they optimize the conditional likelihood $p(\boldsymbol{y}|\boldsymbol{x})$ given the fully observed training data $\mathcal{D}$. This principle is called Maximum Likelihood Estimation (MLE). Typically, for numerical reasons, one chooses to optimize the logarithm of the conditional likelihood. This results in the following objective function, which is obtained by taking the logarithm of equation (4.4):

$$\ell(\boldsymbol{\theta}) = \sum_{C_p \in \mathcal{C}} \sum_{\Psi_c \in C_p} \sum_{k=1}^{K(p)} \lambda_{pk} f_{pk}(\boldsymbol{x}_c, \boldsymbol{y}_c) - \log Z(\boldsymbol{x}) \tag{4.12}$$

Naive computation of the normalization term $Z(\boldsymbol{x})$ is intractable; however, using the inference algorithms presented in section 4.3, one can avoid that computation.

If an optimization algorithm shall be used that optimizes by gradient, the partial derivatives of the objective function need to be calculated. The partial derivative of $\ell(\boldsymbol{\theta})$ with respect to a parameter λ_{pk} of clique template C_p is:

$$\frac{\partial \ell}{\partial \lambda_{pk}} = \sum_{\Psi_c \in C_p} f_{pk}(\boldsymbol{x}_c, \boldsymbol{y}_c) - \sum_{\Psi_c \in C_p} \sum_{\boldsymbol{y'}_c} f_{pk}(\boldsymbol{x}_c, \boldsymbol{y'}_c) p_c(\boldsymbol{y'}_c|\boldsymbol{x}) \tag{4.13}$$

Note that $\boldsymbol{y'}_c$ ranges over all possible label assignments of clique c. The gradient can be considered the difference between the expected value of feature f_{pk} under the empirical distribution of the training data and the expectation of f_{pk} under the model distribution ([SM06]). It is rather intuitive that one wants to minimize this value. Computation of the gradient requires the marginal probabilities $p_c(\boldsymbol{y}_c|\boldsymbol{x})$ for each clique c. Again, these can be computed efficiently using the algorithms discussed in the previous section.

Maximum Pseudolikelihood Estimation

Pseudolikelihood is an approximation of true likelihood which uses local information; in doing so, it avoids costly inference. It is a well-known result that if the model family includes the true distribution, then pseudolikelihood converges to the true parameter setting in the limit of infinite data ([SM07b]).

Various variants of Maximum Pseudolikelihood Estimation exist for CRFs; the objective function we present here is a factor-based variant of log-pseudolikelihood as described by Sanner et al. ([SGHM07]):

$$p\ell(\boldsymbol{\theta}) = \sum_{C_p \in \mathcal{C}} \sum_{\Psi_c \in C_p} \log p_c(\boldsymbol{y}_c | \boldsymbol{x}, MB(\Psi_c)) \tag{4.14}$$

As one can see, this objective function does not require true inference in a CRF, since the pseudo-marginals are conditioned on the Markov blanket[4] of the corresponding factor. By its very definition, this approach can only be applied during parameter estimation from training data, since the "true" variable assignment of the Markov blanket needs to be known.

The gradient can be calculated similar to equation (4.13), except that the marginals $p_c(\boldsymbol{y'}_c|\boldsymbol{x})$ are also conditioned on the Markov blanket, i.e., $p_c(\boldsymbol{y'}_c|\boldsymbol{x}, MB(\Psi_c))$.

Other Approaches

Various other approaches have been suggested; most of these employ some kind of local training in order to avoid the substantial computational effort required for inference over a whole graph. Some of these approaches are discussed and compared in [SM07b].

4.4.2 Regularization

If parameters $\boldsymbol{\theta}$ are determined from training data – in particular if training data is sparse – this harbors the danger of **overfitting**. Overfitting means that the estimated parameters fit the training data $\mathcal{D}$ extremely well, but do not achieve good accuracy on unseen data $\mathcal{T}$.

In order to reduce this effect, extremely large or small parameters are typically penalized; this process is called regularization. Gaussian priors have been used extensively for this purpose

[4] Here, the Markov blanket of a factor Ψ_c denotes the set of variables occurring in factors that share variables with Ψ_c, non-inclusive of the variables of Ψ_c.

in the current literature, but other choices may yield reasonable results as well (see [PM04] for a comparison of some regularization methods).

The use of a Gaussian prior will be assumed in this thesis. In that case, if $f(\boldsymbol{\theta})$ is the original objective function (e.g., log-likelihood or log-pseudolikelihood), a penalized version

$$f'(\boldsymbol{\theta}) = f(\boldsymbol{\theta}) - \sum_{k=1}^{N} \frac{\lambda_k^2}{2\sigma^2} \tag{4.15}$$

will be optimized instead. This results in the following partial derivative with respect to parameter λ_k:

$$\frac{\partial f'}{\partial \lambda_k} = \frac{\partial f}{\partial \lambda_k} - \frac{\lambda_k}{\sigma^2} \tag{4.16}$$

The regularization parameter $\frac{1}{2\sigma^2}$ determines the strength of the penalty. This is a free parameter; optimizing it might require a computationally expensive parameter sweep. However, Sutton notes that the accuracy of the final model is often not sensitive to σ^2, even if the parameter is varied up to a factor of 10 ([SM06]).

4.4.3 Convex Optimization Algorithms

All objective functions $f(\boldsymbol{\theta})$ given in subsection 4.4.1 are concave. This follows from the convexity of functions of the form $g(\boldsymbol{x}) = \log \sum_i \exp(x_i)$ ([SM06]). Therefore, any concave optimization algorithm can be used to optimize $f(\boldsymbol{\theta})$.

In practice, it is common to minimize the penalized negative objective function $-f'(\boldsymbol{\theta})$ and call it the *loss function*. Minimizing the loss function requires a convex optimization algorithm. Therefore, the algorithms in this section all *minimize* the given objective function. Finally, it should be noted that LBFGS and OLBFGS are presented here as described by Schraudolph et al. ([SYG07]).

Limited Memory BFGS (LBFGS)

LBFGS is a variant of the classic Quasi-Newton Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm that was designed for solving large-scale optimization problems.

BFGS incrementally updates an estimate of the inverse Hessian $\boldsymbol{B}_t$ of the objective function $f(\boldsymbol{\theta}), \boldsymbol{\theta} \in \mathbb{R}^n$. This operation is $O(n^2)$ with regards to memory requirements and runtime

complexity. For applications of NLP, CRFs can easily require hundreds of thousands or even millions of parameters; for such cases, BFGS is prohibitively expensive.

In LBFGS, on the other hand, the estimation of the inverse Hessian is based only on the last m steps in gradient and parameter space. The quasi-Newton direction can be obtained directly from these steps. This reduces complexity to $O(nm)$, which is a huge improvement (typical values for m range from 3 to 10).

Consider Algorithm 3. LBFGS is an iterative algorithm that terminates once a certain convergence criterion is fulfilled (see step 2); typically, one checks whether the norm of the gradient ∇f falls below a certain ε. At each iteration, a direction update is first performed (step 2a). Subsequently, a line search function obeying the Wolfe conditions[5] is invoked in order to determine the step length (step 2b). The parameters are then updated using the scaled step $\boldsymbol{s}_t$, and the difference $\boldsymbol{y}_t$ between the old and the new gradient ∇f is computed (steps 2c - 2e).

Algorithm 3 STANDARD LBFGS METHOD

Given:

- objective f and its gradient $\nabla f := \dfrac{\partial}{\partial \boldsymbol{\theta}} f(\boldsymbol{\theta})$
- initial parameter vector $\boldsymbol{\theta}_0$;
- line search `linemin` obeying Wolfe conditions;
- convergence tolerance $\varepsilon > 0$;

1. $t := 0$;
2. while $\|\nabla f(\boldsymbol{\theta}_t)\| > \varepsilon$:
   (a) $\boldsymbol{p}_t =$ LBFGS DIRECTION UPDATE;
   (b) $\eta_t = \texttt{linemin}(f, \boldsymbol{\theta}_t, \boldsymbol{p}_t)$;
   (c) $\boldsymbol{s}_t = \eta_t \boldsymbol{p}_t$;
   (d) $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{s}_t$;
   (e) $\boldsymbol{y}_t = \nabla f(\boldsymbol{\theta}_{t+1}) - \nabla f(\boldsymbol{\theta}_t)$;
   (f) $t := t + 1$;
3. return $\boldsymbol{\theta}_t$.

The direction update (Algorithm 4) uses the last m vectors $\boldsymbol{y}$ and $\boldsymbol{t}$ in order to compute the step direction for each iteration. Typically, an implementation of LBFGS will maintain ring

[5] The Wolfe conditions specify sufficient decrease and curvature conditions.

Algorithm 4 LBFGS DIRECTION UPDATE

Given:

- integers $m > 0, t \geq 0$;
- $\forall i = 1, 2, \ldots, \min(t, m)$:
  vectors $\boldsymbol{s}_{t-1}$ and $\boldsymbol{y}_{t-1}$ from Algorithm 3;
- current gradient $\nabla f(\boldsymbol{\theta}_t)$ of objective f;

1. $\boldsymbol{p}_t := -\nabla f(\boldsymbol{\theta}_t)$;
2. for $i := 1, 2, \ldots, \min(t, m)$:
   (a) $\alpha_i = \dfrac{\boldsymbol{s}_{t-i}^\top \boldsymbol{p}_t}{\boldsymbol{s}_{t-i}^\top \boldsymbol{y}_{t-i}}$;
   (b) $\boldsymbol{p}_t := \boldsymbol{p}_t - \alpha_i \boldsymbol{y}_{t-i}$;
3. if $t > 0$: $\boldsymbol{p}_t := \dfrac{\boldsymbol{s}_{t-1}^\top \boldsymbol{y}_{t-1}}{\boldsymbol{y}_{t-1}^\top \boldsymbol{y}_{t-1}} \boldsymbol{p}_t$;
4. for $i := \min(t, m), \ldots, 2, 1$:
   (a) $\beta = \dfrac{\boldsymbol{y}_{t-i}^\top \boldsymbol{p}_t}{\boldsymbol{y}_{t-i}^\top \boldsymbol{s}_{t-i}}$;
   (b) $\boldsymbol{p}_t := \boldsymbol{p}_t + (\alpha_i - \beta) \boldsymbol{s}_{t-i}$;
5. return $\boldsymbol{p}_t$.

buffers of $\boldsymbol{y}$ and $\boldsymbol{t}$ for this purpose. We refer to the original article of Nocedal ([Noc80]) for details about this part of the algorithm.

In practice, LBFGS works remarkably well for CRF training, leading to fast convergence; it is now used as the default optimization algorithm in various CRF toolkits ([Kud05], [Sut06], [PNN05]). This seems to confirm earlier results by Wallach ([Wal02]).

Online LBFGS (OLBFGS)

Recently, Schraudolph et al. ([SYG07]) developed OLBFGS, a stochastic variant of LBFGS for online convex optimization. Stochastic (or online) gradient-based methods obtain their gradient estimates from small subsamples (batches) of training data. At each iteration, the algorithm updates the parameters $\boldsymbol{\theta}$ using one small batch of data. This means that adaptation of parameters can start much earlier, compared to a traditional approach where the gradient

Algorithm 5 ONLINE LBFGS METHOD

Given:

- stochastic approximation of convex objective f and its gradient ∇f over data sequence $\boldsymbol{X}_t$;
- initial parameter vector $\boldsymbol{\theta}_0$;
- sequence of step sizes $\eta_t > 0$;
- parameters $\lambda \geq 0, \epsilon > 0$;

1. $t := 0$;
2. while `not converged`:
   (a) $\boldsymbol{p}_t =$ OLBFGS DIRECTION UPDATE;
   (b) $\boldsymbol{s}_t = \eta_t \boldsymbol{p}_t$;
   (c) $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{s}_t$;
   (d) $\boldsymbol{y}_t = \nabla f(\boldsymbol{\theta}_{t+1}, \boldsymbol{X}_t) - \nabla f(\boldsymbol{\theta}_t, \boldsymbol{X}_t) + \lambda \boldsymbol{s}_t$;
   (e) $t := t + 1$;
3. return $\boldsymbol{\theta}_t$.

needs to be computed for the whole training data before the first parameter update can occur. Computational requirements can be greatly reduced on large, redundant data sets.

Consider Algorithm 5. Here, $\boldsymbol{X}_t$ denotes one batch of data – the one intended for iteration t. Another remarkable difference compared to Algorithm 3 is that OLBFGS does not use line search. Schraudolph et al. note that line searches are highly problematic in a stochastic setting, since the global criteria they employ cannot be established from local subsamples. Instead, Algorithm 5 employs a sequence of step sizes η_t. A commonly used decay schedule for this sequence is given by $\eta_t = \frac{\tau}{\tau + t}\eta_0$. Optimal values for τ and η_0 depend on the data and on the batch size.

Algorithm 5 iterates until the convergence criteria are met. The convergence test of Algorithm 3 is insufficient in a stochastic setting; it must be replaced with a more robust one. Schraudolph et al. suggest checking whether the gradient ∇f has remained below a given threshold for the last k iterations. The first step in each iteration is to determine the direction update (step 2a). Subsequently, the current step $\boldsymbol{s}_t$ is determined using step size η_t and the new direction $\boldsymbol{p}_t$, and the parameters are updated accordingly (steps 2b - 2c). Finally, step 2d computes $\boldsymbol{y}_t$. Note that the difference of gradients must be computed on the same batch

Algorithm 6 OLBFGS DIRECTION UPDATE

Identical to Algorithm 4, except that step 3 is replaced by:

$$\boldsymbol{p}_t := \begin{cases} \epsilon \boldsymbol{p}_t & \text{if } t = 0; \\ \dfrac{\boldsymbol{p}_t}{\min(t,m)} \displaystyle\sum_{i=1}^{\min(t,m)} \frac{\boldsymbol{s}_{t-i}^\top \boldsymbol{y}_{t-i}}{\boldsymbol{y}_{t-i}^\top \boldsymbol{y}_{t-i}} & \text{otherwise.} \end{cases}$$

$\boldsymbol{X}_t$. This doubles the number of gradient calculations, but is needed in the stochastic setting to prevent sampling noise from entering the direction update ([SYG07]). The additional term $\lambda \boldsymbol{s}_t$ is introduced to cope with regions of low curvature; $\lambda > 0$ is a free model-trust region parameter in this context.

The direction update for OLBFGS (Algorithm 6) is almost identical to the direction update for LBFGS (Algorithm 4). However, Schraudolph et al. introduce a small refinement which ensures that the first parameter update is small and improves online performance by averaging away some of the sampling noise.

While OLBFGS does perform remarkably well under ideal settings, its biggest problem right now is the substantial number of free parameters (η_0, τ, λ), which needed to be tuned according to data, batch size and m. Schraudolph et al. do note, however, that they anticipate further insight regarding ways to automatically set and adapt them ([SYG07]).

The author's own experience with OLBFGS indicates that bad parameter settings can easily lead to divergence of the training algorithm. Manual tuning may be extremely time consuming for large tasks, because divergence often only happens after a large number of iterations.

Chapter 5

Implementation Overview

> *“ Knowledge that is not put into practice is like food that is not digested. ”*
>
> *– Sri Sathya Sai Baba*

The theoretical framework presented in chapter 4 is a sound basis for practical implementation. Yet, there are only a handful of publicly available CRF packages. To the best of the author's knowledge, only two of these support Factorial Conditional Random Fields (FCRFs):

- Charles A. Sutton's **Graphical Models In Mallet (GRMM)** (see [Sut06]): An excellent and very flexible toolkit written in Java. GRMM has support for arbitrary CRF structures (which subsume FCRFs, of course); however, that flexibility comes at a price. The code makes generous use of runtime polymorphism even in performance-critical sections and leaves a lot of room for micro optimization. In addition, its memory requirements are quite substantial.
- Kevin Murphy's **CRF Toolbox for Matlab** supports 2D lattices, but it is restricted to binary labels ($+1$ and -1) and seems to be intended for solving computer vision problems. See [MS06] for details.

Other CRF implementations that do not support FCRFs include Taku Kudo's CRF++ ([Kud05]), Sunita Sarawagi's CRF package ([Sar04]) and the FlexCRFs toolkit by Xuan-Hieu Phan et al. The small number of options for FCRFs is probably both due to the fact that CRFs are a relatively recent development and the relatively high effort required for a thorough implementation (as compared to HMMs, for instance).

Choosing GRMM might have been an option; however, seeing that CRF training times would be a major challenge for the large problem at hand, a lean and efficient from scratch implementation seemed to be the best bet. Indeed, the venture turned out very well, and the resulting CRF toolkit is one of the bigger contributions of this thesis.

5.1 Introducing VieCRF

The Vienna Conditional Random Field Toolkit (VieCRF) is a fast toolkit for Factorial Conditional Random Fields. It was designed from scratch to provide high runtime performance, good scalability and a reasonably low memory footprint.

VieCRF consists of three major parts:

1. The **C++ API** builds the core of VieCRF. It is implemented efficiently using generic programming techniques and wholly contained in a number of header files. All performance-critical algorithms are part of the core. The C++ source code is documented using VieCRF's own inline documentation system (which is similar to perldoc).
2. The **Perl API** exposes the functionality of the C++ Application Programming Interface (API) to Perl code. In addition to this wrapper code, the Perl API also implements some convenience modules for less performance-critical functionality like reading in data files, maintaining a mapping between strings and integer IDs, etc. All modules are thoroughly documented using POD/perldoc.
3. The **Perl utilities** include command line tools that make VieCRF's functionality available to users without knowledge of programming languages. These tools are suitable for experimenting with data and model parameters, creating plots, evaluating accuracy and many more tasks. It is also convenient to invoke them from within shell scripts. Complete documentation for all tools is realized via POD/perldoc.

VieCRF is freely available at `http://www.ofai.at/~jeremy.jancsary/`. It is steadily growing with regards to the implemented functionality and becoming more mature. All experiments presented in this thesis were conducted using VieCRF.

At the time of writing, the C++ API comprises about 9400 LOC; the Perl API accounts for roughly 5000 LOC and the corresponding tools are implemented in just about 3100 LOC. These numbers include inline documentation. Therefore, VieCRF is still easily comprehensible and it should be possible even for people who are unfamiliar with VieCRF to add new functionality.

5.2 Implemented Functionality

The functionality available within VieCRF was mostly dictated by the demands of this thesis. More recently, though, some new features found their way into the core of VieCRF which were not used by the experiments described in chapter 6. In fact, by now, there is such a wealth of parameters, inference algorithms and training algorithms that evaluating their effects and interaction would be a worthy task of its own.

5.2.1 Flexible Feature Support

The description of features in chapter 4 is quite abstract and theoretically motivated. In practice, VieCRF expects a user to provide a number of *observations* for each time step of a sequence. These observations are often binary and indicate a certain condition that holds at a particular time step. For a natural language processing task, such an observation could be `infarction@0`, indicating that the token at the current position in an input document is "infarction".

VieCRF can then create a CRF feature for each possible assignment of each clique template from the observations in the training data, i.e., `infarction@0`, coupled with a label unigram or a label bigram would correspond to one feature. Such a feature is active only when the particular observation is active at a given time step and the label assignment matches. Thus, the clique templates in VieCRF actually maintain a vector of weights for each of their assignments, corresponding to the supported observations. This allows VieCRF to find out how strong an indication a certain observation is for a particular label assignment – the stronger the indication, the higher the *feature weight*. These weights correspond to the *parameters* θ of a CRF. Each clique template C_p maintains its own set of parameters θ_p, since the possible label assignments differ.

Now, it is obvious (and indeed intended) that certain observations will never be active together with a particular label assignment in the training data. As an example, in a Part-of-Speech (POS) tagging task, a time step with an observation of `the@0` will never be labeled as `ADJ`, simply because the determiner "the" cannot be an adjective.

This brings up the question of how to handle these cases. VieCRF implements three different strategies:

- The most obvious strategy is to simply allocate a feature weight for each observation paired with each label assignment (as outlined above). This strategy will be referred

to as using **all features** from now on. The drawback is that this can quickly lead to millions of features and unfeasible training times.

- More commonly, only the **supported features** are allocated. These are combinations of an observation and a label assignment that actually occur in the training data. However, in general, this leads to slightly reduced accuracy since such features can only encode a *positive* indication for a certain label assignment. Unsupported features, on the other hand, can receive a negative feature weight to actively encode that a certain observation is a negative indication for a particular label assignment.
- Any label assignment has at least one weight for the so-called **default feature**. The default feature is active for any time step and any label assignment in the training data. This is useful for cases where no other feature is active. VieCRF allows to reduce the feature support of a clique template to the default features; these features then effectively capture the a priori probability of each label assignment. This approach is quite natural: if nothing else is known about which label assignment should be chosen, the one with the highest a priori probability will be picked. Note that this feature reduction usually only makes sense for bivariate clique templates (transition probabilities will be captured in this case).

VieCRF allows to specify a *feature value* along with each observation. These should be seen as an indication of how strongly pronounced a particular observation is. Most notably, using the feature value to encode a certain symbolic meaning (as in: 1.0 – "red", 2.0 – "blue", 3.0 – "green", ...) is a grave mistake. Instead, binary features should be used for such cases (`is_red`, `is_blue`, `is_green`, ...).

5.2.2 Pre-Pruning of Observations

In a typical NLP task, such as the work presented in this thesis, a training corpus will contain tens if not hundreds of thousands of distinct observations, simply because most observations will – in one or another way – be related to the words occurring in the corpus.

This raises the question of feature selection. It should be noted that through regularization (see subsection 4.4.2), CRFs already include a mathematically principled method of reducing the effects of overfitting. Therefore, dismissing all irrelevant features may not be as essential a task as for some other formalisms.

However, regularization does not help reduce training time. If this is an issue, it may be appropriate to prune features that contribute little to the overall prediction accuracy of a model.

A comparative study on feature selection in text categorization is presented by Yang and Pedersen in [YP97]. One of the feature selection criteria that fares well in this comparison, albeit very simple, is *document frequency*. The idea behind this criterion is that features that occur in very few training documents cannot usually have great impact on prediction accuracy of a classifier.

VieCRF adopts this idea (because it is conceptually simple and can easily be applied to CRFs) and implements it in the form of *instance frequency*: Observations that only occur in few training instances may be pruned. No feature weights are allocated for such observations, thereby effectively reducing training time. This process is referred to as *pre-pruning* in VieCRF's manual because the features are removed before they even find their way into the training step.

5.2.3 Inference and Training Algorithms

For training/feature estimation, VieCRF supports the convex optimization algorithms presented in subsection 4.4.3: the limited-memory Quasi-Newton approach LBFGS and its stochastic online variant OLBFGS. In general, LBFGS seems to be much easier to use than OLBFGS, because it does not require tuning of free parameters; this is why LBFGS is the default choice in VieCRF. However, Schraudolph shows that OLBFGS may reach the minimum of the loss function significantly faster on redundant training data *if* suitable parameters are specified.

For inference, VieCRF implements loopy belief propagation with a tree-based schedule (TRP). The sum-product variant is applied for MLE training, whereas the max-product variant is used for labeling unseen instances. See section 4.3 for in-depth discussion of these algorithms.

Alternatively, VieCRF supports maximization of pseudolikelihood. In that case, no real inference is required. However, the pseudolikelihood-based approach only applies to parameter estimation. It cannot be used for labeling unseen instances by its very definition.

5.2.4 Restriction of Label Transitions

In some cases, it may be preferable to prevent certain label transitions; for instance, in the segmentation task of this thesis, label transitions such as `Diagnosis-Plan` should be avoided. The BIO notation requires that a BEGIN label be used to indicate the beginning of a new section type: `Diagnosis-BeginPlan`.

There are several ways of enforcing such constraints:

- The relevant elements of the bivariate factors can explicitly be set to a fixed value of zero, thereby preventing "forbidden" solutions (the a posteriori probability of any path involving such transitions will end up being zero).
- Roth and Yi present an approach based on Integer Linear Programming (ILP) in [RY05]. They view the inference problem as the task of finding the shortest path through the Viterbi trellis. Shortest path search can be performed using ILP by representing the task as a set of linear inequalities which are then solved by any ILP solver. This approach has the advantage that additional user-defined inequalities can be added before the solver is invoked. These inequalities allow for expression of arbitrary boolean constraints over the predicted label sequence. However, as of now, this approach has only been presented for linear-chain CRFs.

VieCRF implements both approaches presented above; the latter approach is of little use for the purpose of this thesis, though. Finally, a generalization of CRFs called Semi-Markov CRFs should be mentioned. This formalism was first presented by Sarawagi ([SC04]). Semi-Markov CRFs are particularly suitable for segmentation tasks because labels are assigned to variable-length "segments" rather than single time steps. This method inherently solves the issue of invalid label transitions for segmentation tasks; however, again, it is only applicable to linear-chain structures at this time. Semi-Markov CRFs are not yet available in VieCRF.

5.2.5 C++ and Perl APIs

VieCRF was designed from scratch so that most relevant functionality could easily be exposed to scripting languages. Scripting languages can provide a great productivity boost while experimenting with various algorithms and parameter settings. This lends itself well to the explorative approach that is usually taken when performing machine learning experiments.

So far, VieCRF comprises a fairly complete Perl API. Perl is the programming language of choice for many NLP tasks and was thus a natural candidate. The most prominent user of the Perl API is the `viecrf` command line tool itself. This ensures that most if not all of the implemented functionality will remain available to Perl users.

5.3 Efficiency Considerations

Efficient implementation of all core algorithms is of primary importance to any CRF implementation. The discriminative nature of CRFs demands that a convex optimization algorithm involving tens if not hundreds of iterations be applied to finding the optimal features weights; each of these iterations require an updated gradient of the loss function which in turn requires running inference for each instance of the training corpus if MLE is performed.

VieCRF implements several strategies that help to keep training time reasonably low.

5.3.1 Avoiding Log-space Computation

In the sum-product variant of loopy belief propagation, summing of factor elements is a frequent operation. Typically, for numerical stability, the logarithm of the factor elements is used. Unfortunately, addition of numbers is not naturally defined in logarithmic space. This means that numbers need to exponentiated first, and can only then be summed. A numerically stable variant of this operation is defined as follows ([SM06]):

$$a \oplus b = a - \log(1 + e^{b-a}) = b + \log(1 + e^{a-b})$$

Typically, the version of the identity with the smaller exponent will be used. However, albeit numerically stable, this operation can be prohibitively expensive. It involves two $\log$ and two $\exp$ invocations for one primitive operation. On Intel ® processors, VieCRF optimizes this operation using a routine of the Integrated Performance Primitives (IPP) ([Tay07]).

Alternatively, VieCRF leaves the choice of not maintaining factor elements in logarithmic space. The factor elements will then be normalized regularly such that they sum to 1, thereby magnifying particularly small numbers ([SM06]). This approach may be slightly less stable from a numerical point of view; however, it results in much faster factor operations, and the author has yet to see a real-world training corpus on which the log-space approach results in noticeably different feature weights.

5.3.2 Fast Sparse Vector Operations

Since VieCRF usually maintains a sparse list of feature weights (by default, only weights for *supported* features are allocated), it is frequently necessary to merge the list of supported

features and the list of those features that are active at a given time step. Most importantly, this is required for computing factor elements (the dot product between feature weights and feature values is computed for this step).

These sparse vector operations can consume a considerable amount of training and prediction time (although the time required for inference usually dominates).

VieCRF implements sparse vector operations using the same strategy as Meschach ([SL94]), a high-performance library for matrix computations in C:

- If both sparse vectors contain roughly the same number of elements, linear merging is performed.
- If one vector contains significantly fewer elements than the other, binary search will be applied to find the corresponding elements in the more densely populated vector.

Both approaches require that the sparse vector elements be sorted according to their (non-sparse) indices. In practice, for CRF training, the second approach seems to be advantageous if one vector is populated at least ten times as densely as the other.

5.3.3 Parallelization

CRFs allow for parallelization of training. At each step of the training algorithm, inference needs to be run for each training instance; while the steps of the training algorithm need to be run in sequence, inference can be performed for multiple training instances at the same time without violating any data dependencies.

VieCRF exploits this fact and scales up to an arbitrary number of CPUs on a shared-memory system (as long as there are at least as many training instances as CPUs, that is). Synchronization overhead is negligible, and the computations performed by the training algorithm itself only account for a small fraction of the overall computational effort, so VieCRF scales up almost perfectly (on a system with a 4 CPU/core SMP configuration, training will typically only require a fourth of the training time).

On a related note, the FlexCRFs package ([PNN05]) can perform parallel CRF training on distributed memory systems (i.e., separate network nodes) using MPI. Such functionality might be implemented in VieCRF at a later point.

5.3.4 Compile-time Polymorphism

VieCRF tries to achieve nice object-oriented encapsulation while still maintaining the highest performance possible. Polymorphic type hierarchies can be quite costly, since method invocation may involve lookups in the virtual table of an object.

Such overhead is avoided by VieCRF by exploiting the powerful template system of C++. Compile-time polymorphism is applied instead of run-time polymorphism in all performance-critical code regions. Numerical libraries such as Blitz++[1] have shown that this strategy can equal if not surpass the performance of loosely structured, highly specialized Fortran or C code ([VJ97]).

5.3.5 Parameterizable Data Types

Finally, VieCRF allows for parameterization of the floating-point data types used for storing features values and performing factor operations.

For most applications, it will be preferable to hold feature values in memory using single precision only. That way, a lot of main memory can be saved. Certainly, this depends on the application domain; in NLP, features are typically binary (1.0 or 0.0), so single precision is entirely sufficient.

The data type used for factor operations is a bit more intricate; for CRF training involving a multitude of possible label assignments or particularly long sequences/complex structures one will usually have to resort to double precision. However, in other scenarios, where the individual training instances are rather simple, but the corpus consists of tens of thousands of instances, single precision may be adequate and will speed up training considerably (in addition to lower memory requirements).

Combined with the choice of performing factor operations in logarithmic space, parameterizable data types allow for great flexibility when trading off computational performance against numerical stability.

[1] http://www.oonumerics.org/blitz/

Chapter 6

Experiments

> *“A little experience often upsets a lot of theory.”*
>
> *– Samuel Parkes Cadman*

In this chapter, experiments will be presented that investigate the practicability of the approach introduced in the previous chapters. For this purpose, the data described in chapter 3 will be used to train and evaluate CRF models. Detailed statistics and performance measurements are provided.

The first section of this chapter discusses the experimental settings; the approach towards training and evaluating CRFs will be explained. Post-processing of assigned labels will also be addressed.

In the second section, the accuracy of all relevant configurations will be assessed. This shall serve to give a rough general impression of how well the approach works. The initial impression is then refined in section 6.3 by presenting precision, recall and F1 on a per-label basis, as well as macro-averaged variants of these metrics.

Subsequently, section 6.4 goes on to analyze typical errors. Confusion plots are provided for visual comprehensibility; they give insights into how well the topic detection task is solved. Section 6.5, on the other hand, presents results for the WindowDiff metric; this allows for realistic assessment of segmentation quality.

The remaining sections study the impact of various phenomena and settings on the resulting CRF models. Section 6.6 sheds light on the effect of noisy training data. Section 6.7 assesses the impact of early stopping during parameter estimation. Finally, section 6.8 presents preliminary findings regarding the convergence behavior of label prediction.

6.1 Training, Labeling and Post-Processing

For evaluation, 2007 annotated reports of C_{COR} and the corresponding annotated reports of C_{RCG} were available. Remember that C_{COR} denotes the corpus of manually corrected, properly formatted corpus, whereas C_{RCG} refers to the raw, unprocessed output of speech recognition. The goal is to automatically predict the underlying structure in unseen output of speech recognition (i.e., data similar to that of C_{RCG}); however, C_{COR} is also useful for evaluation.

Each report of C_{COR} or C_{RCG} will be referred to as a *training instance*, or *instance*, for short. These training instances are all divided into *time steps*, which correspond to the tokens of a report. The annotation of each time step consists of the expected labels, which describe the structure of the report (see figure 2.1), as well as a number of active features or observations, which give hints regarding the expected labels for that time step. Feature generation was discussed in section 3.4. Naturally, the expected labels are available to the machine learning algorithms only during *training*. During *testing*, the features of each time step serve as the input from which the labels shall be predicted.

For the purpose of evaluating the practicability of our CRF-based approach, we consider two related scenarios:

- C_{COR} will serve to estimate the performance to be achieved under ideal conditions. The features for these reports were prepared in such a way that they dismiss any formatting information (capital letters, blanks, line breaks, etc.); however, punctuation and headings all find their way into the features. Basically, perfect dictation is simulated: speech recognition achieves 100% accuracy, and the speaker properly dictates headings, punctuation, enumerations and related items. Note that such dictation still contains a lot of ambiguity and variance; many different variations of one and the same heading exist, for instance.
- C_{RCG}, on the other hand, is used to assess the performance under more realistic conditions. The features are derived from the output of speech recognition (with varying error rates) on actual, less than perfect dictation without any kind of editing whatsoever. Speakers do not consistently dictate punctuation, headings or other markup elements. Note that the label annotation for these training instances was created semi-automatically (see section 3.3). This means that the expected labels may be erroneous and contain noise. Any performance metric determined on C_{RCG} will thus be slightly lower than it could have been, assuming manually labeled training instances. An attempt is made in section 6.6 to assess the effect of noisy training data.

Whenever we refer to C_{COR} or C_{RCG} in the following, the respective scenario described above is intended. All performance metrics will be determined for both scenarios.

Unless indicated otherwise, each corpus was partitioned into three sets, with two parts used for training (1338 instances) and the remainder (669 instances) used for testing. This allows for $\binom{3}{1} = \binom{3}{2} = 3$ independent test sets and the same number of training sets per corpus. Three disjoint pairings can be built from these sets. For each such pairing, a separate CRF model was trained from the training set and then applied to the corresponding test set. Various performance metrics were then averaged over the three runs. We will use $C_{COR\text{-}ALL}$ and $C_{RCG\text{-}ALL}$ to denote that training and testing has been performed as described above – using *all* 2007 instances, that is.

A confidence interval can also be estimated. If we assume the results of the three independent runs are normally distributed[1], a 95%-confidence interval is given by:

$$\bar{Y} \pm t_{(\alpha/2,N-1)} \frac{s}{\sqrt{N}} = \bar{Y} \pm t_{(0.025,2)} \frac{s}{\sqrt{3}} \tag{6.1}$$

where $\bar{Y}$ is the sample mean, s is the sample standard deviation, N is the sample size (3 in our case), α is the desired significance level (0.05 in our case) and $t_{(\alpha/2,N-1)}$ is the upper critical value of the t-distribution with $N-1$ degrees of freedom.

If a confidence interval is given in the following sections (indicated via $\pm$), it was computed as described above. Naturally, 10-fold cross-validation would have yielded even more reliable results, but it would have been prohibitively expensive.

6.1.1 Parameter Settings and Algorithmic Choices

Chapter 4 should have made it clear that for CRF training, there is a wealth of different algorithms and free parameters. Obviously, not all of these are supported by VieCRF; still, the number of options is enormous (just think of the combinatoric explosion). Given that CRF training is computationally expensive, extensive parameter sweeps are infeasible.

Therefore, experiments were only performed using the following reasonable settings, which were either motivated by practical limitations, experience or ad-hoc experiments[2]:

[1] This assumption should be rather safe since the performance metrics are computed over a large number of instances, cf. Central Limit Theorem (CLT).

[2] The author is painfully aware that this is less than optimal, but a more principled approach would have been extremely time-consuming.

- LBFGS was used for parameter optimization. While OLBFGS may have superior properties in some circumstances, the big number of free parameters made it unsuitable in this context. LBFGS was set to use the last 3 steps in parameter and gradient space for for estimation of the inverse Hessian (i.e., $m = 3$). Larger numbers require more memory and ad-hoc experiments did not indicate significantly faster convergence. The maximum number of iterations of LBFGS was set to 800. Progress became minuscule much earlier in most cases (see figures 6.1 and 6.2).
- Maximum Pseudolikelihood Estimation was performed. This was required in order to keep training times reasonable; pseudolikelihood does not require true inference and is therefore faster than MLE (by a large factor). In addition, it is not ridden by the convergence problems of loopy belief propagation.
- For testing, loopy belief propagation with a TRP schedule was used in order to determine the MAP configuration. This is the only option currently implemented by VieCRF. The algorithm was set to perform a maximum of 1000 iterations. For most instances, labeling converged much sooner (see section 6.8).
- *Supported features* were used for single-node clique templates, whereas in-chain and between-chains clique templates were restricted to the *default features* (see subsection 5.2.1). These settings were chosen in order to achieve reasonable dimensionality of the parameter space, thereby both reducing training time and danger of overfitting.
- A Gaussian prior with a variance of $\sigma^2 = 1000$ was used for regularization. This imposes a relatively weak penalty on extreme weights; the reasoning behind this setting was that it should keep the number of invalid label transitions low by assigning extremely low weights to transitions that do not occur in the training data. In practice, ad-hoc experiments did not show any significant difference between values of 1000 and 10 (see also subsection 6.1.2 on post-processing).

Figures 6.1 and 6.2 depict the progress of the loss function for training on $C_{COR\text{-}ALL}$ and $C_{RCG\text{-}ALL}$, respectively. The curve is averaged over all three training runs in both figures. Since one training process converged after roughly 400 iterations, the loss function is not depicted for higher iteration numbers in figure 6.2 (due to the average being undefined).

In general, what can be seen from these figures is that most of the loss is eliminated rather early; after that, progress happens slowly. Training on $C_{COR\text{-}ALL}$ runs a lot more smoothly than training on $C_{RCG\text{-}ALL}$, which is to be expected, given the greater variance and noise in the latter corpus. Also, the final relative loss is much lower for $C_{COR\text{-}ALL}$.

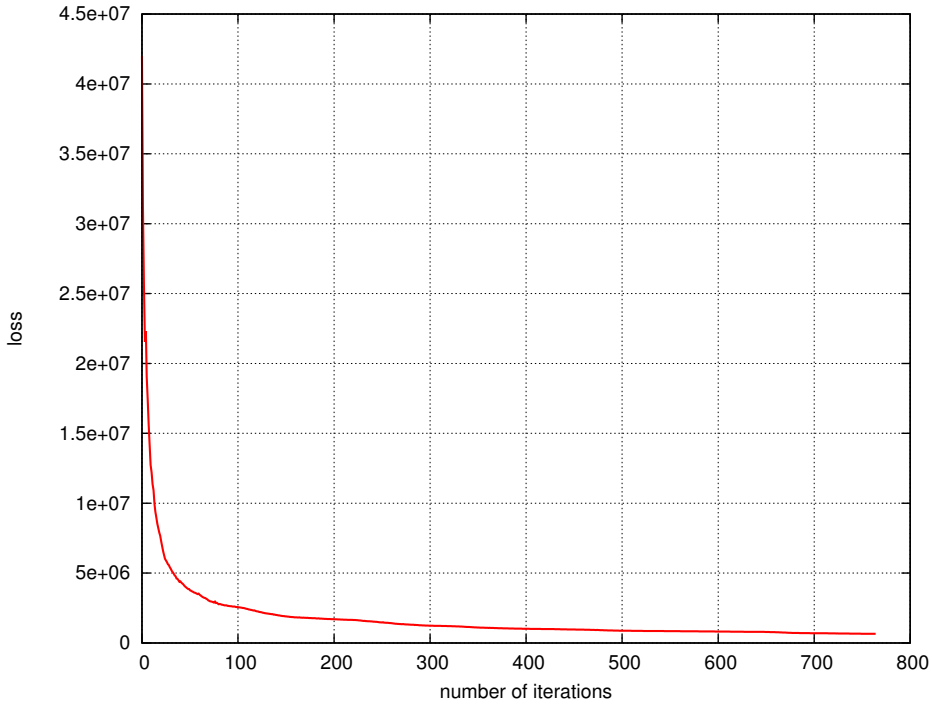

Figure 6.1: Progress of training on $C_{COR\text{-}ALL}$

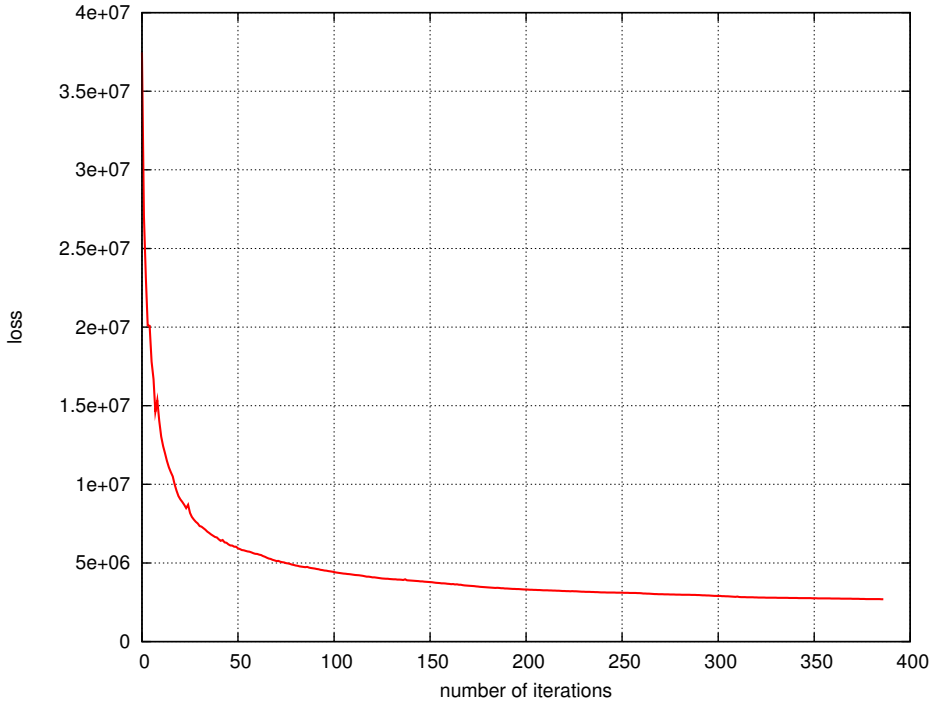

Figure 6.2: Progress of training on $C_{RCG\text{-}ALL}$

Algorithm 7 POST-PROCESS C_{COR}

1. Ensure every segment at the section level starts with a "Begin" label (this is a formal constraint).
2. Ensure boundaries of "typed" subsections occur only at sensible points (e.g., at the start of a subheading)
3. Ensure all segments at the subsection level start with a "Begin" label.
4. Make sure there are no "untyped" paragraphs in between "typed" subsections.
5. Ensure section boundaries occur only at the beginning of a heading.
6. Ensure every section segment starts with a "Begin" label.

On average, each training run on $C_{COR\text{-}ALL}$ required about 385 hours of CPU time and roughly 2.2 GB of RAM on an Intel® Core2Quad machine with four 2.4 GHz cores, resulting in an average training time of $385/4/24 \simeq 4$ days per run. An average training run on $C_{RCG\text{-}ALL}$ took only 3.15 days; this is due to one run converging early. It should be noted that 2.2 GB of RAM were only required for parallel training on all four cores; about 1.1 GB of RAM are sufficient for single-threaded training.

6.1.2 Post-Processing

Labeling of instances (i.e., computation of the MAP configuration) is performed within the CRF framework. However, it became evident that additional improvements could be achieved using further processing. Therefore, besides CRF-based label prediction, a post-processing mechanism was implemented that serves two purposes:

- Illegal label transitions (i.e., those that violate the typed `BIO` notation or simply do not make sense) can be flattened out in a sensible way.
- Additional domain knowledge can be considered without being restricted to the first-order Markov property of FCRF variables.

The first point is slightly delicate: the typed `BIO` notation employed for multi-level segmentation demands that any segment start with a `Begin` label, i.e., sequences like "...`-Plan-Plan-Diagnosis-Diagnosis-`..." do not have a meaningful interpretation in this notation and should be encoded as "...`-Plan-Plan-BeginDiagnosis-Diagnosis-`..." instead. Such constraints do not only exist for the vertical chains, but also horizontally: For example, any section of a report in C_{COR} starts with a heading; therefore we know that a `Begin` label may

Algorithm 8 POST-PROCESS C_{RCG}

1. Make sure section boundaries only occur at sensible points (e.g., at the start of a heading or the start of a paragraph).
2. Ensure every section segment starts with a "Begin" label.
3. If two section segments of the same type occur after each other, merge them in a smart way (e.g., let the new section segment start at the beginning of a heading).
4. Make sure every element at the subsection level starts with a "Begin" label.
5. Eliminate all inappropriate cases of two subsequent "Begin" labels at the subsection level.
6. Make sure every enumeration at the subsection level consists of at least two enumeration elements.
7. Ensure every segment at the sentence level starts with a "Begin" label.
8. Make sure a new sentence starts at every paragraph boundary.
9. Fix headings of excessive length at the sentence level (try to guess a more appropriate length or split the heading into two headings).

only occur on the section level if a `BeginHeading` label occurs at the same time step of the sentence level.

Such constraints could also be enforced by manipulating the relevant elements of the bivariate factors of an instance such that the transitions are effectively eliminated (see subsection 5.2.4). However, this proved to be impractical: First, it turned out that such manipulation of bivariate factors affected the convergence behavior of loopy belief propagation negatively. Second, it *did* enforce the formal constraints, but not in a meaningful way: for instance, many tokens on the section level were erroneously assigned a `BeginHeading` label so that a new segment could be started on the section level. This is the opposite of the actual goal, which is to prevent the beginning of a new section unless a heading starts at the sentence level. In these cases, it was cheaper to "push" one label towards being a heading than to convince multiple labels on the section level that they belong to the previous section.

Post-processing can be much more effective here: by recognizing that headings are identified very reliably, spurious section boundaries can simply be eliminated if a heading doesn't start at the same time step of the sentence level.

The second point is related, but could not be solved by manipulating bivariate factors anyway: Certain kinds of domain knowledge cannot be captured by a first-order Markov dependence. For instance, an enumeration list with only one enumeration item doesn't make sense. It

makes sense to require that an enumeration list must contain at least two enumeration items. Such constraints cannot be modelled within a first-order FCRF (and neither with a FCRF of second order, for that matter), since each enumeration item spans arbitrarily many tokens (and thus labels).

It should be noted that post-processing is slightly different for instances of C_{COR} and C_{RCG}. This is due to the greater noise of the latter corpus and the fact that structural elements like headings do not consistently occur in C_{RCG}; as such, they are less suitable for use as anchor points than those of C_{COR}. Algorithms 7 and 8 give a rough outline of the post-processing algorithms used for C_{COR} and C_{RCG}, respectively. The actual implementations contain some refined heuristics.

Admittedly, the approach of a separate post-processing component is not particularly elegant. However, from a pragmatic point of view, it is a better solution, compared to enforcing constraints within the CRF framework – at least until further research on the topic helps to get a grip on the convergence problems of inference and allows for more detailed control of how these constraints will be satisfied.

For the following evaluation results, unless indicated otherwise, it is assumed that post-processing has been performed before computing the performance metrics.

6.2 Estimated Accuracy

One particularly intuitive and often-used performance metric is $\mathrm{Accuracy}$. $\mathrm{Accuracy}$ can be estimated from a labeled test set as follows. We will use N to denote the total number of time steps in a label chain (over all instances), and $\mathrm{Correct}$ to denote the number of time steps that have been assigned a correct label. The $\mathrm{Accuracy}$ metric is then defined as

$$\mathrm{Accuracy} = \frac{\mathrm{Correct}}{\mathrm{N}} \tag{6.2}$$

This leads to the natural definition of Error:

$$\mathrm{Error} = 1 - \mathrm{Accuracy} \tag{6.3}$$

Table 6.1 shows estimated accuracies for $C_{COR\text{-}ALL}$, with and without post-processing. Accuracies are given for each label chain – *Chain 0* refers to the sentence level, *Chain 1* stands for the subsection level, and finally, *Chain 2* is used to denote the section level. In addition,

Estimated Accuracies		
	Accuracy	±
Average	91.19%	1.10
Chain 0	99.65%	0.04
Chain 1	91.50%	0.57
Chain 2	82.42%	2.80
Joint	77.30%	3.11

(a) without post-processing

Estimated Accuracies		
	Accuracy	±
Average	97.24%	0.33
Chain 0	99.64%	0.04
Chain 1	95.48%	0.55
Chain 2	96.61%	0.68
Joint	92.51%	0.97

(b) with post-processing

Table 6.1: Accuracy achieved on $C_{COR\text{-}ALL}$

Estimated Accuracies		
	Accuracy	±
Average	85.18%	0.97
Chain 0	91.75%	0.16
Chain 1	84.77%	0.64
Chain 2	79.00%	2.39
Joint	66.72%	2.16

(a) without post-processing

Estimated Accuracies		
	Accuracy	±
Average	86.36%	0.80
Chain 0	91.74%	0.16
Chain 1	85.90%	1.25
Chain 2	81.45%	2.14
Joint	69.19%	1.93

(b) with post-processing

Table 6.2: Accuracy achieved on $C_{RCG\text{-}ALL}$

Average indicates the accuracy averaged over all chains, and *Joint* gives the fraction of time steps with correctly assigned labels on all levels. Naturally, *Joint* $\leq$ *Average* holds. Table 6.2 presents the same numbers for $C_{RCG\text{-}ALL}$.

As is obvious from this table, post-processing is much more effective for $C_{COR\text{-}ALL}$; the reasons have been outlined in subsection 6.1.2. Still, post-processing also seems to be effective on $C_{RCG\text{-}ALL}$. In particular, the joint accuracy is improved by ensuring sensible label transitions horizontally.

Overall, high accuracy ($> 97\%$) can be achieved on $C_{COR\text{-}ALL}$. This is a reassuring result, as it shows that the approach works very well under ideal conditions. Unsurprisingly, accuracy on $C_{RCG\text{-}ALL}$ is much lower; however, at just above 86%, performance may still be good enough for real-life tasks (this certainly depends on the nature of errors, which will be discussed later). In addition, it is reasonable to assume that accuracy on $C_{RCG\text{-}ALL}$ could be even higher if the training set had been annotated manually.

It should be noted that $\mathrm{Accuracy}$ is not appropriate in all cases, since more prominent label

categories have bigger impact on the score. In particular, on our data, labels of *Chain 0/* sentence level are unequally distributed. `Sentence` labels account for the biggest part of all labels. Naturally, the baseline accuracy for `Sentence` is quite high. The accuracy achieved for `BeginSentence` is much more interesting in this context; yet, it has only little impact on the total accuracy due to the small number of occurrences. Other performance metrics such as F1 and WindowDiff may be more appropriate here; consequently, these will also be presented in this chapter.

6.3 Estimated Precision, Recall and F1

Precision, Recall and the harmonic mean of these two scores, F1[3], are performance metrics that are popular in information retrieval. They are typically computed on a per-label basis and thus allow for more fine-grained consideration of classification performance than Accuracy, for instance.

If we use $\mathrm{Correct}_y$ to denote how often label y has been correctly assigned, $\mathrm{Returned}_y$ to denote how often label y has been assigned in total (this includes correct and incorrect assignments) and N_y to denote how often label y *should* have been assigned, Precision, Recall and F1 are defined as follows:

$$\mathrm{Precision}_y = \frac{\mathrm{Correct}_y}{\mathrm{Returned}_y} \tag{6.4}$$

$$\mathrm{Recall}_y = \frac{\mathrm{Correct}_y}{\mathrm{N}_y} \tag{6.5}$$

$$\mathrm{F1}_y = \frac{2 \times \mathrm{Precision}_y \times \mathrm{Recall}_y}{\mathrm{Precision}_y + \mathrm{Recall}_y} \tag{6.6}$$

Often, it is useful to give a single number describing the performance of a classification algorithm. This can be done by averaging over the scores for all separate labels. Two different variants are in wide-spread use:

- **Macro-averaged F1** ($\mathrm{F1}_{mac}$) simply sums the F1 scores for each label and then divides that score by the number of distinct labels. The effect is that each label has equal influence, regardless of its relative frequency.

[3] More generally, the $\mathtt{F}_\alpha$ score allows for arbitrary weighting of Precision and Recall, but F1 is the most commonly used variant.

Label	N	Correct	Returned	P	R	F1
Subheading	3119	2926	2989	0.98	0.94	0.96
Enummarker	16740	16284	16507	0.99	0.97	0.98
BeginSubheading	20857	20449	20731	0.99	0.98	0.98
BeginHeading	21753	21471	21645	0.99	0.99	0.99
Heading	26245	25701	25985	0.99	0.98	0.98
BeginSentence	122082	120087	122079	0.98	0.98	0.98
Sentence	1187422	1186223	1188282	1.00	1.00	1.00
Total	**1398218**	**1393141**	**1398218**	**0.99**	**0.98**	**0.98**

Table 6.3: F1 for sentence level of $C_{COR\text{-}ALL}$ (with post-processing)

Label	N	Correct	Returned	P	R	F1
Subheading	6530	2771	5421	0.51	0.42	0.46
Enummarker	16031	9940	14388	0.69	0.62	0.65
BeginSubheading	16562	12543	15879	0.79	0.76	0.77
BeginHeading	18571	14278	19394	0.74	0.77	0.75
Heading	24736	17921	25266	0.71	0.72	0.72
BeginSentence	118065	71416	95079	0.75	0.60	0.67
Sentence	1050772	1019036	1075840	0.95	0.97	0.96
Total	**1251267**	**1147905**	**1251267**	**0.73**	**0.70**	**0.71**

Table 6.4: F1 for sentence level of $C_{RCG\text{-}ALL}$ (with post-processing)

- **Micro-averaged F1** ($\mathrm{F1}_{mic}$) first sums counts (N_y, $\mathrm{Correct}_y$, $\mathrm{Returned}_y$) for all labels and then calculates the score from these sums. This results in more frequent labels having more impact on the score.

It is easy to show that micro-averaged F1 is equal to Accuracy if each time step is assigned a label (which is the case in this scenario). If $\mathrm{N} = \mathrm{Returned}$, the following relations hold:

$$\frac{\mathrm{Correct}}{\mathrm{N}} = \mathrm{Precision}_{mic} = \frac{\mathrm{Correct}}{\mathrm{Returned}} = \mathrm{Recall}_{mic} \tag{6.7}$$

$$\frac{2 \times \mathrm{Precision}_{mic} \times \mathrm{Recall}_{mic}}{\mathrm{Precision}_{mic} + \mathrm{Recall}_{mic}} = \mathrm{F1}_{mic} = \mathrm{Precision}_{mic} = \mathrm{Recall}_{mic} \tag{6.8}$$

If we consider the definition of Accuracy (6.2), we see that it is equal to Recall_{mic}, and therefore equal to $\mathrm{F1}_{mic}$. □

For this reason, macro-averaged F1 scores will be given in this section.

A special case arises if $\mathrm{Returned}_y = 0$ for label y; in this corner case, Recall_y is undefined (division by zero). According to equation (6.6), F1 is then undefined, too. This situation will

be denoted using "N/A" in the results. The $\mathrm{F1}_{mac}$ score is averaged over *defined* $\mathrm{F1}_y$ scores only; the same applies to $\mathrm{Precision}_{mac}$ and Recall_{mac}.

Such denormalization usually only occurs for particularly infrequent labels; it is a strong indication that multiple labels should be collapsed into a single category, or that more balanced training data is required.

Tables 6.3 and 6.4 show $\mathrm{Precision}_y$ (P), Recall_y (R) and $\mathrm{F1}_y$ for all labels of the sentence level of $C_{COR\text{-}ALL}$ and $C_{RCG\text{-}ALL}$, respectively. Counts are computed over all three independent test sets. The bottom row of the tables show estimated $\mathrm{Precision}_{mac}$, Recall_{mac} and $\mathrm{F1}_{mac}$ for the sentence level.

High scores are consistently achieved for $C_{COR\text{-}ALL}$. This is expected, since C_{COR} assumes a scenario in which punctuation is dictated. The scores for $C_{RCG\text{-}ALL}$, on the other hand, leave quite a bit to be desired. First, this is due to punctuation being dictated only sporadically. Second, sentence boundaries are often not unambiguous, since some speakers tend to dictate in telegram style, which leaves room for different correct solutions. This means that some label assignments that are counted as errors (because the assigned label does not correspond to the reference annotation) are actually not so bad. Third, performance on the sentence level suffers particularly from semi-automatic reference annotation. Quite often, the reference labels are off by one or two time steps; this is an artifact of the relaxed alignment procedure. Finally, speech recognition may have got word boundaries wrong, so that the actual sentence boundary lies in the midst of a single token; obviously, the reference annotation of such cases is a bit arbitrary.

$\mathrm{WindowDiff}$ is a metric which properly recognizes that boundaries which are off by a small number of time steps are better than no boundary at all. $\mathrm{WindowDiff}$ results are given in section 6.5; indeed, $C_{RCG\text{-}ALL}$ fares quite a bit better in that comparison.

$\mathrm{F1}$ scores for the subsection level of $C_{COR\text{-}ALL}$ and $C_{RCG\text{-}ALL}$ are given in tables 6.7 and 6.8. For brevity, only `Begin` labels are listed; these are the most interesting ones anyway. The first thing the observant reader will notice is that in both tables, there is a number of labels with extremely low counts. Performance is unsatisfying for such infrequent labels; in fact, for both $C_{COR\text{-}ALL}$ and $C_{RCG\text{-}ALL}$, there is a clear trend that better scores are achieved for more frequent labels. It may therefore be appropriate to reduce the number of distinct labels, perhaps by building sensible clusters. These labels of low frequency also heavily impact the macro-averaged scores.

In general, it can be noted that the scores of $C_{COR\text{-}ALL}$ are still quite high (especially if one considers the greater number of distinct labels); the scores achieved on $C_{RCG\text{-}ALL}$, on

Label	N	Correct	Returned	P	R	F1
BeginBrain	3	0	0	N/A	0.00	N/A
BeginImmunologic	3	0	0	N/A	0.00	N/A
BeginSpeech	4	0	0	N/A	0.00	N/A
BeginPeripheralVascular	4	1	1	1.00	0.25	0.40
BeginStation	18	1	3	0.33	0.06	0.10
BeginEars	40	31	32	0.97	0.78	0.86
BeginCoordination	41	16	22	0.73	0.39	0.51
BeginReflexes	48	24	30	0.80	0.50	0.62
BeginCranialNerves	57	50	53	0.94	0.88	0.91
BeginSensory	59	24	32	0.75	0.41	0.53
BeginMotor	73	28	35	0.80	0.38	0.52
BeginBreasts	76	61	66	0.92	0.80	0.86
BeginNoseAndSinuses	81	65	70	0.93	0.80	0.86
BeginPelvic	98	82	89	0.92	0.84	0.88
BeginMouthAndThroat	116	102	113	0.90	0.88	0.89
BeginMentalStatus	119	102	109	0.94	0.86	0.89
BeginPsychiatric	141	131	133	0.98	0.93	0.96
BeginLymphNodes	144	126	132	0.95	0.88	0.91
BeginBack	155	146	146	1.00	0.94	0.97
BeginEyes	185	179	180	0.99	0.97	0.98
BeginHematologic	204	189	192	0.98	0.93	0.95
BeginAnoRectal	292	279	282	0.99	0.96	0.97
BeginEndocrine	325	319	321	0.99	0.98	0.99
BeginSkin	407	392	399	0.98	0.96	0.97
BeginHead	462	440	445	0.99	0.95	0.97
BeginRespiratory	503	487	495	0.98	0.97	0.98
BeginMusculoskeletal	539	520	527	0.99	0.96	0.98
BeginCardiac	539	523	529	0.99	0.97	0.98
BeginGenitourinary	554	540	546	0.99	0.97	0.98
BeginGastrointestinal	588	581	584	0.99	0.99	0.99
BeginNeck	1427	1410	1423	0.99	0.99	0.99
BeginHEENT	1448	1402	1429	0.98	0.97	0.97
BeginVitalSigns	1572	1565	1568	1.00	1.00	1.00
BeginNeurologic	1586	1538	1576	0.98	0.97	0.97
BeginExtremities	1627	1594	1612	0.99	0.98	0.98
BeginAbdomen	1670	1647	1658	0.99	0.99	0.99
BeginCardiovascular	1713	1667	1686	0.99	0.97	0.98
BeginThoraxAndLungs	1744	1698	1720	0.99	0.97	0.98
BeginGeneral	2062	2046	2060	0.99	0.99	0.99
BeginEnumelement	16740	16254	16588	0.98	0.97	0.98
BeginParagraph	21687	17851	21299	0.84	0.82	0.83
BeginNone	21753	21474	21648	0.99	0.99	0.99
...	...	...	...	...	...	...
Total	**1398218**	**1335024**	**1398218**	**0.90**	**0.77**	**0.85**

Table 6.5: F1 for subsection level of $C_{COR\text{-}ALL}$ (with post-processing)

Label	N	Correct	Returned	P	R	F1
BeginBrain	3	0	0	N/A	0.00	N/A
BeginImmunologic	3	0	0	N/A	0.00	N/A
BeginSpeech	4	0	0	N/A	0.00	N/A
BeginPeripheralVascular	4	0	0	N/A	0.00	N/A
BeginStation	18	2	6	0.33	0.11	0.17
BeginEars	32	11	23	0.48	0.34	0.40
BeginCoordination	41	3	14	0.21	0.07	0.11
BeginReflexes	48	8	29	0.28	0.17	0.21
BeginCranialNerves	57	7	59	0.12	0.12	0.12
BeginSensory	59	13	35	0.37	0.22	0.28
BeginMotor	73	9	62	0.15	0.12	0.13
BeginBreasts	74	31	39	0.79	0.42	0.55
BeginNoseAndSinuses	75	23	59	0.39	0.31	0.34
BeginPelvic	98	27	54	0.50	0.28	0.36
BeginMouthAndThroat	115	61	99	0.62	0.53	0.57
BeginMentalStatus	119	37	128	0.29	0.31	0.30
BeginPsychiatric	132	93	111	0.84	0.70	0.77
BeginBack	133	65	96	0.68	0.49	0.57
BeginLymphNodes	143	65	91	0.71	0.45	0.56
BeginEyes	173	95	152	0.62	0.55	0.58
BeginHematologic	190	103	149	0.69	0.54	0.61
BeginAnoRectal	291	214	279	0.77	0.74	0.75
BeginEndocrine	311	208	272	0.76	0.67	0.71
BeginSkin	390	258	371	0.70	0.66	0.68
BeginHead	454	356	439	0.81	0.78	0.80
BeginRespiratory	486	319	480	0.66	0.66	0.66
BeginMusculoskeletal	519	370	501	0.74	0.71	0.73
BeginCardiac	525	383	540	0.71	0.73	0.72
BeginGenitourinary	529	366	513	0.71	0.69	0.70
BeginGastrointestinal	566	424	589	0.72	0.75	0.73
BeginNeck	1410	1143	1463	0.78	0.81	0.80
BeginHEENT	1427	1062	1544	0.69	0.74	0.71
BeginNeurologic	1549	1114	1763	0.63	0.72	0.67
BeginVitalSigns	1550	1173	1650	0.71	0.76	0.73
BeginExtremities	1601	1285	1747	0.74	0.80	0.77
BeginAbdomen	1647	1383	1718	0.81	0.84	0.82
BeginCardiovascular	1695	1283	1732	0.74	0.76	0.75
BeginThoraxAndLungs	1725	1353	1771	0.76	0.78	0.77
BeginGeneral	2008	1587	2278	0.70	0.79	0.74
BeginEnumelement	16192	9332	14162	0.66	0.58	0.61
BeginNone	18571	14336	19554	0.73	0.77	0.75
BeginParagraph	20542	13109	23146	0.57	0.64	0.60
...	...	...	...	...	...	...
Total	**1251267**	**1074773**	**1251267**	**0.62**	**0.52**	**0.59**

Table 6.6: F1 for subsection level of $C_{RCG\text{-}ALL}$ (with post-processing)

the other hand, indicate that usable performance can only be achieved for the more frequent labels.

Finally, tables 6.7 and 6.8 give the results for the section level of $C_{COR\text{-}ALL}$ and $C_{RCG\text{-}ALL}$. The situation is a bit better here compared to the subsection level; however, again, there are labels that occur hardly at all in the corpora. The results also indicate that some types of sections can be identified much more reliably than others. Unsurprisingly, the scores achieved on $C_{COR\text{-}ALL}$ are quite a bit higher; this can mostly be attributed to the fact that headings are dictated more consistently, and that there aren't any speech recognition errors which obfuscate dictated headings.

Overall, the macro-averaged $\mathrm{F1}$ scores may suggest worse performance than is actually the case; label prediction works pretty well for those labels that do occur frequently in the corpora. This is also reflected in the relatively high $\mathrm{Accuracy}$ scores. The findings do indicate, however, that the set of labels may have to be reduced in a sensible way, or that more training data is needed.

Label	N	Correct	Returned	P	R	F1
BeginPrognosis	1	0	0	N/A	0.00	N/A
BeginAdvanceDirectives	1	0	0	N/A	0.00	N/A
BeginNotes	1	0	0	N/A	0.00	N/A
Prognosis	2	0	0	N/A	0.00	N/A
BeginCourse	6	0	0	N/A	0.00	N/A
BeginProcedures	8	2	2	1.00	0.25	0.40
Notes	11	0	0	N/A	0.00	N/A
BeginTime	30	22	24	0.92	0.73	0.81
AdvanceDirectives	62	0	0	N/A	0.00	N/A
BeginFindings	77	65	67	0.97	0.84	0.90
BeginNeurologic	120	95	102	0.93	0.79	0.86
BeginChiefComplaints	127	121	122	0.99	0.95	0.97
Course	209	0	0	N/A	0.00	N/A
Time	299	261	777	0.34	0.87	0.49
BeginHabits	308	273	284	0.96	0.89	0.92
BeginPractitioner	488	469	472	0.99	0.96	0.98
Procedures	501	94	94	1.00	0.19	0.32
BeginDiagnosisAndPlan	519	454	513	0.88	0.87	0.88
BeginPastSurgicalHistory	964	927	938	0.99	0.96	0.97
BeginPlan	1171	1061	1105	0.96	0.91	0.93
ChiefComplaints	1240	726	730	0.99	0.59	0.74
BeginReviewOfSystems	1496	1477	1493	0.99	0.99	0.99
BeginDiagnosticStudies	1532	1461	1497	0.98	0.95	0.96
BeginMedication	1541	1509	1525	0.99	0.98	0.98
BeginAllergies	1542	1520	1529	0.99	0.99	0.99
BeginDiagnosis	1619	1577	1619	0.97	0.97	0.97
BeginReasonForEncounter	1646	1630	1661	0.98	0.99	0.99
BeginHistoryOfPresentIllness	1898	1868	1896	0.99	0.98	0.98
BeginPhysicalExamination	1948	1941	1945	1.00	1.00	1.00
BeginPastHistory	3074	3006	3123	0.96	0.98	0.97
Findings	3123	2495	2711	0.92	0.80	0.86
Practitioner	4334	3966	4402	0.90	0.92	0.91
Habits	7571	6903	7314	0.94	0.91	0.93
Allergies	8361	8159	8914	0.92	0.98	0.94
PastSurgicalHistory	16646	15982	16786	0.95	0.96	0.96
Neurologic	18383	15810	16505	0.96	0.86	0.91
ReasonForEncounter	42327	41734	44486	0.94	0.99	0.96
Medication	44595	43967	44899	0.98	0.99	0.98
DiagnosisAndPlan	65290	57905	65851	0.88	0.89	0.88
ReviewOfSystems	85744	84278	85040	0.99	0.98	0.99
Plan	104386	93373	96522	0.97	0.89	0.93
DiagnosticStudies	105764	101361	106275	0.95	0.96	0.96
Diagnosis	123488	116736	126298	0.92	0.95	0.93
PastHistory	127348	125161	128131	0.98	0.98	0.98
PhysicalExamination	255692	254475	259970	0.98	1.00	0.99
HistoryOfPresentIllness	362725	357932	362596	0.99	0.99	0.99
Total	**1398218**	**1350796**	**1398218**	**0.95**	**0.73**	**0.90**

Table 6.7: F1 for section level of $C_{COR\text{-}ALL}$ (with post-processing)

Label	N	Correct	Returned	P	R	F1
Prognosis	1	0	0	N/A	0.00	N/A
BeginPrognosis	1	0	0	N/A	0.00	N/A
BeginAdvanceDirectives	1	0	0	N/A	0.00	N/A
BeginCourse	6	0	1	0.00	0.00	N/A
BeginTime	8	0	0	N/A	0.00	N/A
BeginProcedures	8	0	0	N/A	0.00	N/A
AdvanceDirectives	37	0	0	N/A	0.00	N/A
BeginFindings	77	42	55	0.76	0.55	0.64
BeginChiefComplaints	119	83	89	0.93	0.70	0.80
BeginNeurologic	120	37	110	0.34	0.31	0.32
BeginPractitioner	145	85	98	0.87	0.59	0.70
Time	154	0	0	N/A	0.00	N/A
Course	198	0	14	0.00	0.00	N/A
BeginHabits	303	129	267	0.48	0.43	0.45
Procedures	473	0	0	N/A	0.00	N/A
BeginDiagnosisAndPlan	520	239	586	0.41	0.46	0.43
BeginPastSurgicalHistory	947	668	948	0.70	0.71	0.71
ChiefComplaints	977	495	1286	0.38	0.51	0.44
BeginPlan	1169	712	1547	0.46	0.61	0.52
Practitioner	1234	318	2030	0.16	0.26	0.19
BeginReasonForEncounter	1338	1135	1261	0.90	0.85	0.87
BeginReviewOfSystems	1480	1126	1585	0.71	0.76	0.73
BeginAllergies	1516	1099	1523	0.72	0.72	0.72
BeginMedication	1517	1128	1600	0.70	0.74	0.72
BeginDiagnosticStudies	1520	1016	1680	0.60	0.67	0.64
BeginDiagnosis	1612	1204	1730	0.70	0.75	0.72
BeginHistoryOfPresentIllness	1879	1426	2144	0.67	0.76	0.71
BeginPhysicalExamination	1943	1420	1999	0.71	0.73	0.72
Findings	2810	1714	2400	0.71	0.61	0.66
BeginPastHistory	3042	2156	3463	0.62	0.71	0.66
Habits	6404	3708	6012	0.62	0.58	0.60
Allergies	8310	5927	11043	0.54	0.71	0.61
PastSurgicalHistory	15105	12034	17034	0.71	0.80	0.75
Neurologic	17540	9505	13850	0.69	0.54	0.61
ReasonForEncounter	36083	10389	22814	0.46	0.29	0.35
Medication	39580	35419	43582	0.81	0.89	0.85
DiagnosisAndPlan	63944	33904	51909	0.65	0.53	0.59
ReviewOfSystems	73828	65456	73710	0.89	0.89	0.89
DiagnosticStudies	90764	81265	101055	0.80	0.90	0.85
Plan	97415	75320	104530	0.72	0.77	0.75
PastHistory	112300	97538	130032	0.75	0.87	0.80
Diagnosis	116052	82103	109528	0.75	0.71	0.73
PhysicalExamination	209832	187954	204178	0.92	0.90	0.91
HistoryOfPresentIllness	338955	302438	335574	0.90	0.89	0.90
Total	**1251267**	**1019192**	**1251267**	**0.63**	**0.52**	**0.66**

Table 6.8: F1 for section level of $C_{RCG\text{-}ALL}$ (with post-processing)

6.4 Confusion

Apart from the fraction of errors, the nature of these errors may also be of interest. In particular, it may be enlightening to study the *confusion* of labels, i.e., which labels y' are typically predicted instead of a given reference label y.

Such analysis is particularly useful for the section level of our corpora. The section level is completely dissected into a number of typed sections; a wrongly assigned label means here that a token has been attributed to a wrong section type. We would then like to know which section types are typically confused; this may be helpful in determining a better set of section types and allows for further insight into the typical contents of sections.

For this purpose, confusion plots were drawn for both $C_{COR\text{-}ALL}$ and $C_{RCG\text{-}ALL}$ (see figures 6.3 and 6.4). The dots in these plots indicate how often label y' given on the y-axis has been predicted instead of reference label y given on the x-axis. The bigger the dot, the higher the number of confusions. If an algorithm works fairly well, the dots in the diagonal will usually be the biggest ones (since the correct label will have been assigned most of the time).

The plots for the two corpora are rather similar, except that on $C_{RCG\text{-}ALL}$, a lot more errors occurred, so a slightly lower part of the mass is distributed over the diagonal. However, the prominent confusions are typically the same as on $C_{COR\text{-}ALL}$, with the only difference being that they are even more frequent.

Very commonly, `DiagnosisAndPlan` is confused with `Diagnosis` or `Plan` (and vice versa). This is not particularly surprising: Ideally, the `DiagnosisAndPlan` label would not even exist; however, some practitioners tend to dictate such a combined section rather than separate `Diagnosis` and `Plan` sections. Typically, the contents is intermingled, so it's not possible to simply split such sections either.

Another frequent mistake is the confusion of sections of type `HistoryOfPresentIllness` with `ReasonForEncounter` (and the other way around). These section types are rather similar with regards to the vocabulary that is used, and, furthermore, they usually occur right after each other. This makes it hard to find the proper section boundary if headings are not dictated. What makes the task even more difficult is that not all speakers correctly distinguish between these section types, thereby introducing inconsistencies in the training data.

Finally, `Neurologic` is often confused with `PhysicalEximination`, which is due to the fact that some speakers prefer to make `Neurologic` a section of its own, whereas others

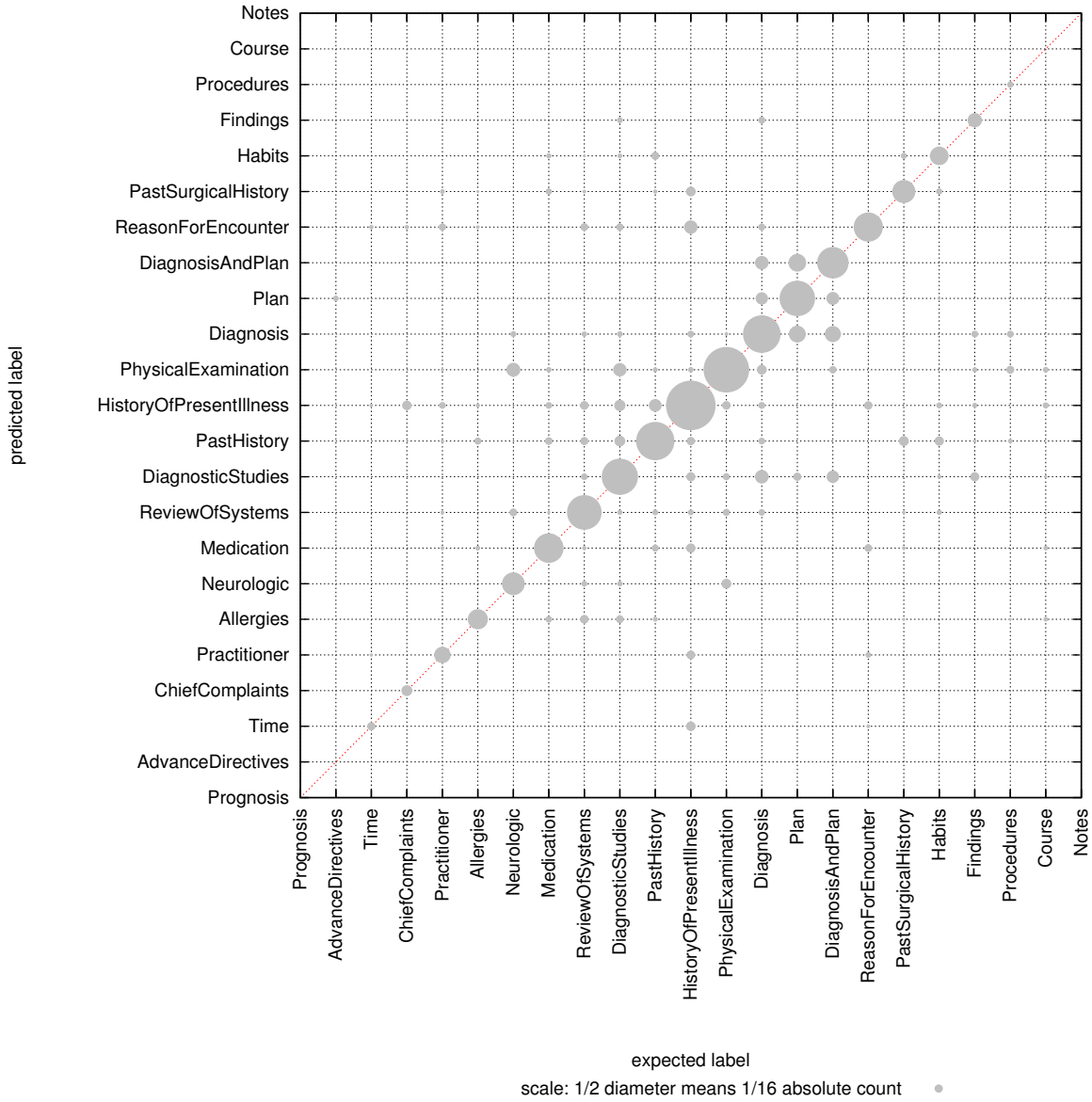

Figure 6.3: Confusion plot for section level of $C_{COR\text{-}ALL}$ (with post-processing)

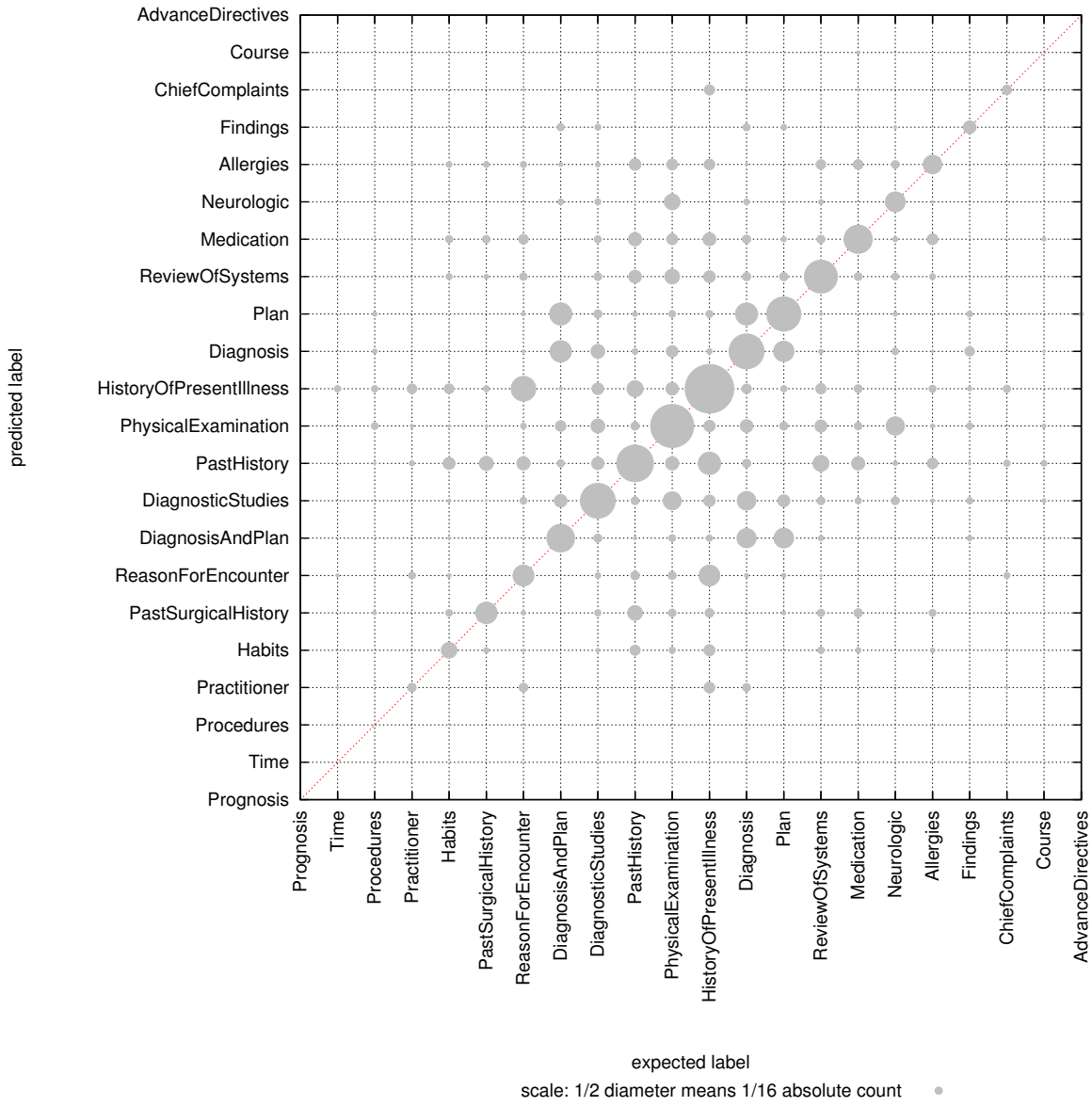

Figure 6.4: Confusion plot for section level of $C_{RCG\text{-}ALL}$ (with post-processing)

dictate it as a subsection of the `PhysicalExamination` section. Obviously, this reduces the discriminative effect of BOW features.

In general, most confusions seem to be rather intuitive. The section types that are typically confused with each other really do bear a lot of resemblance. This is a reassuring result, since it shows that most errors on the section level are not grave; rather, the "correct" type of a section may even be debatable at times.

6.5 Estimated WindowDiff

WindowDiff is an evaluation metric for segmentation algorithms. It measures how accurately segment boundaries are determined. Two compelling properties of WindowDiff are that "near misses" are penalized less strongly, and that "false positives" and "false negatives" are assigned equal penalties.

WindowDiff is defined by Pevzner and Hearst as follows (see [PH02]):

$$\text{WindowDiff}(ref, hyp) = \frac{1}{N-k}\sum_{i=1}^{N-k}(|b(ref_i, ref_{i+k}) - b(hyp_i, hyp_{i+k})| > 0) \qquad (6.9)$$

where $b(i, j)$ represents the number of boundaries between positions i and j in the text, N represents the number of "sentences" in the text and k is half the average length of a "sentence" in the reference annotation.

The notion of "sentences" in their definition is dependent on the task and should not be confused with natural language sentences. Rather, they denote atomic items that are grouped into segments by a segmentation algorithm. In our case, these atomic items are tokens, and they are grouped into sections, subsections, paragraphs, and such (depending on the level).

WindowDiff works as follows: for each position of the probe, it compares how many reference segmentation boundaries fall in this interval ($b(ref_i, ref_{i+k})$), versus how many boundaries are assigned by the algorithm ($b(hyp_i, hyp_{i+k})$); the algorithm is penalized if these numbers differ. WindowDiff is normalized such that it will return a number between 0 and 1, with 0 indicating perfect segmentation and 1 indicating the worst possible algorithm.

Table 6.9 shows the WindowDiff scores for all levels of $C_{COR\text{-}ALL}$, with and without post-processing. Table 6.10 gives the same numbers for $C_{RCG\text{-}ALL}$. It should be noted that WindowDiff does not take into account the type of a segment (i.e., the concrete label); only

Estimated WindowDiff		
	WindowDiff	±
Sentence Level	0.007	0.000
Subsection Level	0.052	0.007
Section Level	0.036	0.007

(a) without post-processing

Estimated WindowDiff		
	WindowDiff	±
Sentence Level	0.007	0.000
Subsection Level	0.050	0.007
Section Level	0.015	0.001

(b) with post-processing

Table 6.9: WindowDiff for $C_{COR\text{-}ALL}$

Estimated WindowDiff		
	WindowDiff	±
Sentence Level	0.193	0.008
Subsection Level	0.148	0.002
Section Level	0.126	0.013

(a) without post-processing

Estimated WindowDiff		
	WindowDiff	±
Sentence Level	0.193	0.008
Subsection Level	0.149	0.005
Section Level	0.118	0.013

(b) with post-processing

Table 6.10: WindowDiff for $C_{RCG\text{-}ALL}$

the position of boundaries is relevant. In order to compute WindowDiff for the levels of the two corpora, any label starting with `Begin` was assumed to denote a boundary; the trivial boundaries at the very beginning of an instance were not counted.

First, it can be noted that post-processing does not seem to affect the WindowDiff scores strongly, with the notable exception of the section level. In particular on $C_{COR\text{-}ALL}$, the WindowDiff score of the section level is improved substantially by post-processing; on this corpus, post-processing can eliminate spurious section boundaries by relying on headings.

Overall, the scores are quite good, even for $C_{RCG\text{-}ALL}$ – especially compared to the low macro-averaged F1 scores. Part of this is due to the fact that WindowDiff does not account for type confusions; the other part can be attributed to WindowDiff penalizing "near misses" less strongly. The WindowDiff scores do not suffer as badly from inaccurate reference annotation for this reason. It is hard to specify a concrete threshold, but WindowDiff scores consistently below 0.2 certainly do promise usable segmentation quality for the purposes of this thesis.

Estimated Accuracies		
	Accuracy	±
Average	90.28%	0.51
Chain 0	99.58%	0.08
Chain 1	90.13%	0.67
Chain 2	81.13%	1.70
Joint	75.38%	1.25

(a) without post-processing

Estimated Accuracies		
	Accuracy	±
Average	96.48%	0.82
Chain 0	99.55%	0.08
Chain 1	94.64%	0.23
Chain 2	95.25%	2.16
Joint	90.65%	2.15

(b) with post-processing

Table 6.11: Accuracy achieved on $C_{COR\text{-}BEST}$

Estimated Accuracies		
	Accuracy	±
Average	86.04%	1.75
Chain 0	93.80%	0.66
Chain 1	86.08%	1.59
Chain 2	78.24%	3.24
Joint	67.11%	3.74

(a) without post-processing

Estimated Accuracies		
	Accuracy	±
Average	87.73%	2.07
Chain 0	93.77%	0.68
Chain 1	87.59%	1.79
Chain 2	81.81%	3.79
Joint	70.91%	4.50

(b) with post-processing

Table 6.12: Accuracy achieved on $C_{RCG\text{-}BEST}$

6.6 The Effect of Noisy Training Data

Measuring the effect of the imprecise reference annotation of C_{RCG} is a difficult task; in fact, without corresponding manual annotation, it is impossible to give exact numbers.

However, a simple experiment may serve to give a rough feeling of the impact on prediction accuracy: A subset of C_{RCG} was compiled which comprises the 1002 reports that yielded the lowest Word Error Rate (WER) when aligned with their counterparts of C_{COR}. This subset will be referred to as $C_{RCG\text{-}BEST}$, while the subset of corresponding reports from C_{COR} will be called $C_{COR\text{-}BEST}$. Accuracy was then determined for $C_{COR\text{-}BEST}$ and $C_{RCG\text{-}BEST}$ using three-fold cross-validation as previously described for $C_{COR\text{-}ALL}$ and $C_{RCG\text{-}ALL}$.

The results of this experiment are shown in tables 6.11 and 6.12. If these numbers are compared to the Accuracy scores achieved on $C_{COR\text{-}ALL}$ and $C_{RCG\text{-}ALL}$, one can see that overall accuracy *decreased* for $C_{COR\text{-}BEST}$, whereas we see an *increase* for $C_{RCG\text{-}BEST}$. This effect can be attributed to two different phenomena:

- Since $C_{RCG\text{-}BEST}$ contains those reports for which alignment worked particularly well, it is safe to assume that the reference annotation created for these reports is better than that of an average report of $C_{RCG\text{-}ALL}$. This helps parameter estimation by decreasing noise and reduces the number of label predictions that are erroneously counted as a misprediction.
- Part of the effect, however, must also be attributed to the fact that the reports in $C_{RCG\text{-}BEST}$ typically contain fewer speech recognition errors (which, in turn, facilitated alignment).

Sadly, it is not possible to isolate these effects; still, the experiment shows that proper manual annotation of C_{RCG} may result in significant performance gains.

6.7 Further Analysis of Training Progress

Subsection 6.1.1 showed plots indicating the progress of the loss function during training (see figures 6.1 and 6.2). The parameter optimization algorithm (e.g., LBFGS) is typically iterated until there is no further progress.

The actual goal, however, is not to minimize the loss function (which is computed over the training set), but to maximize some performance metric over the test set. The loss function is usually chosen such that one can expect it to behave similar to the performance metric; i.e., by minimizing the loss function, the score of the performance metric will be optimized. This need not be the case in general. For instance, without regularization, it may happen that the performance on the test set starts decreasing after some point, because overfitting sets in.

An experiment was performed that compares the progress of the $\mathrm{Accuracy}$ metric on a validation set to the progress of the loss function (negative penalized conditional log-likelihood, in this case) on a training set. The training and validation sets were compiled as follows: the instances in C_{COR} were shuffled first; then the first third (669 instances) was taken for the validation set, whereas the remaining two thirds formed the training set. These two sets will be referred to as $C_{COR\text{-}VAL}$. The same process was applied to C_{RCG}, resulting in $C_{RCG\text{-}VAL}$

For both corpora, a single training run was performed on the training set; $\mathrm{Accuracy}$ on the validation set was computed using the intermediate CRF parameters $\boldsymbol{\theta}_t$ every 5 iterations of LBFGS. The progress of the loss function was then plotted against the progress of the $\mathrm{Accuracy}$ metric. Figures 6.5 and 6.6 show these plots.

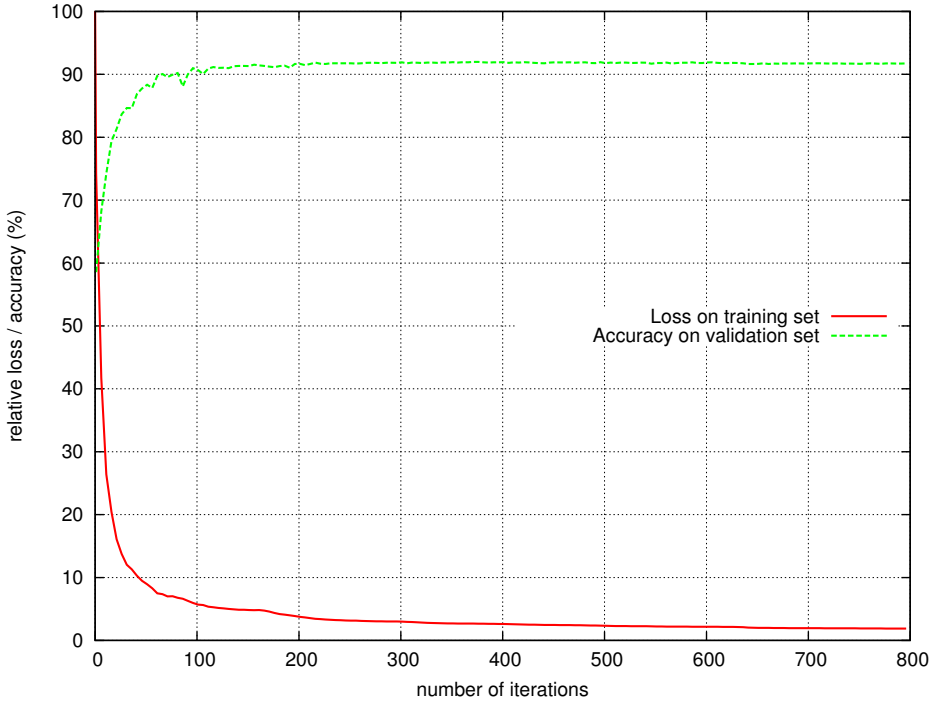

Figure 6.5: Progress of accuracy vs. loss function on $C_{COR\text{-}VAL}$

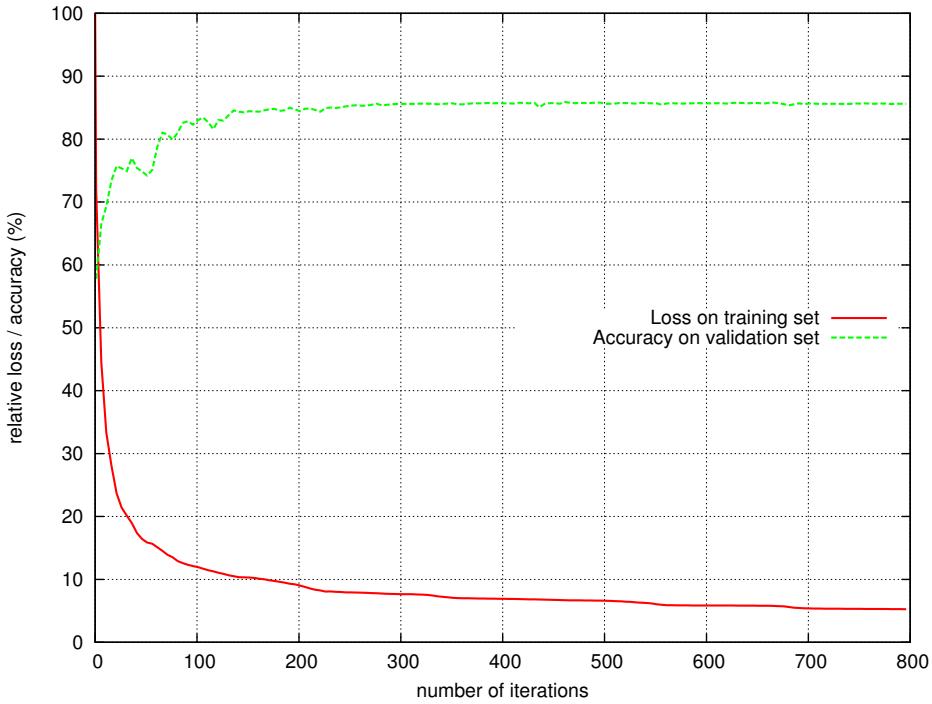

Figure 6.6: Progress of accuracy vs. loss function on $C_{RCG\text{-}VAL}$

	Converged (%)	**Iterations (∅)**
$C_{COR\text{-}ALL}$	0.999	15.4
$C_{RCG\text{-}ALL}$	0.911	66.5
$C_{COR\text{-}BEST}$	0.999	14.2
$C_{RCG\text{-}BEST}$	0.971	37.5

Table 6.13: Convergence Behavior of TRP

The plots demonstrate that the progress of the loss function corresponds well to that of the Accuracy metric, although the latter tends to flatten out quite a bit sooner. It might be tempting to interrupt training well before the loss function has reached its minimum if there is no more progress in performance on a validation set; this approach is called "early stopping" and is often applied in the neural networks community in order to avoid overfitting. However, the regularization applied to parameter estimation of CRFs is a more principled approach which protects against overfitting anyway (it does not cut down on training time, though). Furthermore, the author noticed that CRF parameters seem to be strongly biased towards labels of high frequency during earlier stages of training; this is sufficient for reaching a good Accuracy score, but other metrics such as macro-averaged F1 may suffer.

6.8 Preliminary Analysis of Convergence Behavior

Subsection 4.3.2 revealed that loopy belief propagation, the inference algorithm typically used for determining marginals and the MAP configuration, is not guaranteed to converge in all cases. While this limitation could be avoided during CRF training (by maximizing pseudolikelihood instead of "true" likelihood), it became apparent when labeling instances of the test sets.

As indicated in subsection 6.1.1, instances were labeled using TRP, a particular schedule for loopy belief propagation. The TRP implementation was set to perform a maximum of 1000 iterations (which is more than enough in most cases). Table 6.13 shows the percentage of instances for which TRP converged (i.e., required fewer than 1000 iterations), as well as the average number of iterations required *if* TRP converged. Both numbers are given for $C_{COR\text{-}ALL}$, $C_{RCG\text{-}ALL}$, $C_{COR\text{-}BEST}$ and $C_{RCG\text{-}BEST}$.

The first thing which stands out is that the convergence behavior of TRP is much worse on the C_{RCG} corpora than on their C_{COR} counterparts. The reasons for this behavior are not completely clear; presumably, the greater noise in the C_{RCG} corpora, along with sometimes

contradictory reference annotation, have negative impact on convergence. Preliminary observations by the author also seem to indicate that convergence is related to how similar instances in the training and test sets are. In general, variance is much greater in the C_{RCG} corpora, so chances are bigger that some instances in the test sets are unlike any instances in the training sets.

Another interesting observation is that convergence typically occurs much sooner on $C_{RCG\text{-}BEST}$ than on $C_{RCG\text{-}ALL}$. This supports the assumption that erroneous reference annotation negatively affects the convergence behavior of TRP on unseen instances; however, the lower rate of speech recognition errors in $C_{RCG\text{-}BEST}$ may also contribute to this effect.

Chapter 7

Conclusion and Outlook

> *“The best way to predict the future is to invent it.”*
>
> *– Alan Kay*

A lot has been achieved throughout the course of this thesis: The experiments performed in the previous section indicate that the presented approach may indeed prove to be applicable in practice. However, some refinement may be required: In particular, the semi-automatic annotation process described in section 3.3 results in inferior performance, compared to a manually annotated corpus. This was to be expected, though, and overall, the estimated accuracy is still sufficient for a wide area of applications. Furthermore, the experiments indicate that better performance can be achieved by using a coarser (and more sensible) set of section and subsection types.

The actual practicability of the framework for further processing of dictated reports remains to be seen. Early experiments are promising, but manual improvement of the semi-automatically created annotation might become necessary in order to straighten out systematic errors. In general, most of what is required is in place and works sufficiently well. Still, there is certainly room for further in-depth analysis and new experiments. Some of these potential tasks will be presented in this chapter; they can be broken down into two distinct categories:

- First, there are some remaining challenges with regards to structure identification; these should be analyzed and remedied, if necessary.
- Second, as mentioned in the introduction, *identification* of structure is only the beginning. The resulting information needs to be put to use in order to create reports that are properly arranged and formatted.

7.1 Remaining Challenges

While the approach presented in this thesis works fine in general, there are still some rough edges that may have to be smoothed out; additionally, some investigations that could yield interesting results have not been performed thus far:

- Further analysis of convergence behavior could yield interesting insights. In particular, it would be useful to establish a catalog of sufficient criteria for rapid convergence of loopy belief propagation. Such results are desirable both from a theoretic and a practical point of view. Alternatively, the use of other (ideally approximate) inference algorithms with guaranteed convergence properties should be explored.
- More principled feature selection may be indicated. In particular, the influence of features derived from UMLS should be studied thoroughly. If the same performance could be achieved without using UMLS, this large and resource-intensive knowledge source could be abandoned. Furthermore, it may be helpful to explore the use of more advanced topic modeling techniques, such as latent Dirichlet allocation (see [Wal06]). Alternatively, class probabilities as determined by a separate local classifier, such as a SVM-based model, might be used as input features for the CRF model.
- The post-processing algorithms that are currently applied after CRF-based labeling do work well, but they are inelegant from a theoretic point of view. Further research on constrained CRF inference may provide more satisfying results.
- It may also be interesting to compare the performance achieved by training on a semi-automatically annotated corpus to that of a manually annotated corpus, once (*if*) such a corpus becomes available in the future.
- Another promising attempt would be to work on the word graph produced by Automatic Speech Recognition (ASR), rather than on the single best path only. In particular, this may help for reports with high error rates, assuming other paths in the word graph contain correct solutions.
- Finally, determining the M best label configurations is not currently supported. A suitable algorithm has been proposed by Yanover and Weiss ([YW04]); implementing it may be worthwhile, since it enhances flexibility for further processing and decreases loss of information between loosely coupled components.

Probably, not all of these potential tasks will be completed; their importance greatly depends on the demands of further processing, which are still emerging.

7.2 Further Processing

Currently, two scenarios of further processing are actively pursued which will put the structure identification framework presented in this thesis to use:

- First, the results of this thesis will serve as the basis of a transformation framework for producing properly structured, formatted and phrased reports that conform to the formal and informal requirements of the respective domain. Once the underlying structure has been identified in a report dictation, transformation rules can be applied that ensure these requirements are met. Ultimately, the goal is to enhance speech recognition systems such that they automatically perform many tasks which are routinely carried out by professional transcriptionists today.
- Second, the output of structure identification will serve to improve the error rate of ASR by allowing for segment-specific language models. The section and subsection types identified in this thesis are typically characterized by the use of different vocabulary; choosing a specific language model may therefore result in significant performance gains. One possible architecture might be to perform a second pass of speech recognition after the different segments of a report have been identified; however, architectures involving online adaptation of the language model may also be feasible.

Other applications may arise as the framework matures and its potential is fully exploited. Finally, VieCRF, the easily reusable CRF implementation that lies at the heart of the structure identification framework will certainly be applied to numerous other challenging machine learning tasks.

Chapter 8

Summary

> *“People take the longest possible paths, digress to numerous dead ends, and make all kinds of mistakes. Then historians come along and write summaries of this messy, nonlinear process and make it appear like a simple, straight line.”*
>
> *– Dean Kamen*

A framework has been established in this thesis which allows for identification of structure in report dictations. Unformatted raw output of Automatic Speech Recognition (ASR) serves as the input to this mechanism. This ensures loose coupling and, consequently, equal applicability to any concrete ASR implementation.

The framework can identify the boundaries of sentences, paragraphs, enumerations, subsections, sections and various other structural elements occurring in a dictation, even if no explicit clues are dictated. Furthermore, meaningful types are automatically assigned to subsections and sections. These types provide valuable information for various tasks; they may be used to automatically assign headings, if none were dictated, for instance.

Reports from the medical domain have been used as a showcase and for evaluation of the framework; however, the framework can easily be applied to other domains just as well. A mechanism has been presented that exploits the potential of parallel corpora for semi-automatic annotation of data. Using formatted reports that were manually edited by transcriptionists, and the corresponding raw output of speech recognition, reference annotation can be generated that is suitable for learning how to identify structure in the latter representation.

Conditional Random Fields (CRFs), a recently introduced probabilistic framework for labeling sequences and more general structures, lie at the heart of the structure identification mechanism. Through the course of this thesis, VieCRF, an efficient and scalable CRF software package has been developed. VieCRF is publicly available and supports numerous algorithms for both inference and parameter estimation.

Multiple experiments have been performed using VieCRF. For this purpose, CRF models were trained using parts of the aforementioned, semi-automatically annotated corpus. The other, unseen parts of the corpus were then labeled automatically. Various performance metrics were computed from the labeled data in a three-fold cross-validation setting. These results have been presented and discussed thoroughly. They confirm the practicability of the approach pursued in this thesis and give rise to a number of interesting questions which may be clarified in follow-up studies.

Appendix A

Acronyms

> *“I’m perplexed when people adopt the modish abbreviation Ms., which doesn’t abbreviate anything except common sense.”*
>
> *– Dick Cavett*

API	Application Programming Interface
ASR	Automatic Speech Recognition
ASTM	American Society for Testing and Materials
BFGS	Broyden-Fletcher-Goldfarb-Shanno
BOW	Bag-of-Words
CLT	Central Limit Theorem
CRF	Conditional Random Field
DCRF	Dynamic Conditional Random Field
FCRF	Factorial Conditional Random Field
GRMM	Graphical Models In Mallet
HMM	Hidden Markov Model
IID	independent and identically distributed
ILP	Integer Linear Programming
IPP	Integrated Performance Primitives

LBFGS	Limited Memory BFGS
LOC	Lines Of Code
MAP	Maximum a Posteriori
MLE	Maximum Likelihood Estimation
NLP	Natural Language Processing
OLBFGS	Online LBFGS
POD	Plain Old Documentation - Perl's documentation format
POS	Part-of-Speech
RHG	Regular Hedge Grammar
SVM	Support Vector Machine
TRP	Tree-based Reparameterization
UMLS	Unified Medical Language System
VieCRF	Vienna Conditional Random Field Toolkit
WER	Word Error Rate

Appendix B

Bibliography

> *“ If we steal thoughts from the moderns, it will be cried down as plagiarism; if from the ancients, it will be cried up as erudition. ”*
>
> *– Charles Caleb Colton*

[ADW94] C. Apté, F. Damerau, and S. M. Weiss. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, 12(3):233–251, 1994.

[Bra00] Thorsten Brants. TnT: a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing*, pages 224–231, 2000.

[Cho00] Freddy Choi. Advances in domain independent linear text segmentation. In *Proceedings of the first conference on North American chapter of the Association for Computation Linguistics*, pages 26–33, 2000.

[CS96] W. W. Cohen and Y. Singer. Context-sensitive learning methods for text categorization. In Hans-Peter Frei, Donna Harman, Peter Schäuble, and Ross Wilkinson, editors, *Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval*, pages 307–315, Zürich, CH, 1996. ACM Press, New York, US.

[GS86] E. R. Gabrieli and David J. Speth. Automated analysis of the discharge summary. *Journal of Clinical Computing*, 15:1–28, 1986.

[HANS90] P. J. Hayes, P. M. Andersen, I. B. Nirenburg, and L. M. Schmandt. TCS: A shell for content-based text categorization. In *Proceedings of the Sixth IEEE CAIA*, pages 320–326, 1990.

[Hea97] Marti A. Hearst. Texttiling: Segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, 23(1):36–47, 1997.

[HJK+06] Martin Huber, Jeremy Jancsary, Alexandra Klein, Johannes Matiasek, and Harald Trost. Mismatch interpretation by semantics-driven alignment. In *Proceedings of KONVENS '06*, 2006.

[Int02] ASTM International. ASTM E2184-02: Standard specification for healthcare document formats, 2002.

[JGJS99] Michael I. Jordan, Zoubin Ghahramani, Tommi Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

[Joa98] T. Joachims. Text categorization with support vector machines: learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 137–142, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.

[KFL01] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, February 2001.

[Kud05] Taku Kudo. CRF++: Yet another CRF toolkit. `http://crfpp.sourceforge.net/`, 2005.

[LALS07] S. Lamprier, T. Amghar, B. Levrat, and F. Saubion. Seggen: A genetic algorithm for linear text segmentation. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[LHM93] D. A. B. Lindberg, B. L. Humphreys, and A. T. McCray. The Unified Medical Language System. *Methods of Information in Medicine*, 32:281–291, 1993.

[LMP01] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, 2001.

[LSLT05] Man Lan, Sam-Yuan Sung, Hwee-Boon Low, and Chew-Lim Tan. A comparative study on term weighting schemes for text categorization. In *Proceedings of International Joint Conference on Neural Networks*, 2005.

[Mat03] Evgeny Matsuov. Statistical methods for text segmentation and topic detection. Master's thesis, Rheinisch-Westfälische Technische Hochschule Aachen, 2003.

[MB94] G. William Moore and Jules J. Berman. Automatic SNOMED coding. In *Proceedings of the 18th Annual Symposium on Computer Applications in Medical Care, JAMIA Symposium Supplement*, pages 225–229, 1994.

[MCP05] Ryan McDonald, Koby Crammer, and Fernando Pereira. Flexible text segmentation with structured multilabel classification. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 987–994, 2005.

[MS06] Kevin Murphy and Mark Schmidt. Conditional Random Field (CRF) toolbox for Matlab. `http://www.cs.ubc.ca/~murphyk/Software/CRF/crf.html`, 2006.

[Mur00] M. Murata. Hedge automata: a formal model for XML schemata. Web page, 2000. `http://www.xml.gr.jp/relax/hedge_nice.html`.

[MvG+98] P. Mulbregt, I. van, L. Gillick, S. Lowe, and J. Yamron. Text segmentation and topic tracking on broadcast news via a hidden markov model approach. In *In Proceedings of the ICSLP'98*, volume 6, pages 2519–2522, 1998.

[Noc80] Jorge Nocedal. Updating Quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35:773–782, 1980.

[PH02] Lev Pevzner and Marti Hearst. A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, 28(1), March 2002.

[Phi90] Lawrence Philips. Hanging on the metaphone. *Computer Language*, 7(12), 1990.

[PM04] Fuchun Peng and Andrew McCallum. Accurate information extraction from research papers using Conditional Random Fields. In *Proceedings of Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, 2004.

[PNN05] Xuan-Hieu Phan, Le-Minh Nguyen, and Cam-Tu Nguyen. FlexCRFs: Flexible Conditional Random Field toolkit. `http://flexcrfs.sourceforge.net`, 2005.

[Pro07] ALSO Project. Automatic learning and specific adaptation. http://www.coast.at/, 2007.

[Rab89] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286, February 1989.

[RY05] Dan Roth and Wen-tau Yih. Integer Linear Programming inference for Conditional Random Fields. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 736–743. ACM Press, 2005.

[Sar04] Sunita Sarawagi. CRF package: A Java implementation of Conditional Random Fields for sequential labeling. http://crf.sourceforge.net, 2004.

[SC04] Sunita Sarawagi and William Cohen. Semi-Markov Conditional Random Fields for information extraction. In *Proceedings of ICML 2004*, 2004.

[Seb99] Fabrizio Sebastiani. A tutorial on automated text categorisation. In *Proceedings of THAI-99, European Symposium on Telematics, Hypermedia and Artificial Intelligence*, 1999.

[SGHM07] Scott Sanner, Thore Graepel, Ralf Herbrich, and Tom Minka. Learning CRFs with hierarchical features: An application to go. International Conference on Machine Learning (ICML) workshop, 2007.

[SL94] David E. Stewart and Zbigniew Leyk. Meschach: Matrix computations in C. In *Proceedings of the Centre for Mathematics and its Applications*, volume 32, 1994.

[SM05] Charles Sutton and Andrew McCallum. Composition of Conditional Random Fields for transfer learning. In *Proceedings of Human Language Technologies / Empirical Methods in Natural Language Processing (HLT/EMNLP)*, 2005.

[SM06] Charles Sutton and Andrew McCallum. An introduction to Conditional Random Fields for relational learning. To appear, 2006.

[SM07a] Charles Sutton and Andrew McCallum. Improved dynamic schedules for Belief Propagation. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2007.

[SM07b] Charles A. Sutton and Andrew McCallum. Piecewise pseudolikelihood for efficient CRF training. In *Proceedings of International Conference on Machine Learning (ICML)*, 2007.

[SMR07] Charles Sutton, Andrew McCallum, and Khashayar Rohanimanesh. Dynamic Conditional Random Fields: Factorized probabilistic models for labeling and

segmenting sequence data. *Journal of Machine Learning Research*, 8:693–723, March 2007.

[SRM04] Charles Sutton, Khashayar Rohanimanesh, and Andrew McCallum. Dynamic Conditional Random Fields: Factorized probabilistic models for labeling and segmenting sequence data. In *Proceedings of International Conference on Machine Learning*, 2004.

[Sut06] Charles Sutton. GRMM: A graphical models toolkit. `http://mallet.cs.umass.edu`, 2006.

[SYG07] Nicol N. Schraudolph, Jin Yu, and Simon Günter. A stochastic Quasi-Newton Method for online convex optimization. In *Proceedings of 11th International Conference on Artificial Intelligence and Statistics*, 2007.

[SZ02] J. Semecký and J. Zvárová. On regular analysis of medical reports. In *Proceedings of NLPBA 2002*, pages 13–16, 2002.

[Tay07] Stewart Taylor. *Optimizing Applications for Multi-Core Processors, Using the Intel ® Integrated Performance Primitives*. Intel Press, second edition, 2007.

[VJ97] T. L. Veldhuizen and M. E. Jernigan. Will C++ be faster than Fortran? In *Proceedings of the 1st International Scientific Computing in Object Oriented Parallel Environments (ISCOPE'97)*, 1997.

[Wai02] Martin Wainwright. *Stochastic processes on graphs with cycles: geometric and variational approaches.* PhD thesis, MIT, 2002.

[Wal02] Hanna M. Wallach. Efficient training of Conditional Random Fields. Master's thesis, Division of Informatics, University of Edinburgh, 2002.

[Wal06] Hanna M. Wallach. Topic modeling: Beyond bag-of-words. In *Proceedings of the 23^{rd} International Conference on Machine Learning*, 2006.

[WJW02a] M. Wainwright, T. Jaakkola, and A. Willsky. Tree consistency and bounds on the performance of the max-product algorithm and its generalizations. Technical Report LIDS P-2554, Laboratory for Information and Decision Systems, MIT, July 2002.

[WJW02b] Martin Wainwright, Tommi Jaakkola, and Alan S. Willsky. Tree-based reparameterization framework for analysis of sum-product and related algorithms. *IEEE Transactions on Information Theory*, 49(5), 2002.

[WVA07] Johannes Wagner, Thurid Vogt, and Elisabeth André. A systematic comparison of different HMM designs for emotion recognition from acted and spontaneous

speech. In *Proceedings of Second International Conference on Affective Computing and Intelligent Interaction*, pages 114–125, 2007.

[YFW03] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. *Understanding Belief Propagation and its Generalizations, Exploring Artificial Intelligence in the New Millennium*, chapter 8, pages 236–239. Science & Technology Books, January 2003.

[YP97] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of Fourteenth International Conference on Machine Learning*, pages 412–420, 1997.

[YV04] Ming Ye and Paul Viola. Learning to parse hierarchical lists and outlines using Conditional Random Fields. In *Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition (IWFHR'04)*, pages 154–159. IEEE Computer Society, 2004.

[YW04] Chen Yanover and Yair Weiss. Finding the M most probable configurations using Loopy Belief Propagation. In *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.