VigLink Inc.
Utrecht University
Department of Computer Science

Master of Science Thesis

Identifying Product Entities in text with Conditional Random Fields

by

Dafne van Kuppevelt

Supervisor:
Ad Feelders (UU)
Gabor Melli (VigLink)

Utrecht, 2013

Abstract

With a growing portion of the web dedicated to the discussion, review and retail of consumer products, it is increasingly relevant to develop methods for automated extraction of Product Entities in user generated text. In addition, it is important that the extraction models provide feedback about the quality of their output, in the form of a confidence score associated with each entity.

Conditional Random Fields, which are designed as a discriminative solution for structured output prediction, have shown to be successful for the related problem of Named Entity Recognition. Furthermore, their probabilistic nature provides a natural way to obtain a confidence score.

In this thesis, the optimal application of Conditional Random Fields to the specific problem of identifying Product Entities is investigated. A set of experiments is designed and executed to compare different choices of feature sets. The results prove that Conditional Random Fields perform better than heuristic models for this task.

In addition, several existing methods for confidence scoring are experimentally compared, and an optimized algorithm to calculate the exact confidence estimate (known as the Constrained Forward Backward estimate) is introduced. The experiments show that the more heuristic Gamma Product method has a comparable performance to the Constrained Forward Backward method, and thus provides an alternative confidence estimate to use in practice.

Finally, the F1 score for the Product Entity Recognition is enhanced even further by combining models, using the confidence score for voting.

Contents

Acknowledgements

I would like to take the opportunity to thank the people that supported me during this project.

First of all a big thanks to everyone at VigLink for including me in their team and giving me the chance to work on challenging matter in an inspiring environment. An especially warm thanks to Gabor Melli, who was my day-to-day supervisor at VigLink and who always challenged me to grow in my role and who stimulated me to combine academic research with working for a commercial business. I want to thank Nam Pham, who was a great colleague and who also helped with my research, thanks to him for the figures in chapter 7.

I'd like to thank Ad Feelders for his supervision of my thesis and his reviews of my drafts. Finally I want to thank Marries for supporting me throughout my studies and for his feedback on my thesis.

Chapter 1

Introduction

The research problem in this work can be described as Named Entity Recognition for the specific purpose of Product entity recognition. Named Entity Recognition has been widely studied and it has many applications in various fields. The research question for this thesis originates from the field of Affiliate Marketing.

In this chapter, the background of the research question is described by giving an overview of affiliate marketing and the need for Product Entity recognition. Section 1.2 states the research question and subquestions that we aim to answer in this work. In chapter 2, the problem of Product Entity recognition is explained in more detail. It discusses briefly the CPROD1 contest and the heuristic methods that were designed for this data set. The last section describes how to translate Product Entity Recognition to a sequence classification problem.

The theory of Conditional Random Fields is explained in detail in chapter 3, including an overview of the algorithms for inference and training. Chapter 4 gives a theoretical background of the confidence scoring algorithms. Chapter 5 places this work in context by discussing related research on Named Entity Recognition, Product Entity Recognition, Conditional Random Fields and Confidence scores.

The setup of the experiments for this work are laid out in chapter 6. It discusses the evaluation metrics, the data sets and implementation details of the different experiments. The results of the experiments are given in Chapter 7. Finally, chapter 8 states the conclusion of this thesis.

1.1 Affiliate Marketing and content-driven commerce

Affiliate Marketing is a type of marketing where merchants reward publishers on the World Wide Web for their efforts to generate traffic to the merchants' websites. Publishers can promote products and brands in various ways, for example by placing links, banners, or display ads on their websites. When the promotion of products,

brands or merchants is integrated in the content of a website, we speak about content-driven commerce. Examples of content-driven commerce are lifestyle blogs, editorials with product recommendations, or forums about specific vertical markets [1] [2].

Affiliate marketing is a fast growing way for advertisers and publishers to earn money, and the number of affiliate networks and advertisers offering affiliate programs increases [19]. As a result, there is a growing need to make affiliate marketing more user-friendly and accessible for all types of publishers. VigLink is a company that operates in this space, by offering a tool that automatically monetizes all links on a publishers website, which means that publishers get rewarded for all links on their pages that generate valuable traffic.

In addition, VigLink offers the VigLink Insert product to make affiliation more effective. This tool finds words in existing content that can be hyperlinked to relevant product or merchant pages. By automatically placing these links, the VigLink Insert product can generate significantly more traffic from publishers to merchants [3].

Although there might be many words that could be relevant to link, in this work we restrict our attention to product-related terms. This could be product names or product lines. If these terms are found in the content, they can be linked to a search page or destination page on a merchant's website.

To get an overview of the state-of-the-art techniques for product recognition, VigLink sponsored the CPROD1 contest. This competition was held in the fall of 2012 on the Kaggle Platform*. The results of this contest were presented on the ICDM conference [31]. This work builds on the results of this contest, with a focus on the statistical methods used.

The problem of recognizing product-related terms is very similar to that of Named Entity Recognition (NER). Conditional Random Fields (CRFs) have been successfully applied to the NER problem [28], and thus are a logical choice to apply to this particular problem of product entity recognition. Many of the CPROD1 contestants used CRFs for this task, although not all of them obtained good results.

Another request from the industry for the application of product recognition is to get feedback about the performance of the model in the form of a confidence score for each entity. This information can then be used to tune the recall and precision for a specific application or user.

Although other techniques are possible to solve the problem of product recognition, as can be seen in the results of the CPROD1 contest, this work focuses on Conditional Random Fields. Their probabilistic nature makes them very suitable for confidence estimation and the results can easily be extended to applications in various domains, in contrast to the more heuristic methods specifically designed for a given data set.

*http://www.kaggle.com/c/cprod1

1.2 Research question

Following the requirements from the industry, and building on previous work, we arrive at the following research question:

> *How can we use Conditional Random Fields for the recognition of Product entities in text, together with a reliable and accurate confidence estimate for each found entity?*

The research problem of product recognition was described in the CPROD1 contest [31], and we extend this work by investigating specifically the use of Conditional Random Fields. We are interested in a more in-depth analysis of the statistical method and how we can obtain a statistically sound confidence estimate.

Although CRFs have been applied to NER in several occasions, we want to investigate the optimal use of these models for this specific problem. Therefore we formulate the following subquestion:

> *How can we optimize the use of CRFs for Product recognition? What is an optimal set of features?*

The performance of CRF models is heavily influenced by the choice of feature functions. By specifying features specifically for this purpose we hope to enhance performance.

Since CRFs are meant to be a mathematically grounded alternative for heuristic models, we want to experimentally compare CRFs with heuristic models to get information about their relative performance. We therefore formulate the subquestion:

> *How do the CRF methods compare to heuristic methods?*

We compare some simple heuristic methods for Product Entity Recognition, specifically designed for a given data set, to the optimized CRFs.

The second part of our research question is to obtain a method to evaluate the quality of the results of our models. We want a confidence score associated with each product entity found by the model, for example to make the decision to take further action.

> *How can we obtain confidence scores from our models that give reliable information about the quality of the mentions?*

We aim to use the statistical nature of CRFs to evaluate the quality of the proposed mentions and decide whether to show them to the user.

Chapter 2

Text segmentation for products

2.1 Problem description

A large portion of the web is dedicated to the discussion, review and retail of consumer products. It is therefore useful to see *consumer products* as important entities to be able to identify and recognize in text.

Segmentation is the problem of computationally dividing a piece of text into meaningful segments. *Named Entity Recognition* (NER) is the task of segmentation where we want to find segments that refer to entities in the real world. For Consumer Product recognition, the task is to identify segments that refer to a real-world consumer product. We call these *product mentions*. Examples of product mentions are `iPhone 4s` or `Omnia M Windows Phone 7 smartphone`. Our definition of product mentions include both product lines (such as `iPhone`, which refers to a collection of products of the same line) and specific products (such as `iPhone 4s`). Note that different phrases could refer to the same entity, for example:

- I got an **iphone 4s** for my birthday!
- The **Apple iPhone 4s** has some new features...
- I have a **4s**, but I'd like to get a different phone...

In each of these pieces of text, the emphasized fragment refers to a product entity. The underlying product entity is the same for all three phrases, even though the exact mention might differ in length and spelling. In the last text fragment, the mention itself might be ambiguous, but the context helps us understand which entity is meant. The problem now is to identify phrases that are product mentions.

2.2 The CPROD1 contest

The CPROD1 contest was designed to test the state-of-the art methods for recognition and disambiguation of product mentions. The first task of recognition is, as described above, to identify phrases that refer to a product in a piece of text. The second task of disambiguation is to find all products in an existing catalog that the mention refers to.

For this purpose a collection of text fragments, collected from web pages, was manually annotated with product mentions and items from a catalog with full product descriptions that was provided. This catalog contained product descriptions from the Consumer Electronics and Automotive domains. Additionally, two dictionaries, one with brand names and one with common English words were released.

Although recognition and disambiguation can be seen as two separate tasks, these two tasks could be combined, for example by extracting possible mention phrases from the catalog. For this work we only consider the first task of identifying mentions. It is important and useful to recognize new or unknown products, which makes recognition without some sort of list of existing entities an interesting task. In this chapter we will first consider some heuristic methods, that might also use knowledge about known product entities.

2.3 Heuristic methods

Heuristic methods make use of domain knowledge for a specific task, rather than using a statistical model for predictions. Because of the fairly specific nature of the CPROD1 task, many of the contestants proposed heuristic methods. In this section we will discuss a number of these methods, to illustrate how the problem of product recognition can be solved in a different way than by using a statistical model. It can also be useful to incorporate some heuristics into the feature engineering for the statistical models.

Note that these methods are designed specifically for the CPROD1 task. The rules and heuristics are based on the annotations in the training data as observed by the contestants and myself.

2.3.1 Dictionaries

Dictionaries can serve as a very direct source of domain knowledge for heuristic methods and feature engineering. For the CPROD1 contest, two dictionaries, one with brand names and one with common English words were released. No dictionary of product names was released: if such a dictionary were available we could match exactly those phrases that are in the dictionary. Applying the dictionaries that are available, it can be helpful to filter out mentions that, for example, consist of only common English words (using the dictionary) or that contain words from a list of

blacklisted patterns. This filter is also useful for mentions that are obtained through other, statistical or heuristic, methods.

It is of course also possible to use external dictionaries or create a dictionary heuristically. In the CPROD1 contest, one of the contestants created a heuristic dictionary of product names by extracting bi- and trigrams from the product catalog that followed the words "for" or "for a" in more than some minimal number of entries. This suggest that these bi- and trigrams are popular products for which many accessories can be found in the catalog. Of course this methods can come up with false positives (think of examples like "...for men and women") and will only find products that have accessories associated with them.

2.3.2 Rules and patterns

Studying the product annotations in the CPROD1 data, several patterns come to the eye. Here are some patterns that can be found in the annotated data set. Note that these observations are generalizations and do not apply to all mentions. We need to combine several rules to generate mention candidates and filter out unlikely candidates.

1. The first word of the mention is a brand name.
2. The mention does not contain more than one brand name.
3. Mentions are not very short, at least about 4 characters long.
4. A bigram mention consists of a product line or brand name and a model number. The product line or brand name is not a common English word, and the model number is a numeric (or mostly numeric) term.
5. The mention contains a word that contains at least one numerical character.
6. The mention follows a pronoun (*my*, *your*, etc.), article (*the, a, an*) or quantifier (*one, third*, etc), and does not follow *by*.
7. Unigram mentions do not contain common abbreviations (such as *kg*, *min*, etc)

The patterns in items 2, 3, 6 and 7 can be easily used to filter mentions that are identified by some other method. The others can be used to find candidate mentions, that can possibly be validated by looking them up in the product catalog. The following methods are examples of how to use some of the above rules to generate mentions for the solution:

a If a word consists of both letters and numbers, it is possibly a model-word. Look it up in the catalog and try to expand it while retaining catalog entries that contain the expanded phrase.

b If a word is not a common English word and appears not too frequent in the data set, it is possibly a product word. Look it up in the catalog and try to expand.

c If a token is a possible model word (consisting of letters and numbers) but is not in the catalog, and the word before it is a known brand name, this is probably an unknown model of a known brand. Add it to the solution.

For our experiments, we compare CRFs with a rule based model that uses method a, together with a combination of the filters discussed in the next section.

2.3.3 Filters

Using some of the above rules, we define a set of filters to use in our experiments. These heuristic filters help us to improve the results of a model with low precision. We define the following filters:

1. *MentionLengthFilter* filters out mentions longer or shorter than given thresholds or that contain a token that is longer than a given threshold.
2. *SpecialCharacterFilter* filters mentions that contain a special character (e.g. `,!&^`).
3. *TokenFilter* filters mentions that only consist of common words, or contain words from a blacklist or start with a non-alphanumeric character or consist of only non-alphanumeric characters.
4. *PrecedingWordsFilter* filters mentions that are preceded by a word that is unlikely to precede a mention.
5. *FirstTokenFilter* filters mentions that start with a numeric token or with a non-alphanumeric character.
6. *BrandnameFilter* filters out mentions that contain more than one brand, or that are one token long and consist of a brand.

Note that for some of these filters we need the dictionaries of brand names and common words. We bundle these filters together into a mild and a strict filter. These consist of the following filters:

- **Mild:** *PreceedingWordsFilter* with a small blacklist of preceding word combinations, *TokenFilter* with small blacklist, *SpecialCharacterFilter*, *MentionLengthFilter* with minimum mention length 3 and maximum token length 20.
- **Strict:** *PrecedingWordsFilter* using a whitelist of allowed preceding words, *FirstTokenFilter*, *MentionLengthFilter* with minimum mention length 4 and maximum token length 20, *BrandnameFilter*, *BlacklistFilter*, *TokenFilter* with small blacklist, *SpecialCharacterFilter*.

2.3.4 Baseline: dictionary of training data

Another form of heuristic model makes use of the training data, but does no kind of intelligent learning at all. This method is to create a dictionary of product terms from all annotated mentions in the training data. The test data is matched against this list to find the longest matching mentions. This can be done for example by using the Aho-Corasick algorithm [4]. This method was used as a simple baseline for the CPROD1 contest.

2.4 IOB tagging

To make use of statistical models, we first need to describe the problem of product entity recognition in statistical terms. The first step is to separate the text items into tokens. The text is separated by whitespace, so each word is its own token. Additionally, other symbols such as punctuation marks, brackets and quotes are being treated as separate tokens.

This gives us for each text item a *sequence* of tokens. We now translate the problem of product recognition into a *sequence labeling task*. This means that we define the problem as finding a model that predicts an output label for each token in the sequence. Note that we can not use a binary label ("Product" or "Not Product") because product mentions can include multiple tokens and we want to be able to distinguish mentions that directly follow each other.

We therefore define three possible output labels:

1. **B**: token is the beginning token of a product mention.
2. **I**: token is part of, but not the first token of a product mention
3. **O**: token is not part of any product mention

This transformation method is called IOB-tagging and is often used for Named Entity Recognition [40]. If there are multiple types of entities (for example, Person and Place) each type get its own set of B and I labels. Thus if there are p types of entities, the size of the set of labels is $2p + 1$.

Figure 2.1 shows an example of a sentence that is tokenized and labeled with IOB-tags.

Now we have defined IOB tagging, we can give a more formal problem definition: Given tokenized text item $\mathbf{x} = x_1, \ldots, x_T$, where each x_t is a specific token, predict an output sequence $\mathbf{y} = y_1, \ldots, y_T$ where each y_t is an IOB label for the corresponding token x_t. This is a sequence classification task, where we want to predict a class label (from a finite set of labels) for each element in a sequence. In the next chapter we will see that CRFs complete this task by using a graphical model in which the sequence tokens are represented as observable variables and the output labels as latent variables.

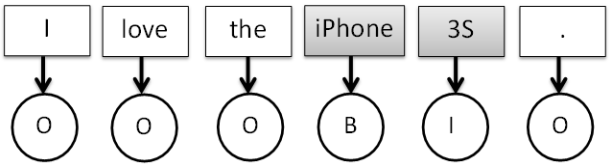


Figure 2.1: This is an example of a text sample with a product mention that has been tokenized and labeled according to the IOB tagging. The shaded tokens belong to the mention.

Chapter 3

Conditional Random Fields

Conditional Random Fields were first introduced by Lafferty and McCallum in [23]. They were introduced specifically for sequence tagging, as an improvement on Maximum Entropy Markov Models (MEMMs). MEMMs were introduced by McCallum, Freitag and Pereira in [27] as a non-generative alternative for Hidden Markov Models (HMMs).

In this chapter we will explain Conditional Random Fields, starting by reviewing Hidden Markov Models. The explanation in this chapter closely follows that of the tutorial about CRFs by Sutton and McCallum [47].

When we speak in this chapter about *observations*, we mean the observable elements of a sequence: in the context of NER these are words. The *hidden* elements in the sequence are the labels for each word, which is what we want to predict.

3.1 Hidden Markov Models

Hidden Markov Models are directed graphical models for sequence tagging, that can be factorized according to the Markov property: each observation random variable only depends on the previous observation and the current label. More formally, if we have a sequence of observations $\mathbf{x} = x_1, \ldots, x_T$ in the input space $\mathcal{X}$ and a sequence of tags $\mathbf{y} = y_1, \ldots, y_T$ in the outcome space $\mathcal{Y}$, a Hidden Markov model consists of a joint distribution over $\mathcal{X}^T \times \mathcal{Y}^T$ that factorizes as follows:

$$P(\mathbf{x}, \mathbf{y}) = \prod_{t=1}^{T} P(x_t|y_t)P(y_t|y_{t-1}) \tag{3.1}$$

A specific HMM is thus defined by only it's transition probabilities $P(y_t|y_{t-1})$ and state probabilities $P(x_t|y_t)$.

In figure 3.1 we see a representation of a HMM as graphical model. In this representation, we define a directed graph $G = (V, E)$ where each node represents a

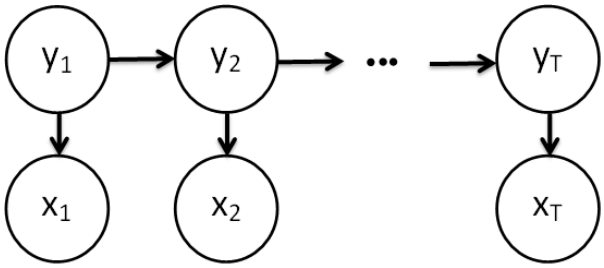


Figure 3.1: Graphical representation of a Hidden Markov Model.

variable. In the case of HMMs this means we have T nodes for the input variables $x_1, \ldots, x_T$ and another T nodes for the outputs $y_1, \ldots, y_T$.

Note that the HMM is a generative model: the assumption is that the observations are *generated* by the corresponding label. The observations are only dependent on the label variables, thus the assumption in a HMM is that all observations are independent given the labels. This is a very strong assumption that can be relaxed by adding some graph structure among the observation variables.

With the introduction of Conditional Random Fields we don't assume any distribution of the observation variables, which gives the freedom to use complex interdependent features.

3.2 Linear-Chain Conditional Random Fields

Linear-chain conditional random fields have a similar structure as HMMs, but allow more freedom in the structure of the observation variables, because they model only the *conditional* distribution. They allow any type of feature function in the factorization, instead of only the transition and state probabilities. Moreover, each label can depend on more than just the corresponding observation variable. We can introduce additional features that are derived from the corresponding observation, but also features that capture information about other observations in the sequence. We therefore assume now that each y_t corresponds to a observation *vector* $\mathbf{x}_t$.

A Linear-Chain Conditional Random Field is defined by a set of K feature functions $f_k(y, y', \mathbf{x})$ and corresponding weights $\theta = \{\theta_k\} \in \mathbb{R}^K$. The conditional distribution should takes the following log linear form:

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \exp\left\{ \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t) \right\} \tag{3.2}$$

where $Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{t=1}^{T} \exp\left\{ \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t) \right\}$ is a normalization constant.

We make the assumption that we have the same feature functions and corresponding weights for each time step t. Is is also possible to define different factors

for each time step, similar to a non-homogeneous HMM, but for this work these types of models are not relevant.

3.2.1 Generative-discriminative pair

Note that the relation between HMMs and CRFs is similar to that between Naive Bayes and Logistic Regression for classification. Given a vector of observations $\mathbf{x} = x_1, \dots x_N$ and one output label y, the Naive Bayes classifier models the joint distribution $P(\mathbf{x}, y)$ assuming that all observation variables are independent:

$$P(\mathbf{x}, y) = P(y) \prod_{i=1}^{N} P(x_i|y) \tag{3.3}$$

Logistic Regression however only models the conditional distribution $P(y|\mathbf{x})$ according to a log-linear function:

$$P(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left\{\theta_y + \sum_{i=1}^{N} \theta_{y,i} x_i\right\} \tag{3.4}$$

where $\frac{1}{Z(\mathbf{x})} = \sum_y \exp\{\theta_y + \sum_{i=1}^{N} \theta_{y,i} x_i\}$ is a normalization factor.

Logistic Regression and Naive Bayes are known to be a generative-discriminative pair [21] which means the models belong to the same parametric family of models but are fit to optimize either the conditional (for discriminative) or joint (for generative) distribution. Every Naive Bayes classifier can be converted to a Logistic Regression classifier with the same decision boundary, and vice versa. An important difference is that Logistic Regression makes no assumption about dependencies among the observation variables. In Logistic Regression we can even add derived input variables such as polynomials of the original observations. On Naive Bayes this could have a disastrous effect because it would lead to a violation of the independence assumption. For CRFs we have a similar advantage.

To visualize the relation between CRFs and HMMs we use the following alternative notation for CRFs to define the distribution as a product of factors:

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \psi_t(y_t, y_{t-1}, \mathbf{x}_t) \tag{3.5}$$

where

$$\psi_t(y_t, y_{t-1}, \mathbf{x}_t) = \exp\left\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\right\} \tag{3.6}$$

Written in this form, we see that (linear-chain) CRFs are an example of *factor graphs*. A factor graph is an undirected graphical model such that the distribution can be factorized as a product of local factors. An example of the factor graph for a simple linear-chain CRF can be seen in figure 3.2. As a comparison, the directed graph for

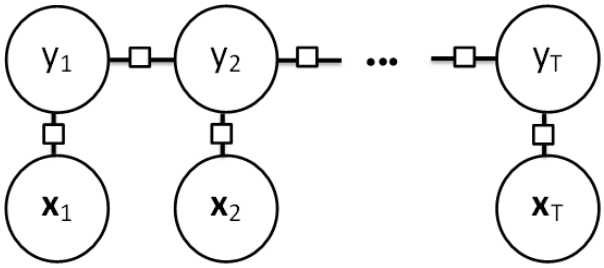
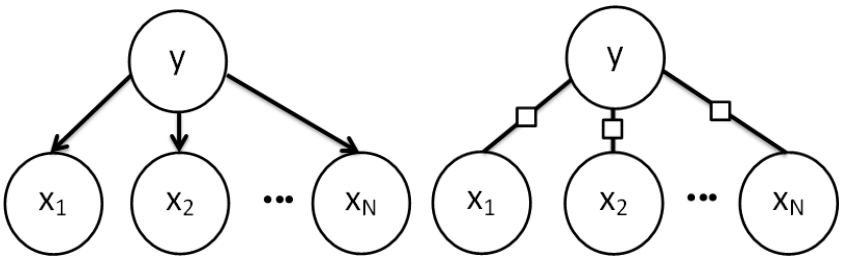


Figure 3.2: Factor graph for a linear-chain Conditional Random Field, with only transition feature functions and node-observation functions. The whole observation vector $\mathbf{x}_t$ is represented as one node.



(a) Naive Bayes (b) Logistic Regression

Figure 3.3: Directed graph for Naive Bayes and factor graph for Logistic Regression.

Naive Bayes (similar to HMM) and the factor graph for Logistic regression (similar to CRF) are shown in figure 3.3.

Every Hidden Markov Model can be written as a CRF. For this purpose, we rewrite equation 3.1 in the following manner:

$$P(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \prod_{t=1}^{T} \exp \left\{ \sum_{i,j \in \mathcal{Y}} \theta_{ij} \mathbf{1}_{\{y_t = i\}} \mathbf{1}_{\{y_{t-1} = j\}} + \sum_{i \in \mathcal{Y}} \sum_{o \in \mathcal{X}} \mu_{io} \mathbf{1}_{\{x_t = o\}} \mathbf{1}_{\{y_t = i\}} \right\} \tag{3.7}$$

where we define the weights as follows:

$$\begin{aligned} \theta_{ij} &= \log P(y' = i | y = j) \\ \mu_{io} &= \log P(x = o | y = i) \\ Z &= 1 \end{aligned}$$

This gives us a feature function $f_{ij}(y, y', x) = \mathbf{1}_{\{y=i\}} \mathbf{1}_{\{y'=j\}}$ for each possible transition and one feature function $f_{io}(y, y', x) = \mathbf{1}_{\{y=i\}} \mathbf{1}_{\{x=o\}}$ for each possible state-observation pair. If we index these $K = |\mathcal{Y}| \times |\mathcal{Y}| + |\mathcal{X}| \times |\mathcal{Y}|$ feature functions as $f_k(y, y', x)$ and the corresponding weights θ_{ij} and μ_{io} as θ_k, we get the following

form for the joint distribution:

$$P(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \prod_{t=1}^{T} \exp \left\{ \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, x_t) \right\} \tag{3.8}$$

Next we can calculate the conditional probability as follows:

$$P(\mathbf{y}|\mathbf{x}) = \frac{P(\mathbf{x}, \mathbf{y})}{\sum_{\mathbf{y}'} P(\mathbf{y}', \mathbf{x})} \tag{3.9}$$

This gives us exactly a definition of a conditional random field with the features f_k.

In conclusion, we can see that HMMs and CRFs form a generative-discriminative pair in the same manner as Logistic Regression and Naive Bayes. In other words, CRFs can be thought of as the structured output extension of Logistic Regression and thus profits from the same advantages as the discriminative Logistic Regression model [33].

3.2.2 Feature functions

In general for CRFs, we are not constrained to the simple feature functions that we have in HMMs. One simple addition is to add the surrounding observations to the observation vector. We define $\mathbf{x}_t = (x_{t-1}, x_t, x_{t+1})$. We retain the transition feature functions $f_{ij}(y, y', \mathbf{x}) = \mathbf{1}_{\{y=i\}} \mathbf{1}_{\{y'=j\}}$ but we add the feature functions:

$$\begin{aligned} f_{io}(y_t, y_{t-1}, \mathbf{x}_t) &= \mathbf{1}_{\{y_t=i\}} \mathbf{1}_{\{x_t=o\}} \\ f'_{io}(y_t, y_{t-1}, \mathbf{x}_t) &= \mathbf{1}_{\{y_t=i\}} \mathbf{1}_{\{x_{t-1}=o\}} \\ f''_{io}(y_t, y_{t-1}, \mathbf{x}_t) &= \mathbf{1}_{\{y_t=i\}} \mathbf{1}_{\{x_{t+1}=o\}} \end{aligned}$$

So for each state/observation pair we add a feature for the previous observation and for the next observation. This gives no violation of independence rules, because the CRF model makes no assumptions about dependencies among the observations.

We can also make transformations on the input observations and use these for feature functions. Suppose we have a set of M transformation functions $\{q_m(\mathbf{x}_t)\}_{m \in \{1,\ldots,M\}}$ that map our observation vector $\mathbf{x}_t$ to a feature vector. These features can be real valued or ordinal, thus if the original values are categorical without any order, they need to be binarized. For the NER applications, we usually have discrete observation functions with a limited number of values, like for example "word x_t starts with a capitalized letter" or "the number of numerical symbols that word x_t contains". We can then define for each transformation function $q_m(\mathbf{x}_t)$ and each label value $i \in \mathcal{Y}$ the *node-observation function*:

$$f_{im}(y_t, y_{t-1}, \mathbf{x}_t) = \mathbf{1}_{\{y_t=i\}} q_m(\mathbf{x}_t) \tag{3.10}$$

This gives us in the model a weight for each feature - label value combination. It is also possible to let transition feature functions depend on the (transformed) observations. We define the *edge-observation functions* as follows for each observation

function $q_m(\mathbf{x_t})$ and label pair $i, j \in \mathcal{Y}$:

$$f_{ijm}(y_t, y_{t-1}, \mathbf{x}_t) = \mathbf{1}_{\{y_t=i\}}\mathbf{1}_{\{y_{t-1}=j\}}q_m(\mathbf{x}_t) \tag{3.11}$$

Note that the node-observation and the edge-observation functions together can result in a very large number of features. It can be useful to apply some feature-reduction techniques to avoid overfitting and memory issues.

3.3 Inference

The task of retrieving conditional probabilities from an existing graphical model is called *inference*. In general graphical models there is no polynomial algorithm known to solve this problem, but for linear-chain CRFs, as for HMMS, we can use dynamic programming to obtain an exact solution efficiently. In sequence label tagging we are first of all interested in predicting the most probable labels, that is the label sequence that maximizes the conditional probability. This can be done by the dynamic programming Viterbi algorithm. If we also want to know the corresponding conditional probabilities, as we will need for the confidence scores, we can calculate these with the similar Forward-Backward algorithm.

3.3.1 Viterbi

Given a linear-chain CRF model with conditional distribution $P(\mathbf{y}|\mathbf{x})$ and an observation sequence $\mathbf{x}$, the sequence labeling task is to find:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}) \tag{3.12}$$

If we substitute equation 3.5 in the above optimization objective, we get:

$$\begin{aligned} \mathbf{y}^* &= \arg\max_{\mathbf{y}} \tfrac{1}{Z(\mathbf{x})} \textstyle\prod_{t=1}^{T} \psi_t(y_t, y_{t-1}, \mathbf{x}_t) \\ &= \arg\max_{\mathbf{y}} \textstyle\prod_{t=1}^{T} \psi_t(y_t, y_{t-1}, \mathbf{x}_t) \end{aligned} \tag{3.13}$$

To compute this, we cache the intermediate scores for the maximizing label sequences so far. For each label value j, we define this cached variable for each position $t \in \{1, \ldots, T\}$ as follows:

$$\delta_t(j) := \max_{\mathbf{y}_{1,\ldots,t-1}} \psi_t(j, y_{t-1}, \mathbf{x}_t) \prod_{t'=1}^{t-1} \psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}_{t'}) \tag{3.14}$$

For each time step t we can calculate this variable for each label value i recursively:

$$\delta_t(j) = \max_{i \in \mathcal{Y}} \psi_t(j, i, \mathbf{x}_t)\delta_{t-1}(i) \tag{3.15}$$

This can be proven with induction [7]. To get the complete optimal sequence we need to compute the variables for all time steps and trace back the optimal sequence by computing each optimal label starting at the end of the sequence:

$$y_T^* = \arg\max_{j \in \mathcal{Y}} \delta_T(j) \tag{3.16}$$

and then recursively:

$$y_t^* = \arg\max_{i \in \mathcal{Y}} \psi_t(y_{t+1}^*, i, \mathbf{x}_{t+1}) \delta_t(i) \tag{3.17}$$

This algorithm gives us the optimal output sequence $\mathbf{y}^*$, but not the corresponding optimized conditional probability. To calculate that, we need the Forward Backward algorithm, which will be explained in the next section.

3.3.2 Forward-Backward algorithm

Given a label sequence $\mathbf{y}$ (for example the optimal sequence $\mathbf{y}^*$ obtained from the Vitberbi algorithm) and an observation sequence $\mathbf{x}$, the conditional distribution of $\mathbf{y}$ given $\mathbf{x}$ according to a CRF model is given by:

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \exp\left\{ \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t) \right\} \tag{3.18}$$

The product of factors is easy to compute if we have the feature functions and weights. The challenge lies in computing the normalization factor $Z(\mathbf{x})$. The forward-backward algorithm can calculate this factor, as well as the state probabilities $P(y_t|\mathbf{x}_t)$ and transition probabilities $P(y_{t-1}, y_t|\mathbf{x}_t)$.

Similar to the cached variables in the Viterbi algorithm, we define the *forward variables* and the *backward variables*. The forward variables cache the sum of all paths up to a certain node from left to right. For time step t and output value j we define:

$$\alpha_t(j) := \sum_{\mathbf{y}_{1,\ldots,t-1}} \psi_t(j, y_{t-1}, \mathbf{x_t}) \prod_{t'=1}^{t-1} \psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}_{t'}) \tag{3.19}$$

Similar, the backward variable caches the sum of all paths *starting* from this time step and label value. For each time step t and output value j the backward variable is:

$$\beta_t(i) := \sum_{\mathbf{y}_{t+1,\ldots,T}} \psi_{t+1}(y_{t+1}, i, \mathbf{x}_{t+1}) \prod_{t'=t+2}^{T} \psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}_{t'}) \tag{3.20}$$

Both forward and backward variables can be computed recursively, similar to the Viterbi algorithm. The difference is that for these variables, we take the summation instead of the maximum value in each computation step.

The recursion formula for the forward variable is:

$$\alpha_t(j) = \sum_{i \in \mathcal{Y}} \psi_t(j, i, \mathbf{x}_t)\alpha_{t-1}(i) \tag{3.21}$$

For the backward variable:

$$\beta_t(i) = \sum_{j \in \mathcal{Y}} \psi_{t+1}(j, i, \mathbf{x}_{t+1})\beta_{t+1}(j) \tag{3.22}$$

This can also be proven with induction.

Note that if we want to compute $Z(\mathbf{x})$ we need the sum over all paths, which we can do both by computing the forward variable for the far right node or the backward variable for the starting node. Thus we have:

$$Z(\mathbf{x}) = \sum_{y_T} \alpha_T(y_T) = \beta_0(y_0) \tag{3.23}$$

where we assume we have a starting state y_0 with one possible value.

The forward and backward variables can also be used to compute the conditional state probabilities $P(y_t|\mathbf{x})$. To calculate this, we write out the probability in factors as follows:

$$\begin{aligned}
P(y_t|\mathbf{x}) &= \sum_{\mathbf{y}_{1,\ldots,t-1}, \mathbf{y}_{t+1,\ldots,T}} P(\mathbf{y}|\mathbf{x}) \\
&= \sum_{\mathbf{y}_{1,\ldots,t-1}, \mathbf{y}_{t+1,\ldots,T}} \frac{1}{Z(\mathbf{x})} \prod_{t'=1}^{T} \psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}_{t'}) \\
&= \frac{1}{Z(\mathbf{x})} \sum_{\mathbf{y}_{1,\ldots,t-1}} \sum_{\mathbf{y}_{t+1,\ldots,T}} \psi_t(y_t, y_{t-1}, \mathbf{x}_t) \left(\prod_{t'=1}^{t-1} \psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}_{t'}) \right) \left(\prod_{t'=t+1}^{T} \psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}_{t'}) \right) \\
&= \frac{1}{Z(\mathbf{x})} \left(\sum_{\mathbf{y}_{1,\ldots,t-1}} \psi_t(y_t, y_{t-1} \prod_{t'=1}^{t-1} \psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}_{t'}) \right) \left(\sum_{\mathbf{y}_{t+1,\ldots,T}} \prod_{t'=t+1}^{T} \psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}_{t'}) \right)
\end{aligned} \tag{3.24}$$

Substituting the definitions for the forward and backward variables in here, we get:

$$P(y_t|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \alpha_t(y_t)\beta_t(y_t) \tag{3.25}$$

We can calculate the conditional probabilities of the state transitions in a similar manner:

$$\begin{aligned}
P(y_{t-1}, y_t|\mathbf{x}) =& \frac{1}{Z(\mathbf{x})} \psi_t(y_t, y_{t-1}, \mathbf{x}_t) \\
& \left(\sum_{\mathbf{y}_{1,\ldots,t-2}} \prod_{t'=1}^{t-1} \psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}_{t'}) \right) \quad \left(\sum_{\mathbf{y}_{t+1,\ldots,T}} \prod_{t'=t+1}^{T} \psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}_{t'}) \right)
\end{aligned} \tag{3.26}$$

So we get:

$$P(y_{t-1}, y_t|\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\alpha_{t-1}(y_{t-1})\psi_t(y_t, y_{t-1}, \mathbf{x}_t)\beta_t(y_t) \tag{3.27}$$

If we cache the forward and backward variables for all states, we can compute all state and transition probabilities easily. This can be useful for example for computing confidence scores, which we will explore in the next chapter. Computing the transition probabilities is also necessary for parameter learning, as explained in the next section.

Each forward, backward or Viterbi computation step runs in order $O(|\mathcal{Y}|^2)$ time: quadratic in the number of states. Note that for simple IOB tagging, these are only 9 possible transitions. We can reduce the running time further by excluding impossible transitions, in case of IOB tagging we can exclude $O \to I$.

Given an input sequence of length T, the total running time of the forward-backward algorithm is order $O(T|\mathcal{Y}|^2)$. Another time issue might be computing the factors $\psi_t(y_t, y_{t-1}, \mathbf{x}_t)$. These are computed by taking a dot product between the feature function values and the feature weights. This computation can be sped up by storing the feature values in sparse feature vectors, since often many of these features are zero. This is also more memory-efficient.

3.4 Parameter estimation

The problem of estimating the parameters for the CRF model is briefly described here, since the models in our experiments are trained using standard techniques without customized alterations.

The task of parameter estimation is, given a CRF model with feature functions f_k, to find a set of weights θ_k that maximizes the likelihood of a training data set $\mathcal{D} = \{(\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\}_{m\in\{1,\ldots,M\}}$. In this training set, each training instance $(\mathbf{x}^{(m)}, \mathbf{y}^{(m)})$ consists of an input and output sequence, where the length T does not need to be constant.

We define the loglikelihood of a weight set according to this training data as follows:

$$L(\theta) = \sum_{m=1}^{M} \log P(\mathbf{y}^{(m)}|\mathbf{x}^{(m)}; \theta) \tag{3.28}$$

We want to find the set of parameter values θ that maximizes this loglikelihood function. To avoid overfitting, we add an extra regularization term to the objective function. For L2 regularization, this term is $-\sum_{k=1}^{K} \frac{\theta_k^2}{2\sigma^2}$ where σ^2 is the regularization parameter. Another choice is L1 regularization, where the penalty term is $-\alpha \sum_{k=1}^{K} |\theta_k|$. In general, L1 regularization favors sparser weight vectors, thus it can be used as a form of feature selection. However, numerical optimization is somewhat more challenging because the optimization function is not differentiable in zero, as a result of the absolute value.

Suppose we use L2 regularization, then our optimization function for a given linear-chain CRF becomes:

$$L(\theta) = \sum_{m=1}^{M}\sum_{t=1}^{T}\sum_{k=1}^{K}\theta_k f_k(y_t^{(m)}, y_{t-1}^{(m)}, \mathbf{x}_t^{(m)}) - \sum_{m=1}^{M}\log Z(\mathbf{x}^{(m)}) - \sum_{k=1}^{K}\frac{\theta_k^2}{2\sigma^2} \quad (3.29)$$

If we take the partial derivative with respect to θ_k we get:

$$\frac{\partial L(\theta)}{\partial \theta_k} = \sum_{m=1}^{M}\sum_{t=1}^{T} f_k(y_t^{(m)}, y_{t-1}^{(m)}, \mathbf{x}_t^{(m)}) - \sum_{m=1}^{M}\sum_{t=1}^{T}\sum_{y',y} f_k(y', y, \mathbf{x}_t^{(m)})P(y', y|\mathbf{x}_t^{(m)}) - \frac{\theta_k}{\sigma^2} \quad (3.30)$$

Unfortunately, there is no closed form solution to this optimization problem. However, there are several out-of-the-box algorithms that approximate the solution numerically, such as steepest ascent and Newtons method. For most of these methods, it is necessary to compute $L(\theta)$ and $\frac{\delta L(\theta)}{\delta \theta_k}$ for all k. If we look at equations 3.29 and 3.30 we see that we need the Forward-Backward algorithm to compute $Z(\mathbf{x}^{(m)})$ and $P(y', y|\mathbf{x}_t^{(m)})$ for each training instance $(\mathbf{x}^{(m)}, \mathbf{y}^{(m)})$. Thus, in each iteration in the numerical optimization we need to execute the Forward-Backward algorithm once for each training instance. Remember that the running time of the Forward-Backward algorithm is $O(T|\mathcal{Y}|^2)$, so with I iterations the total running time will be $O(T|\mathcal{Y}|^2 MI)$. The number of iterations I depends on the numerical optimization algorithm and the dataset.

3.5 Example

To illustrate how CRFs and the inference algorithms work, we consider the example from figure 2.1. Suppose we have three simple binary feature functions: **FUC** denotes whether the first character is upper case, **NUM** denotes whether the word contains numerical characters, and **DICT** denotes whether the word is found in a dictionary of common English words. We also add the features of the previous and succeeding token in the sequence.

In example 1 the featurization of the example phrase is shown. In the notation we previously used in this chapter, the observation vector $\mathbf{x}_t = (x_{t-1}, x_t, x_{t+1})$ consists of a token window of width 3, and we have 9 transformation functions $q_m(\mathbf{x}_t)$ that transform $\mathbf{x}_t$ into a binary feature vector.

Suppose we only make use of edge-observation functions. To compute the factors in the graph, we need weight vectors of length 9 for each possible label transition. An example of the weight vector $\theta_{O\to O}$ is shown in example 2. To calculate the log factor $\log \psi_t(O, O, \mathbf{x}_t)$ we have to take the dot product of this weight vector with the feature vector of token x_t. For example for the second token *love* the log factor becomes: $(0,0,1,1,0,1,0,0,1)\cdot(1,-1,3,2,0,4,1,0,1) = 10$.

In figure 3.4 we see the lattice corresponding to this example. Each transition weight corresponds to the log factor $\log \psi_t(y_t, y_{t-1}, \mathbf{x}_t)$. The weights we calculated

for the $O \rightarrow O$ transitions are shown in the lattice and all other weights can be calculated similarly. The most probable outcome $\mathbf{y}^*$ can be determined by finding the heaviest path in the lattice using the Viterbi algorithm. To calculate the probability of the path we need to find the normalization constant, using the forward backward algorithm. This finds the sum of the exponential weights of all paths.

Suppose now that the most probable path that our model finds is **OOOBIO**. This results in the segment *iPhone 3S*, as we would expect. In the next chapter we will explore methods to find a sound estimate of the probability of such a segment, where the segment is represented in the lattice by a few succeeding states.

	I	love	the	iPhone	3S	.
FUC	1	0	0	0	0	0
NUM	0	0	0	0	1	0
DICT	1	1	1	0	0	0
FUC-1	0	1	0	0	0	0
NUM-1	0	0	0	0	0	1
DICT-1	0	1	1	1	0	0
FUC+1	0	0	0	0	0	0
NUM+1	0	0	0	1	0	0
DICT+1	1	1	0	0	0	0

Example 1: Featurization of the example phrase.

	$O \rightarrow O$
FUC	1
NUM	-1
DICT	3
FUC-1	2
NUM-1	0
DICT-1	4
FUC+1	1
NUM+1	0
DICT+1	1

Example 2: Example of transition weights for one of the possible transitions.

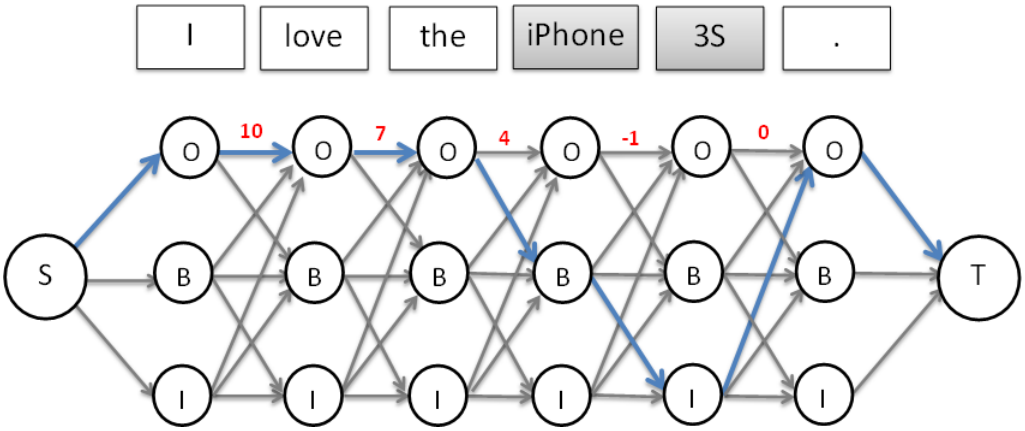


Figure 3.4: Example lattice with the factor weights for the $O \rightarrow O$ transitions.

Chapter 4

Confidence scores

There are several reasons for the desire of a confidence score associated with each segment. If we view the segmentation as a retrieval task, we need a relevance score to rank the found segments. If the relevance score is accurate, we can improve the accuracy of the set of found segments by only considering the top-n ranked segments. If we have a confidence score for segments, we can use this as a relevance score. In the case of consumer product recognition for example, we can show only product mentions above a certain confidence threshold, to improve the accuracy perceived by the user. We can even use this to improve the F1-score, since CRFs are not designed to optimize the F1-score of segments.

Another reason to obtain a confidence score is to get an indicator of the quality of the found mentions. For this purpose, the confidence score is in fact an estimation of the probability that the segment is correct, such that on average, for example, segments with score of 0.8 would be correct eighty percent of the time. These confidence scores can be used to evaluate the model and possibly to provide feedback to further improve the model.

Combining these two requirements, we want a confidence score associated with each segment that is fine-grained and ranges from zero to one and is positively correlated with the segments' probability of being correct.

Many predictive models naturally contain a confidence scoring function. Decision trees use the distribution of training records in the leaf node; Support vector machines can make use of the margin in the prediction - though it cannot be treated as the probability of correctness. Linear CRFs, as we will see in more detail, also have a natural way to define confidence.

We discuss three different methods of estimating the confidence score of a segment. The first method calculates the exact probability of the output labels of the segment according to the model. We also introduce a fast method to compute this value. The other two methods don't have such a proper probabilistic meaning, but

are somewhat simpler to compute and might therefore still be useful in practice.

4.1 Conditional Forward Backward algorithm

Suppose we have a segment consisting of the output tags $S = s_1, \ldots, s_m$ that spans the tokens k to $k+m$, so $y_k = s_1, \ldots, y_{k+m} = s_m$. This segment could be part of the optimal output sequence $\mathbf{y}*$ according to our model (in which case the segment is returned by our model) but in theory the confidence score can be calculated for any segment. We can define the *conditional forward-backward* (CFB) confidence measure as the probability of exactly this span given the input data:

$$c_{cfb}(y_k, \ldots, y_{k+m}) = P(y_k = s_1, \ldots, y_{k+m} = s_m | \mathbf{x}) \tag{4.1}$$

For IOB-tagging we might say that only the probability of the span is not precise enough, because we also want to know how confident we are that the span is not in fact longer. We can therefore include the token following the span, which we require not to be the continue-tag I:

$$c_{cfb}(y_k, \ldots, y_{k+m}) = P(y_k = s_1, \ldots, y_{k+m} = s_m, y_{k+m+1} \neq I | \mathbf{x}) \tag{4.2}$$

In case of NER with different types of entities, we should constrain y_{k+m+1} to be unequal to the inside tag corresponding with the entity type of this segment.

Given a segment $S = s_1, \ldots, s_m$ starting at token k, let $C_S = \langle y_k = s_1, \ldots, y_{k+m} = s_m, y_{k+m+1} \neq I \rangle$ denote the constraints imposed by this segment, and $Y_{C_S} = \{y_1, \ldots y_T | C_S\}$ denote the set of output sequences that satisfy these constraints. The probability of this sequence is given by:

$$P(C_S|\mathbf{x}) = \sum_{\mathbf{y}} P(\mathbf{y}, C_S|\mathbf{x}) = \sum_{y \in Y_{C_S}} P(\mathbf{y}|\mathbf{x}) \tag{4.3}$$

Now we plug in the CRF-factorization from equation 3.5:

$$\sum_{y \in Y_{C_S}} P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \sum_{y \in Y_{C_S}} \prod_{t=1}^{T} \psi_t(y_t, y_{t-1}, \mathbf{x}_t) \tag{4.4}$$

The sum of factors $Z_{C_S}(\mathbf{x}) = \sum_{y \in Y_{C_S}} \prod_{t=1}^{T} \psi_t(y_t, y_{t-1}, \mathbf{x}_t)$ can be computed by performing the forward backward algorithm on the constrained lattice: that is, only states and transitions that satisfy the constraints C_S are allowed. This is called the Constrained Forward Backward (CFB) Algorithm and was introduced by Culotta and McCallum [16]. The confidence is than computed by normalizing the constrained sum by the unconstrained normalization factor:

$$c_{cfb}(y_k, \ldots, y_{k+m}) = \frac{Z_{C_S}(\mathbf{x})}{Z(\mathbf{x})} \tag{4.5}$$

Note that for each found segment, the set of constrains is different, thus the CFB algorithm needs to be performed for each segment, in addition to the run for the unconstrained lattice. Remember that the running time of the forward-backward algorithm is $O(T|\mathcal{Y}|^2)$, so if we have n segments in one sequence the running time becomes $O((n+1)T|\mathcal{Y}|^2)$.

We introduce an optimized variation on the CFB algorithm, that reuses information from the unconstrained lattice. Remember that our constraints always take the form $C_S = \langle y_k = s_1, \ldots, y_{k+m} = s_m, y_{k+m+1} \neq i \rangle$ for a certain continue-label I. We can thus factorize the constrained sum as follows:

$$\begin{aligned} Z_{C_S}(\mathbf{x}) = & \left(\psi_t(y_t, y_{t-1}, \mathbf{x}_t) \sum_{y_1,\ldots,y_{k-1}} \prod_{t=1}^{k-1} \psi_t(y_t, y_{t-1}, \mathbf{x}_t) \right) \times \\ & \left(\sum_{y_{k+m+2},\ldots,y_T} \prod_{t=k+m+2}^{T} \psi_t(y_t, y_{t-1}, \mathbf{x}_t) \right) \times \\ & \prod_{t=k+1}^{k+m} \psi_t(y_t, y_{t-1}, \mathbf{x}_t) \times \\ & \sum_{y_{k+m+1} \neq I} \psi_t(y_{k+m+1}, y_{k+m}, \mathbf{x}_{k+m+1}) \end{aligned}$$

We can recognize the forward and backward variables in this equation. In fact, if we write the last factor somewhat differently we get:

$$\begin{aligned} Z_{C_S}(\mathbf{x}) = & \alpha_k(s_1) \cdot \\ & \prod_{t=k+1}^{k+m} \psi_t(y_t, y_{t-1}, \mathbf{x}_t) \cdot \\ & [\beta_{k+m}(s_m) - \beta_{k+m+1}(I)\psi_t(I, s_m, \mathbf{x}_{k+m+1})] \end{aligned}$$

The first factor is the forward variable for the first token of our span, which can be seen as the total weight left from this output state in the lattice. The second factor is the product of all feature values of the succeeding tokens in the span, which can be seen as the weight inside the segment. The last factor is just the backward variable for the last token in the segment, so all weight in the lattice to the right of the segment, but we subtract the weight resulting from the 'illegal' transition to the output label I.

Note that we store the forward and backward variables for all states in the forward backward algorithm, so these can be retrieved in $O(1)$ time. To make this optimimzed algorithm work, we also need to store the feature factors $\psi_t(y_t, y_{t-1}, \mathbf{x}_t)$. The size of this table will be $|\mathcal{Y}|^2 T$, which might be large if the number of output labels is big. In many cases however it will be more efficient than recomputing the feature factors each time.

In conclusion, the runtime of the Optimized Constrained Forward-Backward (OCFB) only depends on the length of the segment, if we assume $Z(\mathbf{X})$ is known.

4.2 Gamma product

For the Gamma product method, we make a very rigid assumption: that the probability of an output tag inside a segment given the observations is independent of the previous output tag. This assumption is not very likely to be true: we designed our feature functions in such a way that the observations influence the transition probabilities. However, by making this assumption for the segment, we simply ignore the transition weights inside the segment and get the following comprehensive formula:

$$c_{gprod}(y_k, \ldots, y_{k+m}) = \prod_{t=k}^{k+m} P(y_t|\mathbf{x}) \tag{4.6}$$

We define the *gamma* probabilities to be these individual marginal probabilities: $\gamma_t(y_t) = P(y_t|\mathbf{x})$. The gamma product confidence score consists thus, as its name suggest, of the product of the individual gamma probabilities of the tags in the segment.

The gamma probability can be easily computed from the forward and backward variables and total weight from the forward-backward algorithm:

$$\gamma_t(y_t) = \frac{1}{Z(\mathbf{x})}\alpha_t(y_t)\beta_t(y_t) \tag{4.7}$$

We only need to do one run of the forward-backward algorithm for each sequence, in which we could store all gamma probabilities, or we could retrieve them from the stored forward and backward variables.

4.3 Gamma average

The Gamma average confidence score consists of the average value of the gamma probabilities of the tags in the segment. In this method we also disregard the transition weights inside the segment (which makes the computation easier). In mathematical notation:

$$c_{avg}(y_k, \ldots, y_{k+m}) = \frac{1}{m}\sum_{t=k}^{k+m} \gamma_t(y_t) \tag{4.8}$$

The average of the gamma probabilities does not have a very proper probabilistic meaning, but it is based on the intuition that the probabilities of all labels in the segment should contribute to the confidence score, but longer segments should not be in disadvantage. Note that in the gamma product method this could be the case, because longer segments need a bigger product and thus the total confidence will go down, even though the individual marginal probabilities might be high.

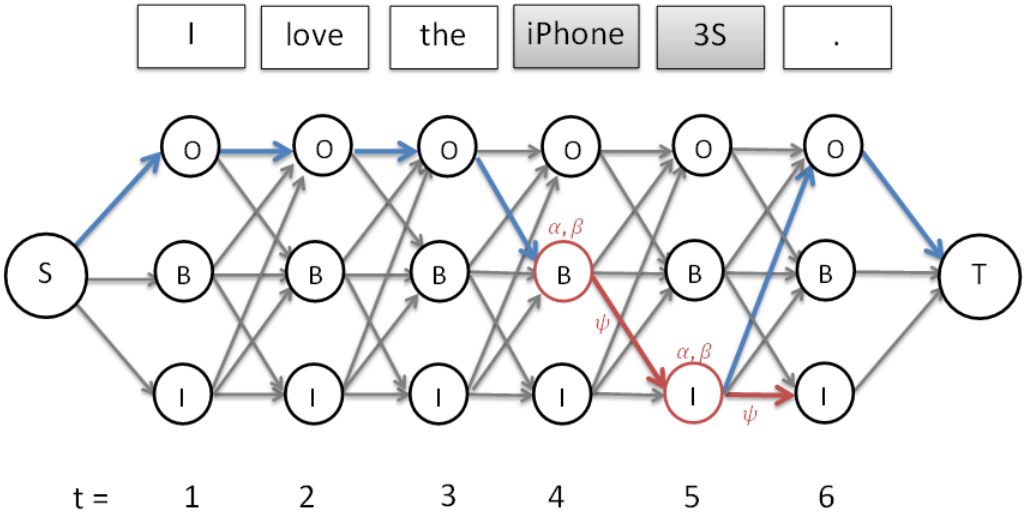


Figure 4.1: Example lattice with corresponding time steps.

The Gamma average score is just as easy to compute as the gamma product score, and requires one run of the forward-backward algorithm for each sequence with at least one segment in it.

4.4 Example

Let's take a look at our previous example in figure 4.1 to see how the confidence scores would be computed for our segment. Suppose we have the segment *iPhone 3S* for which we want to compute the confidence score. If we have performed the forward backward algorithm, we have a α and β value for each node in the lattice. Suppose we also stored the feature factors $\psi_t(y_t, y_{t-1}, \mathbf{x})$ for each edge in the lattice. For the optimized CFB method, we need the weight of the one transition within the segment, namely $\psi_5(B, I, \mathbf{x}_5)$ and the weight of the illegal transition $\psi_6(I, I, \mathbf{x}_6)$ to substract from the backward variable of the last node in the segment. We also need the forward variable $\alpha_4(B)$ and the backward variable $\beta_5(I)$ of the last and first nodes in our segment. All nodes and edges we need information of are color marked in figure 4.1. The complete calculation for the optimized CFB confidence score will be:

$$c_{cfb}(y_4 = B, y_5 = I) = \frac{1}{Z(\mathbf{x})}\alpha_4(B) \cdot \psi_5(B, I, \mathbf{x}_5) \cdot (\beta_5(I) - \psi_6(I, I, \mathbf{x}_6)) \quad (4.9)$$

Where we assume we have computed $Z(\mathbf{x})$ in the forward backward algorithm. For the Gamma Average and Gamma Product we don't need the feature factors $\psi_t(y_t, y_{t-1}, \mathbf{x})$ but we need the forward and backward values for the lattice nodes in our segments. Then we can compute $\gamma_4(B) = \frac{1}{Z(\mathbf{x})}\alpha_4(B) \cdot \beta_4(B)$ and $\gamma_5(I) =$

$\frac{1}{Z(\mathbf{x})}\alpha_5(I) \cdot \beta_5(I)$ for the output tags corresponding to the words *"iPhone"* and *"3S"* and take the product or average value respectively to compute the confidence score.

Chapter 5

Related work

This chapter discusses the most important research that has been done on the topics related to this work.

The first section gives an overview of the research area of Named Entity Recognition. Section 5.2 zooms in on Product entity recognition, for which the CPROD1 competition has resulted in the most research on this topic. Section 5.3 discusses research on CRFs and their most important applications. The final section describes earlier work on confidence scores for segmentation and other related tasks.

5.1 Named Entity Recognition

Since the early nineties, NER has been a widely studied problem. A good survey of all research on NER up to 2006 is given by Nadeau and Sekine [34]. Originally the focus has been on Person, Location and Organization entity types in journalistic articles [18]. Over the years NER has been applied to a wide variety of problems, varying in the origin of the written or spoken text and the types of entities. As several research papers suggest, NER models are hard to generalize and thus models for different applications might differ substantially [39].

The original problem of identifying persons, locations and organization has been studied with several shared tasks and benchmark data sets in different languages, starting with MUC-6 [18] and MUC-7 [10] (both in English), followed by the widely used data sets from the CONLL 2002 [49] (in Dutch and Spanish) and CONLL 2003 [50] (in English and German) shared tasks. Although these tasks have focused on these standard entity types in journalistic text, other text sources and entity types have been researched as well. Examples of text sources on which NER has been applied are email messages [32], Wikipedia pages [54], tweets [41], and a set of diverse text types [26].

Some researchers have expanded the definition of a Named Entity to include other entities than Persons, Organizations and Locations, or focused on a specific type of entity (such as in this work). Examples are NER for web browsing [58] with entity types that can be defined by users, and NER for job title entities [12].

Although early methods for NER often consisted of heuristics and hand-crafted rules, several supervised and unsupervised learning methods have become successful in finding previously unknown entities. Supervised learning methods that have been applied to the NER problems are HMMs [6], Maximum entropy models [9] and MEMMs [27] which are a combination of HMMs and Maximum Entropy models. CRFs were introduced as an improvement on MEMMs and were first applied to the NER problem in [28]. Support Vector Machines (SVMs) for structured variables have also been applied to NER [53] and seem to have a comparable performance to CRFs [22].

The set of features for supervised NER models differs greatly among existing systems. The features chosen can influence the performance of the model heavily [22]. The list of features often include word level features, list look-up features (such as in a dictionary), and document and corpus features (such as word frequencies) [34]. Cohen and Sarawagi [12] explore intelligent techniques to use domain dictionaries in a supervised recognition model. They use Semi-Markov Models to overcome the problem that matching single tokens against the dictionary is not very useful for multi-token entity names.

There have also been unsupervised models designed for NER, that do not require user-labeled training data but instead use information about the structure of language and co-occurrence of words to generate possible entity names. An example of an unsupervised system is KnowItAll [17], that only needs a small set of rules as input and then uses several methods to expand and prune the set of entity candidates.

5.2 Product Entity Recognition

The CPROD1 competition [31] was designed specifically for the purpose of identifying product entities. Prior to this competition, limited research has been done on NER for product entities, although in some cases Product was regarded as one of several categories for NER.

Bick [5] applied a rulebased NER-system to danish text with product/title and brand as possible entities. These entities are mostly misclassified as being a Person or other entity type.

Sekine and Nobata [45] define a list of 200 entity categories, including Product. Their tagger for Japenese however does not find any correct Product entities.

Ritter et al. [41] train a tagger for NER on tweets, with Product as one of 10 entity types that is commonly found in tweets. They separate the tasks of recognition

and type classification, which makes their approach very different from systems that specialize on finding product entities, such as for the CPROD1 competition.

Zhao et al. [57] have developed an annotated corpus in Chinese for Product recognition. They use heuristic methods to generate a set of candidate entities, using a dictionary of brand names. Then they use a Hierarchical Hidden Markov Model to classify the candidate entities.

5.2.1 CPROD1

The purpose of the CPROD1 contest was to identify and disambiguate product mentions in text extracted from web pages. Most contestants separated this task in first identifying the product phrases (the recognition phase) and then linking the found phrases to the correct products in the catalog (the disambiguation phase). The recognition methods used by the contestants were both heuristic models, as described in chapter 2, and CRFs (no other statistical methods were used). The top five contestants submitted a paper about their solutions, and their methods are briefly discussed here.

The winners of the contest were Wu et al. [55], who achieved an F1-score of 42.18% for recognition and an F1-score of 22.04% for the combined task of recognition and linking. Their system is a hybrid model of a rulebased recognizer, the baseline look-up model and two different CRFs that use the same set of regex features as explained in chapter 6, dictionary lookup features and Collins' pattern features [13].

The second place contestant Topchylo [52] used a purely heuristic model, which extracted possible entity names from the catalog that was provided. Romaszko [42], who came in third, used 5 different CRFs in addition to the baseline look-up. For the CRFs he used regex and dictionary features as described in chapter 6, and varied the window size of surrounding token features and maximum value of each feature to create different models. The resulting mentions were filtered using heuristic rules that made use of the product catalog.

The fourth place contestant Gödény used a completely rulebased system that depends on chunking the text to generate candidate mentions. Finally, Toh et al. [51] who were fifth on the leaderboard, used just one CRF for recognition. The features they use for this CRF are quite different from the features used in this work and by the other contestants. In addition to simple unigram and bigram token features, they used complex regex features to capture model name patterns and a feature that looks up the mention in an external dictionary of products.

Tabel 5.1 shows the F1 scores for recognition for the CPROD1 contestants. These numbers have not been published before, because the F1 calculation for the contest incorporated the score for linking to the catalog. The final ranking was also based on the score for the combined task, which explains why the recognition scores are not ranked in decreasing order. We got access to the original solution files to calculate

Place	Author	F1
1	Wu et al.	42.18%
2	Topchylo	31.08%
3	Romaszko	41.50%
4	Gödény	37.21%
5	Toh et al.	26.06%

Table 5.1: Recognition F1 scores for CPROD1 contestants

the recognition scores, for comparison with our CRF models that only solve the recognition problem.

5.3 Conditional Random Fields

CRFs are graphical models for structured predictions, and they can be applied to a wide variety of problems other than Named Entity Recognition. Structured prediction is the problem of any problem where the input or output of the prediction model consist of structured information, e.g. a sequence instead of a single label.

NER is an example of sequence classification, where the input of the model is a sequence and the output of the model is a sequence of corresponding class labels. CRFs have been applied to many other sequence classification problems. In language processing, two other sequence classification problems that are studied and solved by CRFs are Shallow Parsing and Part Of Speech (POS) tagging. Shallow parsing is the prediction of simple, non-hierarchical grammatical elements in text. Sha and Pereira [46] applied CRFs to shallow parsing for finding Noun Phrases (NP chunking), using IOB tagging. Their experiments showed that CRFs outperform other sequence tagging methods. In POS tagging each token should be labeled with a syntactical label. Lafferty et al. experimented on POS tagging when they first introduced CRFs [23].

Other applications of linear-chain CRFs for language processing include Chinese word segmentation [36], table extraction [38], information extraction from research papers [37], identifying sources of opinion [11], word alignment for machine translation [8] and many others. In bio-informatics, linear-chain CRFs have been used for gene prediction [15], identifying non-coding regions in RNA [43] and protein fold recognition [24].

General (non-linear) CRFs have also been applied both to language processing and other problems. Sutton et al. [48] introduce Dynamic Conditional Random Fields for segmenting sequences, which allow multiple output labels per input variable, for example both a POS-tag and an IOB-tag for NP chunking. Gunawardana et al. use CRFs for Automatic Speech Recognition (ASR) where they extend the use of CRFs such that it is possible to have hidden observations. Another field where CRFs are

widely applied is Computer Vision [35], in which case the graphical structure of the model is often more complex.

5.4 Confidence scores

The topic of confidence estimation has been explored in several settings that apply trained sequence labelers.

Culotta and McCallum [16] wrote a seminal paper to find a sound, probabilistic estimate for the confidence of multi-word mentions produced by trained Conditional Random Fields. The authors compare the Constrained Forward Backward and the Gamma Product methods as described in chapter 4, and additionally they test a method consisting of a Maximum Entropy classifier on top of the CRF results. For evaluation of the methods they compute Pearson's R and Average precision, both of which measure the performance of the confidence score as a relevance score. To our knowledge, this is the only work that describes different confidence measures for CRFs for multi-token segments. Mejer and Crammer [30] introduce and compare different methods for probabilistic confidence estimation on top of possibly non-probabilistic chunking algorithms. These methods could also be used on top of a CRF, but wouldn't make any use of the probabilistic information contained in the lattice of a CRF. Corbett and Copestake [14] use the Conditional Forward Backward confidence estimate to calculate confidence scores in MEMMS for Chemical entity candidates, to tune the recall and precision. Their work could be extended to CRFs.

Another domain that regularly applies sequence taggers is Automatic Speech Recognition (ASR). The survey by Jiang [20], for example, gives an overview of all Confidence Estimation methods used to estimate the reliability of words in ASR. Although there is no mention of CRFs, they do explain how the posterior probability of a word can be used as a confidence score. Yu et al. [56] explore the scenario where the trained system does include an ordinal score with each prediction and the requirement is to calibrate the confidence score as a post-processing step. Their approach is of particular interest for situations where the predictive model needs to be treated as a black-box (possibly based on something other than CRFs). Seigel and Woodland [44] use CRFs for confidence estimation on top of features from an existing ASR system. Their work differs from ours in that there is only one confidence score per word, which is the probability that that word is correct.

Chapter 6

Experiments

The experiments for this work can be divided into two parts: the first experiments relate to finding the optimal CRF for product entity recognition. The second set of experiments is designed to compare different methods of confidence estimation for segments.

In this chapter, we will first explain the metrics used to evaluate product entity recognition models and the quality of confidence scores. In Section 6.2 the data sets used for the experiments are described. Section 6.3 explains the set up of the product recognition experiments with CRFs, section 6.4 describes the experiments with confidence scores and the final section 6.5 describes how we use confidence scores to further enhance our models through ensembles.

6.1 Evaluation metrics

6.1.1 F1 score

The F1-score is an evaluation metric often used in information retrieval. It denotes the harmonic mean between the precision and recall. The *precision* p is the fraction of retrieved items that is correct and the *recall* r is the fraction of correct items that is retrieved. In our application, the retrieved items are the segments found by the model. We want both p and r to be as high as possible. The general F_β score is defined as a weighted combination of these scores:

$$F_\beta = (1 + \beta^2) \cdot \frac{p \cdot r}{\beta^2 \cdot p + r} \tag{6.1}$$

The higher the value for β, the more recall is favored over precision. We use the F_1 score which weighs out recall and precision equally:

$$F_1 = 2 \cdot \frac{p \cdot r}{p + r} \tag{6.2}$$

We favor models that have a higher F_1 score.

6.1.2 Confidence evaluation metrics

For the evaluation of the confidence measures, we look at two qualities. The first is how well the confidence score performs as an indicator of the precision. The second is how well it works as a discriminator between true and false positives. To measure the first quality, we divide the data into bins according to their confidence scores. We choose a uniform bin width of 0.05 or 0.1, and only take into account bins that have at least 10 data points in them, to get a reliable estimate of the precision in the bin. We will show the expected precision (which is the center of the bin, assuming the confidence scores have approximately a uniform distribution) plotted against the found precision in the bin. In the ideal case these numbers match and we get a straight diagonal line. The closer to this ideal line in the plot, the better the confidence measure is for estimating the expected precision. To quantify this, we compute the Root Mean Squared Error for the bins.

To measure how well the confidence score can be used to distinguish between true and false positives, we rank all the found segments according to their confidence score and look at the Recall/Precision curves. In the ideal case, all the true positives are ranked at the top, which pushes the curve to the (1,1) corner of the plot. To measure this, the area under the Recall/Precision curve, otherwise known as the Average Precision, is calculated. We compute the RMSE for the binning and the Average Precision for all the tested confidence scoring methods. In addition, the metrics are computed for the ideal case (where all true segments get a confidence score of 1 and all false segments zero), the worst case (where the scores are opposite of what they are supposed to be) and the case where all segments get a constant confidence score and the segments are ranked such that true and false segments are evenly distributed.

6.2 Data

For the experiments on product entity recognition, the CPROD1 data set was used. For the confidence experiments we used the CoNLL 2002 shared task data in addition to the CPROD1 data. The reason for this is that this data set is more well-known and to show that the methods are generally applicable to other domains as well.

6.2.1 CPROD1

The CPROD1 data consists of a set of text items with annotations. A text item is a portion of text extracted from a web page, that is stripped from HTML, divided into paragraphs and tokenized and put into JSON-format. Each text item has a unique ID. An example of a text item is shown in Example 3.

```
"TextItem":{"0c1edc5b2ed5abb25e25b966ccdb01d2": ["Here", "'s", "an",
"example", "of", "a", "(", "pretokenized", ")", "text, item", ".",
 "<s>", "<p>", "Check", "out", "the", "new", "iPhone", "4s", "!"]}
```

Example 3: CPROD1 text item

BRANDNAME	6042
ENCOMMONWORD	79765
GRAMMATICALWORD	149
MERCHANT	35
PRODUCTCAT	24
PRODUCTFEAT	8
Total	86023

Table 6.1: Dictionary categories with the number of phrases in that category provided in the CPROD1 data set.

These text items were manually annotated for product mentions, including references to a product catalog that was provided with the data set. Each annotation is represented in a single line that includes the text item id, the start- and end position of the mention and the product ids of the corresponding products in the catalog. For the purpose of this research we don't need the references to the catalog so we translate the annotations into IOB-tags for the text items.

The data set is divided into text items for training, evaluation and development. The development test set was released during the submission period of the CPROD1 contest, but in this work we will only use it for parameter learning in the experiments with ensemble models. Additionally, a dictionary of words of different categories was released. There were two large categories (Brands and English Common Words) and four small categories (Grammatical words, Merchant, Product Category and Product Feature). Table 6.1 shows the number of items for each of these categories in the dictionary. Some of the CRF models only use the larger dictionaries for the features.

6.2.2 CoNLL 2002

The CoNLL 2002 shared task was held as a contest in the Conference on Computational Natural Language Learning (CoNLL) and provided a benchmark data set for Named Entity Recognition in Dutch and Spanish [49]. One year later a similar contest was held, the CoNLL 2003 task [50] with data sets in English and German, but unfortunately not all data for this task is publicly available for free.

The CoNLL 2002 data set consists of sentences from news articles that are manually annotated with four types of entities: Persons (PER), organizations (ORG), locations (LOC), and miscellaneous names (MISC). This annotated data was divided into training, development and test files. Each file consists of lines with a token

and the corresponding IOB-tag. An empty line denotes the sentence boundary. An example of a part of a sentence as it would appear in the file can be seen in example 4.

We train one model for each language on the training files, and test them on the evaluation data files.

Wolff	B-PER
,	O
currently	O
a	O
journalist	O
in	O
Argentina	B-LOC
,	O
played	O
with	O
Del	B-PER
Bosque	I-PER

Example 4: Sentence with IOB tags in the CoNLL2002 data

6.3 CRFs for CPROD1

This section describes the setup of the experiments to compare several CRF models applied on the CPROD1 data set. We describe the CRF implementation used for the experiments and the different sets of features used within the CRFs.

6.3.1 Mallet

Mallet [29] is a java library for machine learning, including tools for sequence tagging using HMMs, MEMMs and CRFs. For our experiments we used the CRF implementation in Mallet. By default, Mallet uses L2 regularization in training the model.

6.3.2 Feature engineering

The selection of interesting features is an essential step in developing a usable recognition model. The number of features and cardinality also heavily influences the performance of the model, both in time and quality. A too large number of features generally increases the risk of overfitting and makes the model slower in both the training and test phase.

The first set of features that we use is a set of simple regexes, some with a boolean (thus binary) result (Is the first character numerical?) and others as a count (Number of periods). All of these regex features represent a simple property of the token. These regex features are shown in table 6.2. Regex features that represent a

count of a specific property, are capped to a maximum value, to limit the cardinality of the feature. We add a regex feature **URL** that determines whether the token is a URL, shown in table 6.3.

Another set of features in some of the models are the Pattern features, which were introduced by Collins [13]. The **PTRN** feature is the token with all uppercase letters replaced by `A`, all lowercase letters replaced by `a`, all numerical characters replaced by `0` and all other characters replaced by `_`. So for example if we have a token `GT-i9100`, it gets the pattern feature `AA_a0000`. This gives an abstract notation of the structure of the token, which seems very promising for product names that often have numerics, capitals and special characters within the token. We also add a compressed pattern feature **CPTRN** that corresponds to the pattern feature with all consecutive repeating pattern characters removed. In the example from above, the token `GT-i9100` gets a compressed pattern feature `A_a0`. This gives even more abstract information about the structure of the feature. The compressed pattern feature has a lower cardinality than the regular pattern feature, so it might capture some more general information.

Finally, we use the dictionaries provided in the CPROD1 dataset to add dictionary features. In some of the models we only used the largest dictionaries, namely Brands and Commonwords. For each of the dictionaries used, a feature is added that expresses how many times the token is found in the dictionary.

Given of the generative nature of CRFs, we can add information about the surrounding tokens in the sequence to the feature vector of a token. We choose to include the complete feature vector of surrounding tokens of different window sizes. For a window of size three, only the features of the token directly before and after the token are included. For a window of size five we include the two tokens before and the two tokens after the current token. A window size of more than 5 seems to be infeasible as the feature space would become too large. We can also choose to include not only the features of the surrounding tokens but also the surrounding tokens themselves as features. Note that this increases the feature space even further because the cardinality of 'token' is very large.

6.4 Confidence scores

For the confidence score experiments, we choose one model from the previous experiments to test the confidence scoring methods on the CPROD1 data. For the CoNLL we train two models, one for Dutch and one for Spanish, using Mallet and featurization as implemented by Mejer and Crammer [30]. The features consist of a set of regex features similar to the ones used for CPROD1, the surrounding tokens and bigrams of the current and surrounding tokens, and prefixes and suffixes of the current token.

These models were used to find segments in the data with corresponding confi-

Name	Description	Type	Regex	Cap
FC-UC	Is the first character uppercase alphabetical?	BOOL	`^[A-Z]`	
FC-LC	Is the first character lowercase alphabetical?	BOOL	`^[a-z]`	
FC-NUM	Is the first character numerical?	BOOL	`^[0-9]`	
CHARCNT	Number of characters	COUNT	`[.]`	13
NUMCNT	Number of numerical characters	COUNT	`[0-9]`	6
UCCNT	Number of uppercase alphabetical characters	COUNT	`[A-Z]`	5
LCMCNT	Number of lowercase alphabetical characters	COUNT	`[a-z]`	11
DSHCNT	Number of dashes	COUNT	`[\-]`	2
SLSHCNT	Number of slashes	COUNT	`[\/]`	2
PERIODCNT	Number of periods	COUNT	`[\.]`	2

Table 6.2: Regex features. The cap is the maximum value allowed for this feature.

URL	Is the token a url?	BOOL	`^www\.|^http|.*\.com$|.*\.org$|.*\.net|.*@`	

Table 6.3: URL feature

dence scores according to the three tested methods: constrained forward backward, gamma product and gamma average.

6.5 Ensemble methods

To improve the F1 score of the models, it can be advantageous to combine several CRF models, trained on the same data but with different sets of features. These models often have their own strengths, thus combined together they can find more correct mentions. A heuristic way of combining these models is by voting. The optimal minimal number of votes can be determined by experiments. A more advanced way to combine the models is to use the most reliable confidence score that each model provides with the mentions that it finds, as described earlier.

If we interpret the confidences as probabilities, we can use the probability framework to estimate a combined confidence score for the output segments of the ensemble method. Let $M_1 \dots M_k$ be the different models, and S the set of all segments that were found by any of the models. For any segment s in S, let $c_j(s) = P(s|M_j)$ be the confidence output by model M_j. i.e. the probability that s is a segment under the assumption that M_j models the true distribution. If the segment is not found by this model, we assume the confidence is zero. Note that this is a simplifying assumption, in reality the probability of an IOB-sequence with this segment as an output can have a probability larger than zero, but this becomes expensive to calculate.

The combined confidence for segment s then is:

$$c_{tot}(s) = P(s) = \sum_{j=1}^{k} P(s|M_j)P(M_j) \tag{6.3}$$

We assume that the priors $P(M_j)$ for all the models sum up to one. We choose these priors to be uniform, but they can be chosen according to some other distribution if that suits the application.

The combined confidence score of the ensemble model can be used to tune the precision and recall and to optimize the F1 score. By estimating the confidence threshold that results in the highest F1 score on the development data set, we can use this threshold to let the ensemble model return only mentions with a confidence score higher than this threshold. This results in the probabilistic equivalent of voting of models.

Chapter 7

Results

In this chapter we lay out the results of the experiments as described in the previous chapter. In addition, we interpret the results to conclude where improvements can be made.

We will begin with the results of the experiments of the different CRFs for product entity recognition. Then we use confidence scoring to further evaluate the ensemble methods of CRFs. For these experiments the CFB method was used. Finally, the different confidence scoring methods are compared.

7.1 F1 scores of models

We tested six CRF models with varying combinations of feature sets. The results of these models are shown in table 7.1, compared against the simple look-up baseline and the best scoring heuristic model. As can be seen from these results, most CRF models score better than the heuristic models.

Note that all models have a much larger precision than recall, which means that quality of the mentions found by the models is relatively good, but the outcome does not by far cover all true mentions in the text. This is a clue that we can get a higher F1 score by combining the results of the models in ensembles.

We test ensemble methods by starting from the two best scoring models, and each time adding the next best scoring method. If the results of an ensemble model contained overlapping segments, the segment with the highest confidence score was chosen.

We also test what happens if we set the threshold for the combined confidence (as explained in the previous chapter) to the optimal value learned from the development test set. If in the development test set the highest F1 was retrieved by including all found mentions, we set the threshold to zero. The results, together with the

	CRF							
Modelname	1	2	3	4	5	6	Heuristic	Baseline
Regex features	Yes	Yes	Yes	Yes	Yes	Yes		
URL features	No	No	No	Yes	No	Yes		
Pattern Features	No	No	No	No	Yes	No		
Surrounding tokens	Yes	No	No	No	No	No		
Dictionary features	All	All	All	Brands, CommonWords	All	Brands, CommonWords		
Token window	3	5	3	3	3	5		
Precision	66.7%	42.0%	56.2%	54.7%	66.7%	50.0%	52.6%	64.5%
Recall	19.0%	22.3%	19.4%	19.4%	18.0%	9.0%	19.0%	9.5%
F1 score	29.5%	29.1%	28.9%	28.7%	28.4%	15.3%	27.9%	16.5%

Table 7.1: Recall, precision and F1 score of different CRF models

minimum confidence threshold learned from the development set, are shown in table 7.2.

We can see that, as we expected, the F1 score increases when we combine models. Even though the quality of the models we add to the ensemble decreases, the output of these models still adds value to the final F1 score. We see that even when we combine all models together, the precision still 'wins' from the recall. Nonetheless, the F1 can be enhanced somewhat for these models by filtering out the lowest confidence mentions.

	no confidence filter				with confidence filter		
Models	prec	rec	F1	min conf	prec	rec	F1
1,2	46.9%	28.9%	35.8%	0	46.9%	28.9%	35.8%
1,2,3	46.0%	29.9%	36.2%	0	46.0%	29.9%	36.2%
1,2,3,4	45.3%	31.8%	37.3%	0	45.3%	31.8%	37.3%
1,2,3,4,5	45.9%	32.2%	37.9%	0.103	49.3%	32.2%	39.0%
1,2,3,4,5,6	44.4%	33.6%	38.3%	0.086	47.6%	33.2%	39.1%

Table 7.2: Recall, precision and F1 score of the ensemble models.

In preparation for the experiments to compare confidence methods, we show the performance of the models used in the confidence experiments in table 7.3. We have one model per data set.

		Segments true	Segments predicted	Precision	Recall	F1
Dutch	PER	703	811	70.3%	81.1%	75.3%
	LOC	479	506	73.9%	78.1%	75.9%
	ORG	686	478	83.3%	58.0%	68.3%
	MISC	748	691	77.7%	71.8%	74.6%
	Overall	2616	2486	75.6%	71.8%	73.7%
Spanish	PER	1222	1077	88.1%	77.7%	82.6%
	LOC	984	1209	64.5%	79.3%	71.1%
	ORG	1687	1565	78.5%	72.9%	75.6%
	MISC	444	364	54.1%	44.4%	48.8%
	Overall	4337	4215	74.9%	72.8%	73.8%
CPROD1	Overall	211	73	56.2%	19.4%	28.9%

Table 7.3: Overall performance of CRF models used for the confidence experiments, in terms of recall, precision and F1 measures.

7.2 Confidence scoring methods

The CFB, Gamma product and Gamma average methods were tested on the CPROD1 and CoNLL2002 data. For the evaluation of the confidence as a measure of precision, the items were binned into intervals of width 0.05 based on their confidence score for the CoNLL2002 data, and bins of width 0.1 for the CPROD1 data (since there were fewer segments found by the model in the latter). The average precision in the bins was plotted against the expected precision (the center of the bin). These graphs, together with the optimal line $x = y$ are plotted in figure 7.1. The Root Mean Squared Error between the expected precision and precision in the bin is shown in table 7.4.

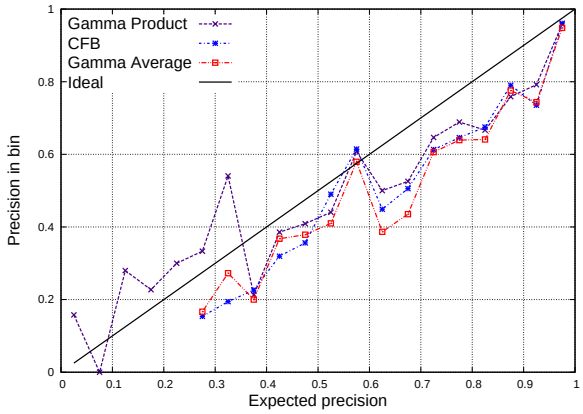
Next, the retrieved mentions were sorted on confidence scores and the recall and precision were calculated for the top-n mentions for each n. The resulting recall-precision graphs are shown in figure 7.2. The closer to the (1,1) corner of the graph, the better the ranking differentiates between relevant and non-relevant items. The area under the graph, which corresponds to the Average Precision, is also shown in table 7.4, together with the ideal and worst case scores.

The results suggest that CFB and Gamma Product confidences perform better than Gamma average both as an indicator of expected precision and as a discriminator between true and false segments. For NER in Dutch, Gamma Product seems to be the winner, and for NER Spanish CFB has a slight win. The difference in performance between the methods is very small however, and thus hardly significant. We conclude from these results that CFB and Gamma Product are equally good methods to estimate confidence, and are preferred over Gamma average. In both datasets the

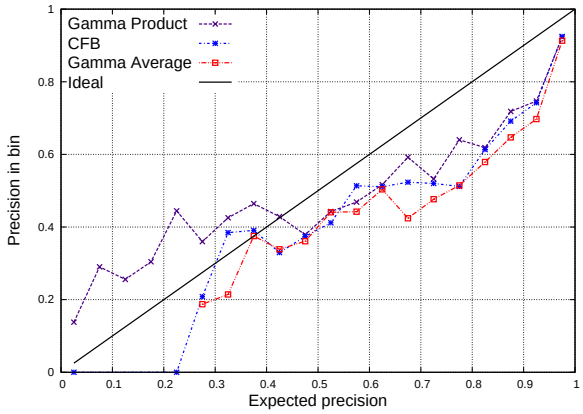
Gamma Product confidences are more spread out, giving more reliable estimates for the lower confidence segments. This could be a reason to choose for gamma product instead of CFB. Another reason to choose for Gamma Product is that the gamma probabilities are slightly easier to compute and also easily accessible in several of the CRF packages.

	NER Dutch		NER Spanish		CPROD1	
	RMSE	Avg Prec	RMSE	Avg Prec	RMSE	Avg Prec
CFB	0.097	0.681	**0.117**	**0.681**	0.177	**0.153**
Gamma Average	0.108	0.680	0.138	0.676	0.247	0.144
Gamma Product	**0.084**	**0.682**	**0.117**	0.676	**0.170**	0.151
Ideal	0	0.718	0	0.728	0	0.194
Constant	-	0.543	-	0.545	-	0.109
Worst case	1	0.391	1	0.390	1	0.071

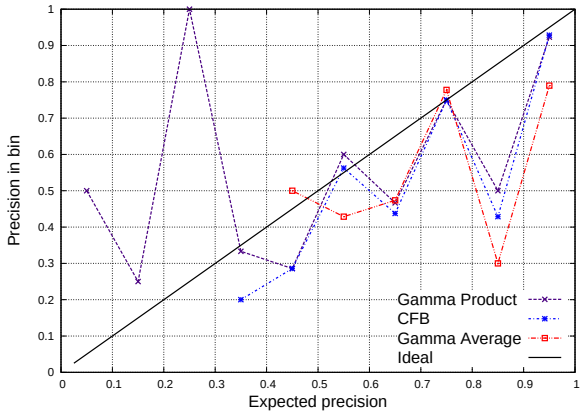
Table 7.4: Average precision and RMSE for the different confidence score methods.



(a) Dutch

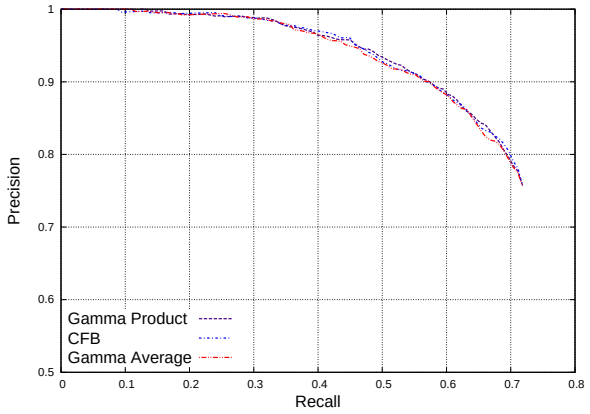


(b) Spanish

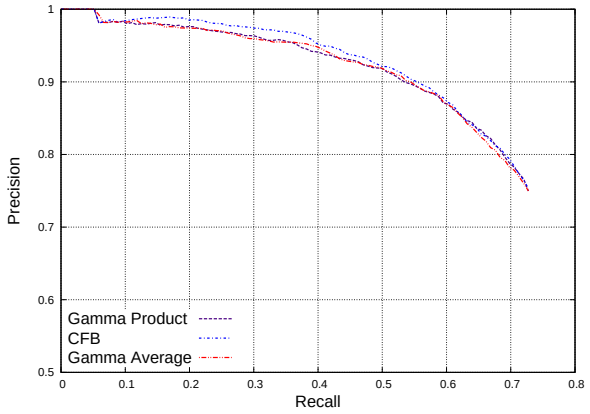


(c) CPROD1

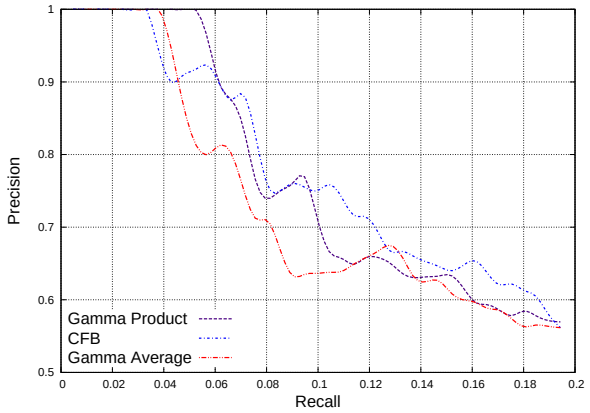
Figure 7.1: The x-axis in these figures shows the expected precision in the bins based on the confidence scores, the y-axis denotes the found precision.



(a) Dutch



(b) Spanish



(c) CPROD1

Figure 7.2: The recall-precision graphs for the different methods.

Chapter 8

Conclusion

The problem of Product Entity Recognition has not been very widely studied and seems to be a hard problem to solve. The research question for this thesis was to find the optimal use of Conditional Random Fields for this problem, together with a sound confidence score. This resulted in an experimental comparison between models with different sets of features. The performance of the models differed slightly, but most performed better than heuristic methods. The best result was obtained by combining the different CRF models in ensembles.

In addition, a method has been proposed to efficiently calculate a statistically sound confidence estimate for segments outputted by a CRF, introduced as the Constrained Forward Backward confidence estimate by Culotta and McCallum [16]. Our experiments have shown that these confidence scores are a good estimate of the expected precision, and they work very well as a ranking score for the mentions. We also showed that the approximative Gamma Product method obtained a similar performance and can also be used in practice.

8.1 Future work

The relatively low F1 score obtained for the CPROD1 dataset (never more than about 40%) suggests that there is still opportunity for improvements on this topic. However, as many contestants of the CPROD1 contest pointed out, the difficulty of the task is at least partially caused by low quality of the annotations in the data. An obvious follow up study would be to repeat the experiments on a re-annotated data set.

The low recall of the models and the success of the ensemble models suggests that it might be valuable to train a wider variety of CRF models with different sets of features. Besides more combinations of features used for our models, there are many possible features not examined in this work that could be profitable. One example of

features that could be explored more are global features, such as word frequencies. Another useful addition could be combination features: if we have binary features `A` and `B` we can also include `A AND B` as a feature. This can especially be beneficial when we combine features from the surrounding tokens in the window with features from the current token. This can increase the feature space significantly, so we suggest to apply feature reduction techniques.

The ensemble models have shown good results in this work, thus exploring them more seems promising. First of all, it is interesting what the effect is of calculating the confidence for models that don't have the segment as the most probable output. We assumed a confidence of zero for these mentions, but it is possible to use the forward-backward lattice to calculate probabilities of segments that are not part of the optimal output. Second, other methods than taking the average of confidences could be explored. Much previous work has been done on ensembles of classifiers [25] that could be extended to ensembles of segmentation models.

Finally, the task of Product Entity Recognition could be extended to wider applications. It would be interesting to develop (possibly hierarchical) models that find not only Products, but also for example Brand, Merchant and Product Category entities. The models can also be generalized to other types of text, for example product reviews, tweets or product catalog entries, or text in other languages than English. All these tasks require new annotated data.

In conclusion, Product entity recognition is a topic with increasing relevance, that asks for more research on high-quality recognition models. This work suggests that Conditional Random Fields provide a good solution to this problem, especially if several feature sets are explored and a number of diverse models is combined in an ensemble.

Bibliography

[1] http://www.cj.com/what-is-affiliate-marketing. [cited at p. 3]

[2] http://www.viglink.com/blog/press/about/. [cited at p. 3]

[3] http://www.viglink.com/blog/2013/05/08/this-old-hyperlink/. [cited at p. 3]

[4] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333–340, June 1975. [cited at p. 9]

[5] Eckhard Bick. A named entity recognizer for danish. In *LREC*, 2004. [cited at p. 30]

[6] Daniel M Bikel, Richard Schwartz, and Ralph M Weischedel. An algorithm that learns what's in a name. *Machine learning*, 34(1-3):211–231, 1999. [cited at p. 30]

[7] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 1, chapter 8. Springer New York, 2006. [cited at p. 17]

[8] Phil Blunsom and Trevor Cohn. Discriminative word alignment with conditional random fields. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 65–72. Association for Computational Linguistics, 2006. [cited at p. 32]

[9] Andrew Borthwick, John Sterling, Eugene Agichtein, and Ralph Grishman. Nyu: Description of the mene named entity system as used in muc-7. In *In Proceedings of the Seventh Message Understanding Conference (MUC-7*, 1998. [cited at p. 30]

[10] Nancy Chinchor and Patricia Robinson. Muc-7 named entity task definition. In *Proceedings of the 7th Conference on Message Understanding*, 1997. [cited at p. 29]

[11] Yejin Choi, Claire Cardie, Ellen Riloff, and Siddharth Patwardhan. Identifying sources of opinions with conditional random fields and extraction patterns. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 355–362. Association for Computational Linguistics, 2005. [cited at p. 32]

[12] William W. Cohen and Sunita Sarawagi. Exploiting dictionaries in named entity extraction: Combining semi-markov extraction processes and data integration method. In *In Proceedings of the ACM SIGKDD Conference*, 2004. [cited at p. 30]

[13] Michael Collins. Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 489–496. Association for Computational Linguistics, 2002. [cited at p. 31, 38]

[14] Peter Corbett and Ann Copestake. Cascaded classifiers for confidence-based chemical named entity recognition. *BMC bioinformatics*, 9(Suppl 11):S4, 2008. [cited at p. 33]

[15] Aron Culotta, David Kulp, and Andrew McCallum. Gene prediction with conditional random fields. Technical report, 2005. [cited at p. 32]

[16] Aron Culotta and Andrew McCallum. Confidence estimation for information extraction. In *Proceedings of HLT-NAACL 2004*, pages 109–112, 2004. [cited at p. 24, 33, 47]

[17] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134, 2005. [cited at p. 30]

[18] Ralph Grishman and Beth Sundheim. Message understanding conference-6: a brief history. In *Proceedings of the 16th conference on Computational linguistics - Volume 1*, COLING '96, pages 466–471, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics. [cited at p. 29]

[19] Forrester Research Inc. Affiliate marketing the direct and indirect value that affiliates deliver to advertisers. commissioned by Rakuten LinkShare, 2012. [cited at p. 3]

[20] Hui Jiang. Confidence measures for speech recognition: A survey. *Speech Communication*, 45(4):455–470, 2005. [cited at p. 33]

[21] A Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 14:841, 2002. [cited at p. 13]

[22] S Sathiya Keerthi and Sellamanickam Sundararajan. Crf versus svm-struct for sequence labeling. Technical report, Technical report, Yahoo Research, 2007. [cited at p. 30]

[23] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001. [cited at p. 11, 32]

[24] Yan Liu, Jaime Carbonell, Peter Weigele, and Vanathi Gopalakrishnan. Segmentation conditional random fields (scrfs): A new approach for protein fold recognition. In *Research in Computational Molecular Biology*, pages 408–422. Springer, 2005. [cited at p. 32]

[25] Richard Maclin and David W. Opitz. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999. [cited at p. 48]

[26] Diana Maynard, Valentin Tablan, Cristian Ursu, Hamish Cunningham, and Yorick Wilks. Named entity recognition from diverse text types. In *In Recent Advances in Natural Language Processing 2001 Conference, Tzigov Chark*, 2001. [cited at p. 29]

[27] Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 591–598, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. [cited at p. 11, 30]

[28] Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 188–191, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. [cited at p. 3, 30]

[29] Andrew Kachites McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002. [cited at p. 37]

[30] Avihai Mejer and Koby Crammer. Confidence in structured-prediction using confidence-weighted models. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 971–981, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. [cited at p. 33, 38]

[31] Gabor Melli and Christian Romming. An overview of the cprod1 contest on consumer product recognition within user generated postings and normalization against a large product catalog. In *Proceedings of the ICDM-2012 Workshop on the CPROD1 Contest*, 2012. [cited at p. 3, 4, 30]

[32] Einat Minkov, Richard C. Wang, and William W. Cohen. Extracting personal names from email: applying named entity recognition to informal text. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 443–450, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics. [cited at p. 29]

[33] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*, pages 684–693. The MIT Press, 2012. [cited at p. 15]

[34] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Linguisticae Investigationes*, 30(1):3–26, January 2007. Publisher: John Benjamins Publishing Company. [cited at p. 29, 30]

[35] Sebastian Nowozin and Christoph H Lampert. *Structured learning and prediction in computer vision*, volume 6. Now publishers Inc, 2011. [cited at p. 33]

[36] Fuchun Peng, Fangfang Feng, and Andrew Mccallum. Chinese segmentation and new word detection using conditional random fields. In *In Proceedings of COLING*, pages 562–568, 2004. [cited at p. 32]

[37] Fuchun Peng and Andrew McCallum. Information extraction from research papers using conditional random fields. *Information Processing & Management*, 42(4):963–979, 2006. [cited at p. 32]

[38] David Pinto, Andrew McCallum, Xing Wei, and W Bruce Croft. Table extraction using conditional random fields. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 235–242. ACM, 2003. [cited at p. 32]

[39] Thierry Poibeau and Leila Kosseim. Proper name extraction from non-journalistic texts. In *In Computational Linguistics in the Netherlands*, pages 144–157, 2001. [cited at p. 29]

[40] Lance A Ramshaw and Mitchell P Marcus. Text chunking using transformation-based learning. In *Natural language processing using very large corpora*, pages 157–176. Springer, 1999. [cited at p. 9]

[41] Alan Ritter, Sam Clark, Mausam, and Oren Etzioni. Named entity recognition in tweets: an experimental study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1524–1534, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. [cited at p. 29, 30]

[42] Lukasz Romaszko. An ensemble-based named entity recognition solution for detecting consumer products. In *Proceedings of the ICDM-2012 Workshop on the CPROD1 Contest*, pages 865–868. IEEE, 2012. [cited at p. 31]

[43] Kengo Sato and Yasubumi Sakakibara. Rna secondary structural alignment with conditional random fields. *Bioinformatics*, 21(suppl 2):ii237–ii242, 2005. [cited at p. 32]

[44] Matthew Stephen Seigel and Philip C. Woodland. Combining information sources for confidence estimation with crf models. In *INTERSPEECH*, pages 905–908, 2011. [cited at p. 33]

[45] Satoshi Sekine and Chikashi Nobata. Definition, dictionaries and tagger for extended named entity hierarchy. In *LREC*, 2004. [cited at p. 30]

[46] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 134–141. Association for Computational Linguistics, 2003. [cited at p. 32]

[47] Charles Sutton and Andrew McCallum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4):267–373, 2012. [cited at p. 11]

[48] Charles Sutton, Andrew McCallum, and Khashayar Rohanimanesh. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *The Journal of Machine Learning Research*, 8:693–723, 2007. [cited at p. 32]

[49] Erik F. Tjong Kim Sang. Introduction to the conll-2002 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2002*, pages 155–158. Taipei, Taiwan, 2002. [cited at p. 29, 36]

[50] Erik F Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 142–147. Association for Computational Linguistics, 2003. [cited at p. 29, 36]

[51] Zhiqiang Toh, Wenting Wang, Man Lan, and Xiaoli Li. An ner-based product identification and lucene-based product linking approach to cprod1 challenge: Description of submission system to cprod1 challenge. In *Proceedings of the ICDM-2012 Workshop on the CPROD1 Contest*, pages 869–871. IEEE, 2012. [cited at p. 31]

[52] Olexandr Topchylo. Identification and disambiguation of product mentions with information retrieval and problem specific methods. In *Proceedings of the ICDM-2012 Workshop on the CPROD1 Contest*, pages 872–873. IEEE, 2012. [cited at p. 31]

[53] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. In *Journal of Machine Learning Research*, pages 1453–1484, 2005. [cited at p. 30]

[54] Yotaro Watanabe, Masayuki Asahara, and Yuji Matsumoto. A graph-based approach to named entity categorization in wikipedia using conditional random fields. In *EMNLP-CoNLL*, pages 649–657. ACL, 2007. [cited at p. 29]

[55] Sen Wu, Zhanpeng Fang, and Jie Tang. Accurate product name recognition from user generated content. In *Proceedings of the ICDM-2012 Workshop on the CPROD1 Contest*, pages 874–877. IEEE, 2012. [cited at p. 31]

[56] Dong Yu, Jinyu Li, and Li Deng. Calibration of confidence measures in speech recognition. *Trans. Audio, Speech and Lang. Proc.*, 19(8):2461–2473, November 2011. [cited at p. 33]

[57] Jun Zhao and Feifan Liu. Product named entity recognition in chinese text. *Language Resources and Evaluation*, 42(2):197–217, 2008. [cited at p. 31]

[58] Jianhan Zhu, Victoria Uren, Enrico Motta, and Jianhan Zhu Victoria Uren. Espotter: Adaptive named entity recognition for web browsing. In *3rd Conf. on Professional Knowledge Management*, pages 518–529, 2005. [cited at p. 30]

List of Symbols and Abbreviations

Abbreviation	Description	Definition
CRF	Conditional Random Field	page 3
NER	Named Entity Recognition	page 5
HMM	Hidden Markov Model	page 11
MEMM	Maximum Entropy Markov Model	page 11
CFB	Conditional Forward Backward	page 24
POS	Part Of Speech	page 32
NP chunking	Noun Phrase chunking	page 32
ASR	Automatic Speech Recognition	page 32

List of Figures

List of Tables