

**RECONHECIMENTO DE ENTIDADES  
NOMEADAS NA WEB**

JOÃO MATEUS DE FREITAS VENEROSO

**RECONHECIMENTO DE ENTIDADES  
NOMEADAS NA WEB**

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR: BERTHIER RIBEIRO-NETO DE ARAÚJO

Belo Horizonte

Agosto de 2019

JOÃO MATEUS DE FREITAS VENEROSO

## NAMED ENTITY RECOGNITION ON THE WEB

Dissertation presented to the Graduate Program in Computer Science of the Universidade Federal de Minas Gerais – Departamento de Ciência da Computação in partial fulfillment of the requirements for the degree of Master in Computer Science.

ADVISOR: BERTHIER RIBEIRO-NETO DE ARAÚJO

Belo Horizonte

August 2019

© 2019, João Mateus de Freitas Veneroso.  
Todos os direitos reservados.

Veneroso, João Mateus de Freitas

V456g      Named entity recognition on the Web / João Mateus  
de Freitas Veneroso. — Belo Horizonte, 2019  
xvi, 101 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de  
Minas Gerais – Departamento de Ciência da  
Computação

Orientador: Berthier Ribeiro-Neto de Araújo

1. Computação – Teses. 2. Recuperação da  
informação. 3. Aprendizado do computador.  
4. Resolução de entidades. 5. Sites da Web. I. Título.

CDU 519.6\*73(043)




UNIVERSIDADE FEDERAL DE MINAS GERAIS  
INSTITUTO DE CIÊNCIAS EXATAS  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

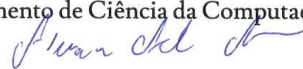
## FOLHA DE APROVAÇÃO

Named Entity Recognition on the Web

**JOÃO MATEUS DE FREITAS VENEROSO**

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

  
PROF. BERTHIER RIBEIRO DE ARAÚJO NETO - Orientador  
Departamento de Ciência da Computação - UFMG

  
PROF. ADRIANO ALONSO VELOSO  
Departamento de Ciência da Computação - UFMG

  
PROF. RENATO MARTINS ASSUNÇÃO  
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 9 de Agosto de 2019.

# Acknowledgments

I am extremely grateful to Berthier for his valuable advice and encouragement during the course of my master's. His guidance was important not only in the intellectual pursuit that led to this dissertation, but in a general sense. His insightful observations about questions that go far beyond Computer Science are an enduring source of inspiration.

I would like to extend my gratitude to Alberto Ueda, for his extraordinarily detailed review that contributed quite a lot to the quality of this work. I also thank him, Filipe, Gabriel, Fabricio, Alef, Amir, and all the lab fellows from LATIN and PENSI for their friendship and support during the whole process. My life has changed forever for the better by getting to know this awesome group of people.

Many thanks to Nivio for his support in the laboratory and for his relentless enthusiasm in promoting science and technology that is a great source of inspiration. Also, many thanks to Adriano for introducing me to the field of Machine Learning and for his very helpful research advices. I also extend my sincere thanks to all professors in the department that inspired me during lessons and day to day conversations.

I would like to gratefully acknowledge the assistance provided by Sonia Borges during my many visits to the postgraduate office. Also, I would like to thank our representatives, Evellyn and Evelin, for their determined struggle to improve the well-being of the other students, many times at the expense of their own welfare.

I am extremely grateful to my mother, my father and my brother for their support in every aspect of life. My mother taught me to value education and to seek knowledge everywhere. My father taught me to work hard to achieve my goals and make a difference. My brother taught me to think big and to think differently. But, above all else, they made me who I am today.

Lastly, I thank all the friends and family that were not directly involved in the process of writing this dissertation but were always present in my life.

# Resumo

Métodos tradicionais de extração de informação na web normalmente utilizam regras rígidas para extrair dados relevantes de páginas da internet. Estes métodos são adequados para resolver tarefas de extração dentro de um mesmo website, mas eles são bem menos eficientes quando a tarefa compreende um conjunto heterogêneo de websites. Por outro lado, modelos de Reconhecimento de Entidades Nomeadas (NER) baseados em aprendizado de máquina oferecem uma alternativa mais flexível para resolver o problema. No entanto, na maior parte das vezes, páginas HTML tem uma organização substancialmente diferente do texto em prosa, porque as frases são muito curtas, o que piora o desempenho dos modelos tradicionais de NER. Em contrapartida, a estrutura do HTML contém informação valiosa que pode ser utilizada para melhorar o desempenho dos modelos de NER. Nós propomos duas formas de utilizar esta informação: a estratégia de auto-treinamento para Hidden Markov Models e o mecanismo de atenção para a Bi-LSTM-CRF, um tipo de rede neural. Além disso, nesta dissertação, nós avaliamos o desempenho de diversos métodos de NER na tarefa de extração de informação na web. Em particular, introduzimos um *dataset* novo que consiste em páginas de departamentos de pesquisa extraídas dos sites de múltiplas universidades ao redor do mundo e testamos os modelos de NER na tarefa de extração de nomes de pesquisadores. Uma arquitetura de redes neurais que combina uma Bi-LSTM-CRF com representações de caracteres baseadas em LSTMs e o mecanismo rígido de atenção tem um desempenho superior aos demais métodos, alcançando um F1 de 90,2 na tarefa. Contudo, por meio da aplicação de estratégias como o auto-treinamento, conseguimos obter um modelo muito mais simples, o Hidden Markov Model de segunda ordem, que alcança um F1 de 87,9 na mesma tarefa.

**Palavras-chave:** Reconhecimento de Entidades Nomeadas, Extração de Informação na Web, Extração de Nomes de Pesquisadores.

# Abstract

Web Data Extraction methods often rely on hand-coded rules to identify and extract data from webpages. These methods are suited for extracting information from pages within the same website, however they perform poorly on extraction tasks across different websites. Alternatively, statistical and machine-learning-based Named Entity Recognition (NER) methods provide a more flexible approach to Web Data Extraction. This is important, because sentences in HTML pages are often too short to provide adequate context for conventional NER methods to work properly. Nonetheless, the HTML structure also encodes useful information that can be used by NER models to achieve a better performance. We propose two methods to use this information: the self-training strategy for Hidden Markov Models and the hard attention mechanism for Bi-LSTM-CRFs, a type of neural network. Also, in this dissertation we evaluate the performance of different methods of NER in the task of Web Data Extraction. In particular, we introduce a novel dataset consisting of faculty listings from university webpages across the world in multiple languages and test different NER models in the task of extracting researcher names from these listings. We found that a neural network architecture that combines a bidirectional LSTM with a Conditional Random Fields output layer, LSTM-based character representations and a Hard Attention mechanism for HTML features outperforms other methods achieving 90.7 F1-score in the task. But, with the aid of clever strategies such as self-training, we can get a much simpler model, the second-order Hidden Markov Model, to achieve a 87.9 F1-score.

**Palavras-chave:** Named Entity Recognition, Web Data Extraction, Researcher Name Extraction.



# List of Figures

2.1	Named Entity Recognition as a sequence labeling task. The "O" stands for a token Outside a named entity. . . . .	10
2.2	Example of a faculty webpage. . . . .	12
4.1	Finite state machine for a Markov Chain. . . . .	24
4.2	Finite state machine for a Hidden Markov Model. . . . .	27
4.3	Example of a webpage snippet with labels attributed by a HMM. . . . .	32
4.4	RNN for NER . . . . .	39
4.5	RNN with a long term memory cell $c$ . . . . .	40
4.6	LSTM Cell . . . . .	41
4.7	Bidirectional LSTM-CRF . . . . .	42
4.8	CNN-based character representations for the word "jaguar". . . . .	45
4.9	LSTM-based character representations for the word "jaguar". . . . .	46
5.1	Word frequencies plot for the CoNLL-2003 dataset and the RNE dataset. . . . .	55
6.1	Performance of the Naive Bayes classifier and Hidden Markov Models with no features besides the current word on the test set of the RNE dataset. . . . .	61
6.2	Hidden Markov Models trained with the features in Group A and Group B. . . . .	63
6.3	HMMs trained with the features in Group A and HMMs trained with features from Group A and self-trained with features from Group C. . . . .	64
6.4	Performance of the Logistic Classifier, second-order HMM and CRF with no features besides the current word on the test set of the RNE dataset. . . . .	65
6.5	CRFs trained with the features from Group A and Group B and the HMM-2 trained with features from Group A and self-trained with features from Group C. . . . .	66
6.6	Performance of the Bi-LSTM-CRF with only GloVe embeddings in comparison to the HMM-2 (Group A) with self-training and the CRF (Group B). . . . .	68

7.1	Attention mechanism for the Bi-LSTM-CRF model. . . . .	74
7.2	Building HTML representations by climbing the DOM tree. . . . .	76
7.3	Results for the Bi-LSTM-CRF + LSTMc that optimized the expected $F_\alpha$ - score function. A larger $\alpha$ means preference for recall rather than precision.	81

# List of Tables

3.1	Model performances in the CoNLL-2003 English test set. . . . .	19
4.1	Statistics of pre-trained word embeddings. . . . .	43
5.1	Description of the data files in the RNE dataset. . . . .	49
5.2	Description of the CoNLL-2003 English dataset . . . . .	50
5.3	DBLP dictionary coverage in each data file of the RNE dataset, when an exact dictionary matching strategy is used. . . . .	52
5.4	Performance of three classifiers trained with the CoNLL-2003 training set and tested in the CoNLL-2003 and RNE test sets. . . . .	53
5.5	Ten most frequent words for the CoNLL-2003 and the RNE datasets. . . .	55
5.6	Number of documents per country for the ten most frequent countries in the RNE dataset and the number of universities in each country according to the Universities Worldwide Database. . . . .	56
6.1	Naive Bayes and Logistic Classifier results for the researcher name extraction task using only the current word as a feature. . . . .	59
6.2	Unknown tokens in the RNE dataset. . . . .	60
6.3	Performance of the Naive Bayes classifier and Hidden Markov Models with no features besides the current word on the validation and test sets of the RNE dataset. . . . .	60
6.4	Features used in the RNE dataset. Feature 10 is the token’s enclosing HTML tag and its parent tag concatenated. . . . .	62
6.5	Performance on the validation and test sets for Hidden Markov Models trained with the features in Group A and Group B. . . . .	63
6.6	HMMs trained with the features in Group A and HMMs trained with features from Group A and self-trained with features from Group C. . . . .	64

6.7	Performance of the Logistic Classifier, second-order HMM and CRF with no features besides the current word on the validation and test sets of the RNE dataset. . . . .	65
6.8	Conditional Random Fields using features from Group A and Group B. . .	66
6.9	Results for the HMM using features from Group A and self-training, the CRF with features from Group B and the Plain Bi-LSTM-CRF. . . . .	67
6.10	Bi-LSTM-CRF with Character Representations. . . . .	69
6.11	Bi-LSTM-CRF with LSTM characters and different sets of word embeddings.	69
6.12	Overview of the best models for the name extraction task in the RNE dataset.	71
7.1	Relationship between positive and negative matches. . . . .	78
7.2	Hard and Soft Attention . . . . .	80
7.3	Bi-LSTM-CRF with LSTM character embeddings and F-score optimization objective. . . . .	82
7.4	Overview of the best models for the name extraction task in the RNE test set after the filtering strategy. . . . .	83
7.5	Overview of the subjective complexity and training times of the best models for the RNE task. . . . .	84
A.1	Precision (P), Recall (R) and F1 for Hidden Markov Models and Naive Bayes in the RNE task using five fold cross validation (CV). . . . .	99
A.2	Precision (P), Recall (R) and F1 for CRFs and Logistic Classifier in the RNE task using five fold cross validation (CV). . . . .	100
A.3	Precision(P), Recall (R) and F1 for Bi-LSTM-CRF variations in the RNE task using five fold cross validation (CV). All the models used GloVe-300 embeddings except if stated otherwise. . . . .	101
A.4	Precision(P), Recall (R) and F1 for Bi-LSTM-CRF variations optimizing the $F-\alpha$ objective in the RNE task using five fold cross validation (CV). . .	101

# Glossary

**CNN** Convolutional Neural Networks.

**CRF** Conditional Random Fields.

**HMM** Hidden Markov Models.

**IE** Information Extraction.

**LSTM** Long Short-Term Memory Networks.

**NER** Named Entity Recognition.

**NLP** Natural Language Processing.

**POS** Part-of-Speech.

**RNE** Researcher Name Extraction.

**RNN** Recurrent Neural Networks.

**WDE** Web Data Extraction.

# Contents

Acknowledgments	vi
Resumo	vii
Abstract	viii
List of Figures	ix
List of Tables	xi
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	3
1.2 Objective . . . . .	4
1.3 Contributions . . . . .	4
1.4 Dissertation Outline . . . . .	5
<b>2 Problem Definition</b>	<b>6</b>
2.1 Information Extraction . . . . .	7
2.2 Named Entity Recognition . . . . .	9
2.3 Researcher Name Extraction . . . . .	11
2.4 Summary . . . . .	13
<b>3 Related Work</b>	<b>14</b>
3.1 Web Data Extraction . . . . .	14
3.2 Named Entity Recognition . . . . .	17
3.3 Summary . . . . .	19
<b>4 Techniques for Sequence Labeling</b>	<b>20</b>
4.1 Hidden Markov Models . . . . .	21
4.1.1 Smoothing . . . . .	28

4.1.2	Predicting sequences . . . . .	28
4.1.3	Self-Training . . . . .	29
4.1.4	Experimental Considerations . . . . .	32
4.2	Conditional Random Fields . . . . .	32
4.2.1	Experimental Considerations . . . . .	37
4.3	Neural Networks . . . . .	38
4.3.1	BI-LSTM-CRF . . . . .	41
4.3.2	Word Embeddings . . . . .	42
4.3.3	Character Representations . . . . .	44
<b>5</b>	<b>Researcher Name Extraction Dataset</b>	<b>47</b>
5.1	Data Description . . . . .	48
5.2	Evaluation . . . . .	50
5.3	Dictionary . . . . .	51
5.4	Comparison with CoNLL-2003 . . . . .	52
5.5	Dataset Size . . . . .	54
<b>6</b>	<b>Experiments</b>	<b>57</b>
6.1	Simple Baseline . . . . .	58
6.2	Hidden Markov Models . . . . .	59
6.2.1	HMM order . . . . .	60
6.2.2	Feature selection . . . . .	61
6.2.3	Self-training strategy . . . . .	63
6.3	Conditional Random Fields . . . . .	64
6.3.1	Only the Current Word Feature . . . . .	65
6.3.2	Feature Selection . . . . .	66
6.4	Neural Networks . . . . .	67
6.4.1	Bi-LSTM-CRF . . . . .	67
6.4.2	Character Representations . . . . .	68
6.4.3	Word Embeddings . . . . .	69
6.4.4	Technical Details for Neural Networks . . . . .	70
6.5	What is the best model? . . . . .	70
<b>7</b>	<b>Improvements to Neural Networks</b>	<b>73</b>
7.1	Attentions Models . . . . .	73
7.1.1	Hard Attention Function . . . . .	75
7.1.2	Soft Attention Function . . . . .	75
7.1.3	Dataset Split and Experimental Considerations . . . . .	76

7.2	F-score Optimization . . . . .	77
7.3	Experiments . . . . .	79
7.3.1	Attention Mechanisms . . . . .	79
7.3.2	F-score Optimization . . . . .	80
7.3.3	Filtering False Positives . . . . .	82
<b>8</b>	<b>Conclusions and Future Work</b>	<b>85</b>
8.1	Summary of Conclusions . . . . .	85
8.2	Future Work . . . . .	86
8.3	Final Remarks . . . . .	87
	<b>Bibliography</b>	<b>88</b>
	<b>Appendix A Cross Validation</b>	<b>97</b>
A.1	Hidden Markov Models . . . . .	98
A.2	Conditional Random Fields . . . . .	99
A.3	Neural Networks . . . . .	99
A.3.1	F-score Optimization . . . . .	100



# Chapter 1

## Introduction

The amount of data generated in almost every imaginable human endeavor is increasing at an accelerating pace, and the astonishing growth of the Web is perhaps the most important manifestation of this movement. The recurrent appearance of terms such as *Big Data* and *Artificial Intelligence* in mass media testify to the growing attention devoted to this topic. People, companies and academia are all interested in taming the colossal flood of data, each for their own reasons. Despite all this interest, meaningful information in the Web gets frequently lost over torrents of unimportant data. As a result, to extract structured information from this haystack we need sophisticated solutions.

Web Data Extraction (WDE) methods are often employed when there is the need to extract massive amounts of structured information from webpages. WDE is basically the task of extracting useful information from unstructured Web sources. In this sense, it is a specific setting of the more general problem of Information Extraction (IE), that regards extraction tasks in any type of document. Some examples of such tasks are the extraction of product descriptions and their prices from online shops, or the extraction of house locations and number of bedrooms from real estate portals. Web documents however, most often lie in between the structured/unstructured data paradigm. This means that they are not structured in the same sense that a relational database is structured, neither are they unstructured in the same sense that plain text is unstructured. That is, the way that elements are organized and displayed in a webpage contribute to their meaning. For example, an entry in a list is more meaningful than an isolated entry, because the entry's meaning becomes evident once we know the listed category. Yet we cannot expect that such organizational patterns will be completely constrained by an underlying set of rules. Patterns tend to follow some guidelines but they are in no way subject to strict rules.

Traditional WDE methods often relied on the identification of webpage patterns such as listings and tables to build wrappers, simple programs aimed at extracting relevant entities from text by identifying their common boundaries. These programs were somewhat prone to failure when the webpage changed, and for this reason they demanded constant maintenance. Some improvements were made with the invention of automatic wrapper generators such as WIEN [Kushmerick, 2000], Soft Mealy [Hsu and Dung, 1998] and STALKER [Muslea et al., 1999], that aimed to reduce the cost of wrapper maintenance. However, wrappers generated by these systems only worked well on webpages with a very similar structure (e.g. product listings from Amazon). In fact, the problem of data extraction across different websites has not yet been solved effectively, even though some remarkable improvement was made with tools that build upon the tradition of wrapper generators such as ObjectRunner [Abdessaem et al., 2010], Automatic Wrapper Adaptation [Ferrara and Baumgartner, 2011], AMBER [Furche et al., 2012], and AutoRM [Shi et al., 2015], among others.

Comparatively, statistical machine learning provides more robust and flexible methods to WDE. In recent years, we saw amazing progress in the field of Natural Language Processing (NLP) that is extremely relevant to the WDE community, particularly with the introduction of Deep Neural Networks for Sequence Labeling by Collobert et al. [2011], but these advancements were not widely incorporated by WDE systems. Also, most of the attention of the NLP community regarding this topic is devoted to solving IE tasks in plain text, such as the identification of people and organizations in news texts. However, the difference between some types of webpages and plain text is significant, so IE systems trained in plain text datasets tend to perform poorly when confronted with some Web extraction tasks, as we have confirmed in this dissertation.

A concrete example of a WDE problem that demands the type of flexible solution provided by statistical machine learning methods is the extraction of researcher names from faculty directories. Automatically extracting researcher information from university websites is important, for example, to construct researcher affiliation databases and compare the academic impact of research groups using bibliographic indices [Ribas et al., 2015; Hirsch, 2005]. The problem of Researcher Name Extraction (RNE) can be solved with machine learning methods of Named Entity Recognition (NER), the task of finding entities such as people, organizations, and locations in a text.

Many IE tasks in the Web share a common factor with the RNE task. That is, they need to extract similar entities from different webpages or they need to extract entities from short snippets of text instead of complete sentences. Some examples would be comment extraction from blog posts and NER in tweets. Because of the lack

of context in tweets, NER methods trained in conventional plain text corpora tend to perform badly for more or less the same reasons they perform badly in other WDE tasks.

In this dissertation, we compare different machine learning approaches to NER on the Web by evaluating their performance in the RNE task. With this purpose in mind, we introduce a novel NER dataset with labeled entities, consisting of faculty directories from universities across the world. We compare the models in terms of their precision, recall, F-scores and their overall complexity. That is, even if a model has a slightly worse performance in terms of the objective evaluation metrics in comparison to other approaches, it may still be useful if it is considerably simpler and faster to train. The tested models were the Hidden Markov Models (HMM) up to third order, Conditional Random Fields (CRF), and a range of Deep Learning architectures based on Bidirectional Long Short-Term Memory Networks (LSTM). We also introduce three strategies that can boost the performance of sequence labeling methods for HTML. The self-training strategy for HMM, the Self-Attention layer with HTML features for Neural Networks, and the direct F-score optimization, also for Neural Networks.

The usability of the evaluated models is not limited to the particular problem of RNE. By reliably detecting named entities on the Web, we can boost the performance of existing WDE approaches or even construct an end-to-end architecture that solves the problem of data extraction across different websites with improved robustness. Also, the researcher affiliation database constructed with the application of these methods can be used later on bibliometric studies.

## 1.1 Motivation

Web Data Extraction emerges in the context of many Web projects. It consists of extracting structured information from unstructured web pages to build a knowledge base. Wrapper-based methods are effective for extracting regular data from similar webpages, for example, when extracting prices from a web catalog. However, when a particular extraction task requires more flexibility, such as in the cases where the website layout varies significantly between different sources, extraction systems based on machine learning usually perform better. In the cases where these machine learning systems need to handle complex entities such as person or organization names, NER comes forward as an essential subtask. Nevertheless, most of the research concerning NER only handles extraction tasks in plain text, which can be quite different from HTML webpages. HTML webpages may contain plain text, however they usually have

a smaller average sentence size and a considerable amount of boilerplate text that may hurt the performance of regular NER models trained to solve extraction tasks in plain text. Also, the webpage layout contains useful information that is typically not present in plain text.

A good example of a Web Data Extraction task that requires the type of flexible approach discussed above is the task of acquiring affiliation data from university websites to build a researcher affiliation database or complement existing ones. One way to complete this task is to crawl university websites, find faculty directories for each department, extract the researcher names and link these names to their public profiles or create new ones if they do not already exist. Acquiring this data manually is tedious and impractical because of the sheer number of universities and researchers in the world<sup>1</sup>, so we must resort to automatic methods.

The crawling stage is relatively simple once we have a somewhat reliable way of detecting faculty webpages, but the researcher name extraction stage is trickier. Faculty listings vary widely between different universities, and they are not always available in English, so we need an extraction system that can handle the possible variations without needing to be trained again for each website.

The problem of RNE provides a good study case to attest the performance of different NER systems on the Web, because the named entities concerning this problem are complex, the faculty directories are usually very different from plain text and the documents vary widely between different university websites.

## 1.2 Objective

The main objective of this dissertation is to find the best NER methods for the Web in terms of their precision, recall, and F-scores in the RNE task, considering their overall complexity and flexibility, taking into consideration that there is a trade-off between model complexity and objective performance. Therefore, to accomplish this goal we need to perform both a quantitative and qualitative assessment of different methods.

## 1.3 Contributions

The main contributions of this dissertation are:

---

<sup>1</sup>The field of Computer Science alone has more than 2,200,000 researchers listed on DBLP: <https://dblp.uni-trier.de/>.

1. Labeled RNE dataset: we introduce a labeled NER dataset specific for the task of extracting researcher names from faculty directories in different languages across 42 countries with the aim of comparing different extraction systems.
2. Comparison of models for NER on the Web: we compare the performance of HMMs, CRFs and state-of-the-art Neural Networks for NER on the Web, describing their relative advantages and disadvantages in terms of quantitative measures such as the F-score and qualitative aspects such as complexity and difficulty of training.
3. Self-training strategy for Hidden Markov Models: we propose a self-training strategy to improve the performance of HMMs on HTML NER.
4. Attention mechanism for Neural Network sequence taggers: we propose an attention mechanism for improving the performance of Neural Networks on HTML NER.
5. F-score optimization for Neural Networks: We propose an optimization objective based on the F-score to control how much a Neural Network values recall over precision.

## 1.4 Dissertation Outline

The organization of this dissertation is as follows: Chapter 2 (Problem Definition) defines the problem of RNE and discusses how it relates to the broader problems of IE and NER. Chapter 3 (Related Work) describes the literature concerning WDE and NER that is related to the models explored in this dissertation. Chapter 4 (Techniques for Sequence Labeling) describes HMMs, CRFs and Deep Neural Networks for performing NER. Chapter 5 (Researcher Name Extraction Dataset) presents the RNE Dataset that concerns the NER on HTML task of extracting research names from faculty directories. Chapter 6 (Experiments) presents the results of multiple experiments with the models from Chapter 4 in the RNE dataset introduced in Chapter 5. Chapter 7 (Improvements to Neural Networks) describes variations and improvements to the neural networks described in Chapter 4 and presents some additional experiments. Chapter 8 (Conclusion) discusses the conclusions to the research questions that guided the experiments and proposes future work to be done in the field.

# Chapter 2

## Problem Definition

WDE is the task of extracting relevant information from Web documents. Traditional methods of WDE were mainly concerned with the extraction of entities described by a simple ontology from webpages generated with the same template (e.g. collecting house prices and addresses from real estate portals). This case could be solved with rigid tools employing hard coded rules. For example, a simple pattern matching strategy can yield close to perfect results in a price extraction task, since there is not a lot of variation in the way prices are presented in a webpage. However, the extraction of complex entities from many sources presents a different class of problems. These problems demand more flexible approaches since the extraction tools are required to deal with a greater variety of page arrangements and with entities that are ambiguously defined. In many aspects, the flexible extraction of entities from webpages is more similar to the tasks of NER in plain text than to the tasks that concerned early WDE methods.

Recent advancements in sequence modelling for natural language (i.e. modelling text as a sequence of words or characters) led to important breakthroughs in applications such as language modelling [Peters et al., 2018; Devlin et al., 2018], machine translation [Bahdanau et al., 2015; Vaswani et al., 2017], and sequence labeling [Collobert et al., 2011; Lample et al., 2016; Ma and Hovy, 2016] (which concerns us more directly). In fact, if we treat all sentences in a webpage as sequences extracted from an underlying presentation graph (i.e. the DOM tree), the problem of WDE can be, in many cases, solved with the consecutive application of three NLP techniques: sentence segmentation, named entity recognition, and relationship extraction. First, we need to segment the relevant grouping structures (e.g. sentences, rows in a table, items in a list). Then, we must identify relevant named entities (e.g. person names, companies, locations). And finally, we need to discover the relationships between these named entities (e.g. person X works in company Y). This last step is optional depending on

the task at hand. The work flow is the same for plain text and webpages, but there are important differences chiefly related to the structure of the data or rather the lack of structure.

In this dissertation, we investigate the best methods for sequence labeling on HTML. To evaluate these methods, we assess their performance on the task of RNE from faculty directories of universities across the world. We will be mainly concerned with NER, but a brief discussion of the challenges involved in the task of sentence segmentation for HTML will be provided in Chapter 5. The task of relationship extraction, however, is not relevant to the problem of RNE because we only consider one type of named entity (researchers) and we are only concerned with mapping relations between researchers and universities (affiliations) that are obvious since we know which university website produced each faculty listing.

The remainder of this chapter will describe in more depth the subject of this dissertation. Section 2.1 discusses the problem of IE. Section 2.2 discusses the importance of NER for WDE. Finally, Section 2.3 describes the specific problem of RNE.

## 2.1 Information Extraction

IE consists of mapping unstructured or poorly structured data to a semantically well defined structure. It “is the process of filling the fields and records of a database from unstructured or loosely formatted text” [McCallum, 2005]. Usually, the input consists of a corpus containing useful entities that are scattered in the text and the IE system is responsible for finding these entities and organizing them according to a rigid hierarchy, such as the one defined by the schema of a relational database. It must be stated that it is somewhat misleading to refer to plain text as unstructured data, since prose has a loosely defined structure that ultimately renders it comprehensible. However, in the context of IE, we refer to unstructured data in contraposition to tabular data (e.g. XML, SQL tables, etc.), which are in most cases easier to work with than plain text.

IE is a multifaceted research topic that spans communities of researchers in the fields of Text Mining, Information Retrieval, and Natural Language Processing. That is:

- Text mining is the search of patterns in unstructured text. This may involve document clustering, document summarization, IE, and other subtasks.
- Information Retrieval is typically concerned with the parsing, indexing and retrieval of documents. In this case, IE methods can help giving a more precise answer to the user’s information needs.

- NLP is a field of Computer Science concerned with how computers process and understand natural language of which two subtasks, namely NER and Relationship Extraction, are of special importance to IE.

A popular application of IE is the identification of entities such as people, organizations, or events in news sources and the determination of their relations. For example, one could be interested in determining who is the CEO of a company that was mentioned in the news or which politicians support a bill that is being considered by Congress. Another interesting news-related application of IE is tracking disease outbreaks through the extraction of disease names and locations from news sources and determining their relation to outline the geographical area affected by an epidemic [Grishman et al., 2002].

The field of bio-informatics has also found important applications for IE. The first BioCreative challenge dealt with the “extraction of gene or protein names from text, and their mapping into standardized gene identifiers for three model organism databases” [Hirschman et al., 2005]. In the BioCreative/OHNL Challenge 2018 [Rastegar-Mojarad et al., 2018], researchers were required to investigate methods of IE for acquiring family history data, given that family history is a critical piece of information in the decision process for diagnosis and treatment of diseases. The difficulty lies in the fact that the main sources of data are unstructured electronic health records. The task was divided in two subtasks: 1) entity recognition such as family members and disease names; and 2) relation extraction such as the relation between family members and corresponding observations.

An application of IE that concerns this dissertation specifically is the extraction of information from research papers to populate citation databases and bibliography search engines such as CiteSeer [Lawrence et al., 1999], DBLP <sup>1</sup>, Semantic Scholar <sup>2</sup>, and Google Scholar <sup>3</sup>. The vast amount of scientific knowledge produced daily demands automatic methods to extract bibliometric information such as authors, titles, affiliations, references, venues and the year of publication from research papers and academic websites. This application is especially important to the evaluation and bibliometrics research communities, that are concerned with the measure of academic impact and researcher productivity through the usage of quantitative indices of academic impact such as raw citation counts, the H-index [Hirsch, 2005] and P-score [Ribas et al., 2015].

WDE is the task of IE on webpages. It works differently from IE in plain text because HTML documents frequently have more structured content with less context

---

<sup>1</sup><http://dblp.uni-trier.de/>

<sup>2</sup><https://www.semanticscholar.org>

<sup>3</sup><https://scholar.google.com/>



than plain text. In HTML, relevant entities may occur inside tables, lists, or other types of visual elements that provide little to no contextual information that could give hints about an entity’s category. That is, sentences in HTML tables or lists are very short and provide little textual information. Contrastingly, in plain text, entities are usually preceded and succeeded by discourse that may provide textual evidence about an entity (e.g. John is a man, Mary is a woman). Webpages have a two dimensional tabular structure that is usually more similar to a spreadsheet than to text from news corpora or literary works. For this reason, features extracted from the DOM hierarchy such as element disposition, CSS classes, and nesting structure can provide valuable information in identifying entities and extracting their attributes.

Most existing WDE methods are tailored to extract data from webpages generated from the same template, producing different compromises between efficacy and degree of human supervision. Usually, these methods work in two steps. In the record segmentation step, we seek to cluster visually and structurally similar webpage regions and identify repeating data records. In the attribute labeling stage, we seek to identify the correct attributes for each data record, maybe resorting to regular expressions or simple dictionary matching strategies depending on the task at hand. The outcome of each step can aid one another. The inner patterns of data records can help identifying attributes in other data records. Also, by properly identifying attributes, it becomes easier to determine boundaries and perform record segmentation correctly.

Some tasks can likely be solved by a rather inflexible system that operates mainly with hard coded rules and simple regular expressions, especially if we only consider pages with a very similar template (e.g. Amazon product listings). Traditional wrapper generators are well suited for this type of task. However, when we need to identify more complex entities such as researcher names, we might be better off with a more flexible approach. The task of NER aims to identify named entities (e.g. people, organizations, etc.) usually in plain text, but we will see that the sequence models that work well in plain text can also be employed successfully in WDE, sometimes with a few alterations.

This section gave a brief introduction to IE, but there are many applications that were not discussed here. A more detailed view of the field is given in the survey by Sarawagi [2008].

## 2.2 Named Entity Recognition

NER is a task in NLP that aims to identify named entities in a text. The NER problem definition first appeared as a subtask of IE in the context of Third the Message

Understanding Conference (MUC-3) promoted by the American Naval Ocean Systems Center in 1991. In MUC-3 [Sundheim, 1991], the task involved extracting information on terrorist incidents (incident type, date, location, perpetrator, target, instrument, outcome, etc.) from messages disseminated by the Foreign Broadcast Information Service of the U.S. Government. In MUC-6 [Grishman and Sundheim, 1996], the named entity task was created with the goal of identifying the names of people, organizations, and geographic locations from articles of the Wall Street Journal. In MUC-7, the task was expanded to handle multilingual evaluation and Named Entities (NE) were defined as proper names and quantities of interest. Person, organization, and location names were marked as well as dates, times, percentages, and monetary amounts according to Chinchor [1998]. The shared-task at the Conference on Computational Natural Language Learning in 2003, CoNLL-2003 [Tjong Kim Sang and De Meulder, 2003], concerned language-independent NER and was especially important because it established an enduring data format for NER. Further, it introduced a dataset of articles extracted from news sources that is commonly used to evaluate the quality of NER systems.

NER is essentially a sequence labeling task. That is, given a sequence of tokens, we want to attribute labels to each token, classifying them into one of a limited set of predefined classes. Figure 2.1 shows how to attribute named entity labels to a sentence according to the format defined in CoNLL-2003. Regular entities such as dates

Julian Assange, the founder of Wikileaks, was arrested Thursday at the Ecuadorean Embassy in London.

I-PER I-PER O O O I-ORG O O I-DATE O O I-ORG I-ORG O I-LOC

**Figure 2.1.** Named Entity Recognition as a sequence labeling task. The "O" stands for a token Outside a named entity.

and prices can be extracted with almost perfect accuracy using regular expressions, which are search patterns that describe a regular language and that can be easily implemented in most programming languages. Entities that belong to a limited set, such as the names of states in a country, can also be easily extracted with simple dictionary matching. However, other types of entities, such as the ones investigated in the CoNLL-2003 challenge, require more sophisticated methods for sequence labeling.

Some traditional statistical methods that can label complex entities with good accuracy are Hidden Markov Models [Bikel et al., 1999], Maximum Entropy Markov Models [McCallum et al., 2000], and Conditional Random Fields [Lafferty et al., 2001]. These statistical approaches have been frequently employed to sequence labeling tasks

and still provide fairly good solutions because of their simplicity, speed and accuracy. However, they are rapidly being replaced by Deep Neural Networks.

Recent research in deep learning has brought great advancements to NER. Some examples are the LSTM-CRF [Huang et al., 2015] and neural character representations [Lample et al., 2016; Ma and Hovy, 2016]. Deep learning differs from classical machine learning in regard to the levels of abstraction learned by the classifiers. Deep learning techniques combine feature extraction and classification in a single system. While a conventional feed-forward neural network may perform classification by learning the weights of a single hidden layer through backpropagation, a deep learning model is usually composed of multiple hidden layers that handle different levels of abstractions. In text related tasks, the first level of abstraction usually consists of a word embedding layer, where words are mapped to a continuous vectorial space with reduced dimensionality, and the next layer usually consists of a multi-layered recurrent neural network or a convolutional neural network.

Essentially all the best scoring models to date at the CoNLL-2003 task employ some form of deep learning, and most often LSTMs. When combined with pre-trained word embeddings, neural networks provide a powerful method of sequence labeling without any feature engineering or dictionary matching. However, these models have become quite complex, contrasting with earlier approaches that only required the estimation of a comparatively small number of weights. The training time required by these deep models is many orders of magnitude longer than that of traditional approaches, and if we take into consideration the training of word embeddings such as Word2Vec’s skip gram model in a billion token corpus [Mikolov et al., 2013], the difference gets even more substantial. Additionally, deep learning models usually require expensive hardware for training to become practical options.

In summary, the aforementioned sequence labeling models are exactly what we are looking for in the RNE task. By labeling tokens in a webpage with machine learning models, we can extract researcher names with more flexibility than that provided by traditional tools for WDE.

## 2.3 Researcher Name Extraction

The objective of this dissertation is to find the best sequence models for NER on the Web by assessing their performance in the RNE task. This task consists of extracting researchers names from university faculty listings across the world with the purpose of discovering their affiliations and linking their profiles to public databases such as

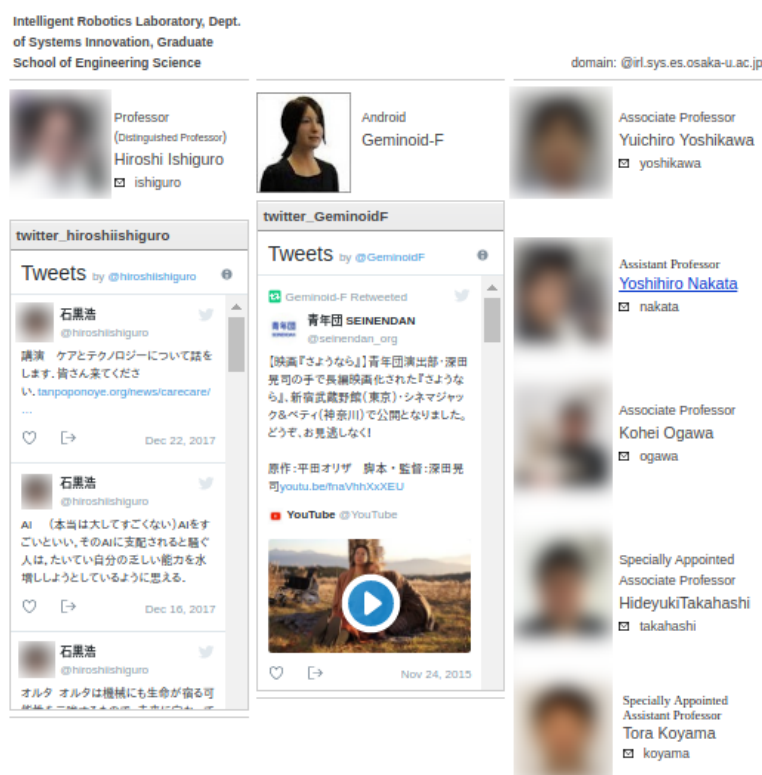


Figure 2.2. Example of a faculty webpage.

DBLP or Google Scholar<sup>4</sup>. Public researcher databases only have sparse information about author affiliation and, even in fields for which the information is more easily available, such as Computer Science, only a small fraction of the records have reliable affiliation information, as we have verified in our preliminary studies. To acknowledge the complexity of this task, take for example a snippet extracted from the staff page for the intelligent robotics laboratory from Osaka University shown in Figure 2.2. There is some pattern to the way member profiles are arranged, but the organization is rather flexible. There are different character encodings in the page, there is some variation to the presentation style of each researcher data record (some are links, some are text, the font varies, etc.), and also, even the laboratory's android makes an appearance in a way that could be easily confused with a researcher. Other pages, even from the same website, can show very different patterns, ranging from tables and lists to free form. Researcher names can appear inside plain text, similar to the case of NER in news texts or in a more tabular structure. Names may also be part of larger sentences such as in "Michael Johnson Chair" and "John Doe Avenue" yielding false positives. Or they can be composed of common words, e.g. Summer Hall, yielding false negatives.

<sup>4</sup> This is useful, for instance, if one needs to compare the research output of departments in a country, or study international publication patterns.

There is no extraction rule that can fit all cases, so we need flexible solutions.

State-of-the-art NER models trained on news datasets do not perform well at this task, because in many webpages, textual information alone is insufficient to indicate the semantic category of a word. The absence of context demands extraction systems to rely on information from sources outside the text, being them features extracted from unlabeled corpora obtained through unsupervised pre-training, dictionaries containing instances of the relevant entities, HTML structural features, or other clever solutions. That is, a key problem in the task of entity name extraction is accounting for every possible name combination. Since there is no database with all possible named entity combinations, we need holistic statistical methods that can handle unknown tokens with relative efficacy. In Chapter 5, we introduce the RNE dataset to evaluate the performance of different sequence models in the RNE task.

Lastly, another important related task in the context of RNE is performing named entity disambiguation, which consists of linking the extracted named entities to a unique profile in a database. This task is called Named Entity Linking. In preliminary studies, we verified that roughly half the records found in faculty webpages can be linked to the respective records in public databases with the aid of string matching techniques. For the other half, we may need to employ more complex systems or perform manual classification. This task is also important, but it will not be covered in the present study.

## 2.4 Summary

This chapter discussed the broad problem of IE in Section 2.1, presenting its challenges and relations to different fields of Computer Science, and showing how it encompasses the problem of WDE. Despite being a specific setting of IE, WDE has specificities that demand solutions that are different from those used for plain text. In Section 2.2, we discussed the related task of NER, which is very useful to IE both in plain text and Web extraction tasks. Recent developments in NER can help WDE systems become more flexible, handling extraction tasks across many website formats. Lastly, in Section 2.3, we discussed the problem of RNE, which is the main topic of this dissertation, showing how it can be challenging and why traditional solutions to WDE and NER may have difficulty handling this specific setting. In fact, by combining the valuable knowledge from both fields of study, we can get a more robust approach to RNE.

# Chapter 3

## Related Work

The RNE task is a problem in WDE that we proposed to solve with the application of machine learning methods of NER. This chapter presents the related research in the fields of WDE in Section 3.1 and NER in Section 3.2.

### 3.1 Web Data Extraction

Since the early 1990s, the astonishing growth of public information in the Web has led to the development of a number of different approaches to the problem of WDE. Traditionally, the task was solved by designing special purpose programs called wrappers to recognize relevant data and store records in a structured format. These early tools varied wildly relative to their degree of automation.

It was readily perceived that manual wrapper generation was a rather tedious and error prone process, unsuited for large scale operations. Wrappers tend to break frequently because they rely on webpage features that can change often. So, in the late nineties, several authors advocated for wrapper induction, a technique that consists of automatically constructing wrappers from a small set of examples by identifying delimiters or context tokens that single out the desired attributes. Some remarkable wrapper induction methods are WIEN [Kushmerick, 2000], Soft Mealy [Hsu and Dung, 1998] and STALKER [Muslea et al., 1999].

Despite being better than constructing wrappers manually, wrapper induction methods still suffered from a lack of expressive power and flexibility. These methods had trouble handling records with missing attributes or unusual structures because patterns could only be identified if they happened at least once in the examples.

Other approaches such as NoDoSE [Adelberg, 1998] and Debye [Laender et al., 2002a] brought greater flexibility to wrapper induction methods by requiring a greater

level of human interaction through graphical user interfaces. WDE techniques often require some sort of assistance from human experts to boost accuracy. One of the main challenges in the field lies in determining an adequate trade-off between the degree of automation and the precision and recall of the data extraction tool.

To automate the task of WDE completely some approaches, such as Road Runner [Crescenzi et al., 2001], removed entirely the need for data examples. Road Runner parses documents belonging to a same class (e.g. books in Amazon) and generates wrappers based on their similarities and differences, yielding comparable results to those obtained by wrapper induction methods. However, like previous approaches, it was unsuited for cross site extraction tasks because the learned rules were not general enough.

Early NLP-based approaches aimed at extracting more general rules that could possibly be employed over multiple websites. RAPIER [Califf and Mooney, 1999] is a method of rule extraction that uses information such as part-of-speech tags and semantic classes from a lexicon to derive patterns from a set of training examples. This approach is more flexible than the wrapper induction methods, however it achieves much lower rates of precision and recall.

In 2002, a survey [Laender et al., 2002b] made a thorough classification of the early approaches with a taxonomy based on their main technology, being them: languages for wrapper development, HTML-aware tools, NLP-based tools, Wrapper Induction Tools, Modeling-based tools and Ontology-based tools. Some noteworthy examples from this era are:

- TSIMMIS [Hammer et al., 1997] and WebOQL [Arocena and Mendelzon, 1998], which are special purpose languages for building wrappers.
- Road Runner [Crescenzi et al., 2001], XWRAP [Liu et al., 2000] and W4F [Sahuguet and Azavant, 1999], which are HTML-aware tools that infer meaningful patterns from the HTML structure.
- RAPIER [Califf and Mooney, 1999], SRV [Freitag, 1998], WHISK [Soderland, 1999], which are NLP-based tools.
- WIEN [Kushmerick, 2000], Soft Mealy [Hsu and Dung, 1998] and STALKER [Muslea et al., 1999] which are wrapper induction methods.
- NoDoSE [Adelberg, 1998] and Debye [Laender et al., 2002a], which are semi-supervised tools that require some interaction with the user by means of a graphical user interface.

Chang et al. [2006] complemented the previous surveys with semi-supervised technologies such as Thresher [Hogue and Karger, 2005], IEPAD [Chang et al., 2001] and OLERA [Chang and Kuo, 2004]. They differed from supervised and unsupervised methods because they either needed only a rough description of data from users for extraction rule generation or some level of post processing that needed user attention.

Most of the early WDE systems were rule-based with either manual rule description or automatic rule learning from examples, thus they suffered from a lack of flexibility when dealing with noisy and unstructured data. Huge progress in the field of statistical learning led to the development of models that tried to solve this problem.

Sarawagi [2008] introduced a classification that segmented wrappers in rule-based methods, statistical methods and hybrid models, bringing together the fields of NER, Relationship Extraction and WDE. The rule based methods encompass most of the previous models. While the statistical methods convert the extraction task into a token labeling task, solved with NER and Relationship Extractions methods. Traditionally, these subtasks were solved with generative models based on HMMs or discriminative models based on the Maximum Entropy principle, but recently these have been largely superseded by Deep Neural Networks. Different from automatic wrapper generators, statistical methods are suitable for a large variety of tasks, especially when we want the system to handle cross website information extraction and plain text information extraction. That is why this class of methods is of special interest to our application. The progress of statistical models will be discussed in Section 3.2.

Surveys by Ferrara et al. [2014], Schulz et al. [2016] and Varlamov and Turdakov [2016] updated the previous surveys on Information Extraction methods with some interesting innovations. Some examples are: the Visual Box Model [Krüpl et al., 2005], a data extraction system that produces a visualization of the webpage to exploit visual cues to identify records presented in a tabular form; automatic wrapper adaptation [Ferrara and Baumgartner, 2011], a technique that tries to reduce the cost of wrapper maintenance by measuring the similarity of HTML trees and adapting wrappers to the new page structure; AutoRM [Shi et al., 2015], a method to mine records from a single webpage by identifying similar data regions through DOM tree analysis; and Knowledge Vault [Dong et al., 2014], a method that combines different extraction approaches to feed a probabilistic knowledge base.

Most data extraction systems focus on extracting information from single websites and are therefore unsuited for cross website extraction tasks. Even unsupervised approaches that are application domain independent, such as RoadRunner and EXALG, only work well when extracting data from pages generated from a same template.

A statistical approach to unsupervised domain independent WDE was described



by Zhu et al. [2005]. The 2D CRF model takes a webpage segmented into data blocks and employs a two dimensional CRF model to perform attribute labeling. The model was further improved in Zhu et al. [2006] to model record segmentation and attribute labeling as a joint task. Some of the limitations of early unsupervised methods were also addressed by ObjectRunner [Abdessalem et al., 2010] and AMBER [Furche et al., 2012]. These methods work by annotating webpages automatically with regular expressions, gazetteers and knowledge bases. They can rectify low quality annotations and even improve the annotators by exploring regular structures in the DOM during the record segmentation phase.

WDE methods have undoubtedly improved extraordinarily, but as pointed by Schulz et al. [2016], it is difficult to compare the results achieved by competing tools. Also, many tools seem to rely excessively on heuristic methods. The recent advancements in sequence taggers may provide more robust and flexible techniques to address this problem.

## 3.2 Named Entity Recognition

NER is an important subtask in IE. The NER task was defined in the Third Message Understanding Conference (MUC-3) [Sundheim, 1991], where researchers were asked to extract entities from a news corpus about terrorist incidents. However, it was the language-independent NER shared task at the Conference on Computational Natural Language Learning in 2003 (CoNLL-2003) [Tjong Kim Sang and De Meulder, 2003], that established an enduring labeled dataset constructed with news texts from Reuters. Despite its reduced size and the limited variability of its documents, the CoNLL-2003 dataset is still used to compare the performance of NER systems in English and German.

Sequence labeling for NER and Part-of-Speech (POS) tagging (labeling nouns, verbs, pronouns, etc.) is very similar, so, many times, research articles that propose new methods for NER also report results for the POS Tagging task. And frequently, the best methods for POS Tagging are also the best methods for NER.

Traditionally, the NER task was solved with generative models based on HMMs. The first appearance of HMMs in the field of NLP occurred in the mid-seventies and was primarily focused on the problem of Speech Recognition. But in the late nineties, HMMs also found important applications in IE and NER as in the works by Bikel et al. [1999], Freitag and McCallum [1999], and Freitag and McCallum [2000].

The problem with HMMs is that they model the joint probability between se-

quences of observations and labels  $P(X, Y)$ , a harder problem than modelling the corresponding conditional probability  $P(Y|X)$ . They assume conditional independence for the observations  $X$ , and therefore, they cannot handle overlapping features. This lack of flexibility led to the development of discriminative approaches to sequence labeling based on the Maximum Entropy principle.

Some early examples are Maximum Entropy Taggers for NER [Borthwick et al., 1998] and POS Tagging [Ratnaparkhi, 1998]. The Maximum Entropy Markov Model [McCallum et al., 2000] was developed just a while later, building up on the intuition of HMMs combined with the flexibility of discriminative approaches. However, because of the label bias problem, this model was superseded by CRFs [Lafferty et al., 2001; McCallum and Li, 2003]. At this time, the best performing systems almost always resorted to external gazetteers and hand-chosen features, as is the case for the CoNLL-2003 winning model [Florian et al., 2003].

With the advancement of Machine Learning in recent years, we saw the rise of Neural Networks applied to sequence labeling. In 2011, Collobert et al. [2011] introduced Neural Networks free of feature engineering to sequence labeling tasks, using Convolutional Neural Networks (CNN) over word embeddings with a CRF output layer to tackle the problems of POS tagging, Chunking, Semantic Role Labeling, and NER. A similar architecture proposed in 2015 by Huang et al. [2015], the Bi-LSTM-CRF, replaced the CNN in Collobert et al. [2011] with a bidirectional LSTM, achieving better results.

In 2016, further advancement to the NER models was achieved by incorporating character representations at the bottom of the Bi-LSTM-CRF architecture to extract morphological features from words automatically. The model by Lample et al. [2016] uses a bidirectional LSTM over character embeddings and the model by Ma and Hovy [2016] uses a one dimensional CNN with max pooling over character embeddings. Both models also made use of pre-trained GloVe word embeddings, created by Pennington et al. [2014].

Small improvements to NER systems have been made since then, primarily due to the introduction of new word embeddings on variations of the Bi-LSTM-CRF architecture. From this group, we can mention ELMo [Peters et al., 2018], BERT [Devlin et al., 2018], and Flair [Akshik et al., 2018]. All of them were introduced in 2018 and they differ primarily in how to construct contextual embeddings from the internal states of a language modelling neural network.

To our knowledge, the best NER system up to this date (considering the performance in the CoNLL-2003 dataset) is the bidirectional transformer model proposed by Baevski et al. [2019] that uses a stack of self-attention modules and ELMo embed-

dings. Table 3.1 shows a comparison of the history of reported F1 scores in the English test set of the CoNLL-2003 dataset for the NER systems that were mentioned in this section. Each of the mentioned systems held the record for the task at a given time.

Model	Test F1
Florian et al. [2003] (CoNLL-2003 Winner)	88.76
Collobert et al. [2011]	89.59
Huang et al. [2015]	90.10
Lample et al. [2016]	90.94
Ma and Hovy [2016]	91.21
Peters et al. [2018]	92.22
Devlin et al. [2018]	92.80
Akbik et al. [2018]	93.09
Baevski et al. [2019]	93.50

**Table 3.1.** Model performances in the CoNLL-2003 English test set.

### 3.3 Summary

In this chapter, we discussed the related work in WDE and NER. The models discussed in this dissertation and employed in the experiments section are methods for NER that were each deemed the best machine learning techniques for solving the problem at a given time. However, the RNE task is essentially a WDE task, so some ideas from this field are extremely useful, such as using HTML structural features, as we did in the Self-Training strategy for HMMs and the attention models for Neural Networks, that will be introduced in later chapters. Also, NER is a subtask that is mostly useful when we think of it in the context of a larger task, such as a specific setting of WDE. Because, NER approaches may be more or less effective depending on the objective extraction task. For example, we will see in the experiments section that, despite the significant improvements brought by ELMo embeddings to NER in plain text (as attested by the results reported in the CoNLL-2003 English dataset), their performance is far from great when we consider the RNE task, being surpassed by much simpler methods.

## Chapter 4

# Techniques for Sequence Labeling

Many tasks in NLP, such as POS Tagging and NER, can be solved by attributing labels to sequences of words. While in NER, we want to classify words into a set of predefined labels, any sequence labeling task in NLP can be modeled generically by considering that  $X = \{X_1, X_2, \dots, X_n\}$  is a sequence of random variables representing words from a finite vocabulary  $V = \{a, \textit{aardvark}, \dots, \textit{zebra}\}$ , and  $Y = \{Y_1, Y_2, \dots, Y_n\}$  is a sequence of random variables representing labels attributed to these words that take values from a finite set  $L = \{\text{B-PER}, \text{I-PER}, \dots, \text{O}\}$ , with both sequences generated by an unknown probability distribution. Considering that we observe the sequence of words  $x$  but we do not observe the sequence of labels  $y$ , then the goal for our classifier is to find:

$$y^* = \arg \max_y P(Y = y | X = x) \quad (4.1)$$

That is, the sequence of labels  $y^*$  that maximizes the conditional probability of  $y$  once we have observed the sequence of words  $x$ .

Estimating  $P(Y|X)$  naively from the relative frequencies in a labeled corpus is usually inadequate due to the exponential increase in the number of word and label combinations that needs to be considered as we increase the sequence size  $n$ . As  $n$  becomes large, label combinations will become increasingly uncommon in the dataset, what makes probability estimation less reliable. For example, if the vocabulary size consists of 1,000 words and there are three possible labels for each word, then there are already 27 billion combinations for sequences of only three words and labels. Also, sequences that were not seen during training will always have zero probability. That is problematic, because proper names are often unique or very infrequent in a corpus, so many word sequences that we care about will belong to this category.

Ultimately, the difference between statistical sequence labelling models lies in the

assumptions that we make about the distribution  $P(Y|X)$ . For practical reason, most statistical models of natural language make simplistic assumptions about the language structure. For example, a common assumption that is shared to some extent by all the models presented in this chapter is the Markov Assumption, that will be discussed thoroughly in Section 4.1. Stated simply, it assumes that by looking at a limited history in the sequence, for example the two previous words, we have sufficient information to make accurate predictions about the current word's label.

The related task in NLP called Language Modelling consists of predicting the next word in a sequence by looking at the tokens that precede it. Using the Markov Assumption, a language model that looks at the two previous words is called a bigram model and a model that looks at the three previous words is called a trigram model.

For anyone from a linguistics background, the idea that we would choose to use a model of language structure which predicts the next word simply by examining the previous two words - with no reference to the structure of the sentence - seems almost preposterous. But, actually, the lexical co-occurrence, semantic, and basic syntactic relationships that appear in this very local context are a good predictor of the next word, and such systems work surprisingly well. Indeed, it is difficult to beat a trigram model on the purely linear task of predicting the next word [Manning and Schütze, 1999].

Considering this, we do not usually evaluate the quality of statistical natural language models by how reasonable their assumptions are, but by how instrumentally effective the models are for solving objective tasks. The epistemological implications of this view are important, but in this work we will consider it to be valid.

In this chapter, we discuss several machine learning methods for NER. In Section 4.1, we discuss Hidden Markov Models, in Section 4.2 we discuss Conditional Random Fields, and in Section 4.3 we discuss Neural Networks.

## 4.1 Hidden Markov Models

In this section, we derive a supervised HMM approach for NER. The HMM is an expanded Markov Chain where the sequence of observed states depends on an underlying sequence of hidden states. In the context of NER, we can think that the states of a Markov Chain correspond to the set of possible labels or classes (e.g. PER, ORG, LOC, etc.). First, we consider a standalone Markov Chain and then we proceed to discuss HMMs.

A Markov Chain is a stochastic model for describing sequences when the described sequences have the Markov Property. That is, consider a sequence of random variables  $Y = \{Y_1, Y_2, \dots, Y_n\}$  taking values from a restricted set of possible states  $L = \{L_1, L_2, \dots, L_m\}$ . Sequence  $Y$  only satisfies the Markov Property if:

$$P(Y_i = L_k | Y_{i-1}) = P(Y_i = L_k | Y_{i-1}, Y_{i-2}, \dots, Y_1) \quad (4.2)$$

The probability of observing state  $Y_i = L_k$  depends only on the previous state. What the Markov Assumption implies is that, at any position in the sequence, the entire history of observed states is encoded in the previous state. So, by knowing the previous state, we can make as accurate predictions about the future as we would make by knowing all the history. Conditioned on this previous state, the future and past states are independent.

When we consider two or three previous states instead of a single previous state, the Markov Chain is said to be of second order and third order, respectively. Any Markov Chain of higher order can be converted into an equivalent first order Markov Chain. For example, a Markov Chain of second order with two states  $\{A, B\}$  can be transformed into an equivalent Markov Chain of first order with four states  $\{AA, AB, BA, BB\}$ . Therefore, from now on, we will proceed with the analysis of first order Markov Chains without loss of generality regarding higher order Markov Chains.

A Markov Chain can represent a wide range of phenomena such as daily temperatures in a region, closing prices of a stock in the financial market, yearly demographic growth, or words in a text. However, in all these cases, the model is a simplification of reality that makes sequences mathematically treatable. The extent to which this simplification will hurt our predictions depends on the nature of the studied phenomena.

To see why assuming the Markov property may introduce a problem on text related tasks, consider the following labeled sentence:

"[Jon Snow]<sub>PER</sub> is [Ned]<sub>PER</sub> 's son. Snow is piling up over [Winterfell]<sub>LOC</sub>,  
but [Snow]<sub>PER</sub> is not bothered."

Snow is a very uncommon name, so a statistical model may have difficulty labeling the last mention of Snow correctly, but it would certainly be helpful if it could refer to the first mention in the beginning of the sentence, where the named entity label is more obvious. To do this we would need to keep track of distant relationships in the text that would not be captured by a Markov Model of second or third order. While we could consider employing a Markov Model of higher order to overcome this limitation, considering more than three or four states at any time makes parameter estimation

unreliable because each combination of states becomes increasingly uncommon in the training set. Also, when the considered number of states increases, the cost for predicting the optimal sequence of states quickly becomes prohibitively high. There are some Markov Chain variations that try to address this problem, as well as the other models presented later in this chapter. However, HMMs up to third-order can work well if we add independent features to make up for the lack of a longer context.

Furthermore, besides the Markov Assumption, we make the Time Invariance Assumption:

$$P(Y_{i+1} = L_b | Y_i = L_a) = P(Y_i = L_b | Y_{i-1} = L_a) \quad (4.3)$$

That is, given the observed states  $L_a$  and  $L_b$ , the probability of going from  $L_a$  to  $L_b$  in position  $i$  is the same as in any other position in the sequence. We refer to "time" invariance eventhough the correct name for this assumption in the context of sequence labeling would be positional invariance, because this is the usual name adopted in the literature. Everytime we refer to time or timesteps in the text, we mean the position of a token or label in the sequence.

With this, we can calculate the probability of observing sequence  $Y = \{Y_1, Y_2, \dots, Y_n\}$  as:

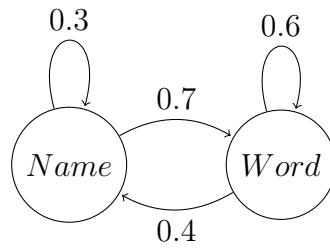
$$P(Y) = \prod_{i=1}^n P(Y_i | Y_{i-1}) \quad (4.4)$$

With the Time Invariance Assumption, all transition probabilities in our model can be described by a  $|L| \times |L|$  transition matrix  $\theta$  where each element  $\theta_{a,b}$  with row  $a$  and column  $b$  holds the probability of going from state  $L_a$  to  $L_b$  and  $|L|$  is the number of possible labels. Naturally, the probabilities of row  $\theta_{a,*}$  must sum up to one since they represent the entire scope of transition possibilities starting from state  $L_a$ . For example, consider the finite state-machine in Figure 4.1 that describes a Time Invariant Markov Chain with only two states (Name, Word), that models a sequence of words that can be either names or common words. The edges in this graph represent the transition probabilities between states and the transition matrix for this Markov Chain takes the form:

$$\theta = \begin{bmatrix} 0.3 & 0.7 \\ 0.4 & 0.6 \end{bmatrix}$$

If we know the transition matrix  $\theta$  for a Markov Chain, we can easily calculate the probabilities of being in each state at time  $t$ . If  $\rho_t$  is a vector of size  $m$  with the probabilities for each state at time  $t$ . Then:

$$\rho_t = \rho_0 \cdot \theta^t \quad (4.5)$$



**Figure 4.1.** Finite state machine for a Markov Chain.

Where  $\rho_0$  is the vector of starting probabilities for each state. We assume that we have no knowledge about the initial probabilities of our Markov Chain, so we simply attribute the same probability to every state at time  $t = 0$ . For the Markov Chain described in Figure 4.1, this assumption would make  $\rho_0 = (0.5, 0.5)$ . For convenience, consider that  $P(Y_1|Y_0)$  is the probability of  $Y_1$  given the initial vector of uniform state probabilities  $\rho_0$ .

Now we know how to calculate the state probabilities for a Markov Chain at any position in the sequence provided that we know the transition matrix. In the context of sequence labeling, we want to obtain the transition matrix  $\theta$  from our training data. The most usual choice is to obtain  $\theta$  such that it maximizes the probability of reproducing our labeled dataset with Maximum Likelihood Estimation. More formally, the likelihood of our model is given by:

$$\mathcal{L}(\theta) = \prod_{i=1}^n \theta_{y_i, y_{i-1}} \quad (4.6)$$

Where  $y_i$  is the correct label at position  $i$ . By defining  $\eta_{a,b}$  to be the count of transitions from state  $a$  to  $b$  in the dataset, where  $a$  and  $b$  are states from a finite set of states  $L = \{L_1, L_2, \dots, L_k\}$ . Instead, we can write the likelihood as:

$$\mathcal{L}(\theta) = \prod_{a,b \in L} \theta_{a,b}^{\eta_{a,b}} \quad (4.7)$$

where the product is calculated over all possible transitions from  $a$  to  $b$ . The logarithm function is monotone increasing, so we can maximize the log-likelihood instead of the likelihood for ease of calculation. With the log-likelihood, the product over transition probabilities becomes a sum of logarithms. The log-likelihood is defined by:

$$\ell(\theta) = \sum_{a,b \in L} \eta_{a,b} \cdot \log(\theta_{a,b}) \quad (4.8)$$



Then, the maximum likelihood estimator becomes:

$$\hat{\theta} = \arg \max_{\theta} \ell(\theta) \quad (4.9)$$

However, if we try to optimize this function as it is, we will find that  $\hat{\theta}_{a,b} = \infty$ . This happens because we did not take into account the degrees of freedom of our model. Every row in the transition matrix must sum up to one, so the parameters for the last row are defined by the preceding rows. The degrees of freedom for the model are actually  $|L|(|L| - 1)$  and not  $|L|^2$  as we were inadvertently considering before. That is, we have a set of constraints:

$$\sum_{b \in L} \theta_{a,b} = 1, \quad \forall a \in L \quad (4.10)$$

With the aid of Lagrange Multipliers, we can incorporate this set of constraints into our optimization objective and optimize the Lagrangian:

$$\Lambda(\theta, \lambda) = \ell(\theta) - \sum_{a \in L} \lambda_a \cdot \left( \sum_{b \in L} \theta_{a,b} - 1 \right) \quad (4.11)$$

Setting  $\frac{\partial \Lambda(\theta, \lambda)}{\partial \lambda_a} = 0$  simply yields the initial set of constraints, but by setting the partial derivatives  $\frac{\partial \Lambda(\theta, \lambda)}{\partial \theta_{a,b}} = 0$ , we get:

$$\frac{\partial \Lambda(\theta, \lambda)}{\partial \theta_{a,b}} = \frac{\eta_{a,b}}{\theta_{a,b}} - \lambda_a = 0 \quad (4.12)$$

$$\theta_{a,b} = \frac{\eta_{a,b}}{\lambda_a} \quad (4.13)$$

Finally, using the initial constraint equations, we find that:

$$\sum_{b \in L} \frac{\eta_{a,b}}{\lambda_a} = 1 \quad (4.14)$$

$$\lambda_a = \sum_{b \in L} \eta_{a,b} \quad (4.15)$$

$$\hat{\theta}_{a,b} = \frac{\eta_{a,b}}{\sum_{b' \in L} \eta_{a,b'}} \quad (4.16)$$

yielding a closed form expression for the maximum likelihood transition matrix. What is really convenient, since this is the average number of transitions from state  $a$  to state  $b$  in the dataset. A value that can be easily calculated.

So far, we have assumed that all states/labels are observable, but in NER, only

the words are observed, while the named entity labels associated with these words are not. That is why we need another layer of complexity. The HMM differs from the Markov Chain in that it does not observe the states  $Y$  directly, but rather a probabilistic function of these states. In Markov Chains, only the labels were being considered without mention of the accompanying words. With the HMM, we want to estimate the probabilities of a sequence of labels  $Y = \{Y_1, Y_2, \dots, Y_n\}$  (i.e. the Markov Chain) from a sequence of observed states  $X = \{X_1, X_2, \dots, X_n\}$  (i.e. the words in a text, or a feature vectors with independent features). Now, to calculate the conditional probability for a sequence of labels we employ Bayes Theorem:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \quad (4.17)$$

Also, since  $P(X)$  is invariant for every sequence of labels  $Y$ , to find the most likely sequence of labels, we can simply optimize in terms of the joint probability:

$$P(Y|X) \propto P(X|Y)P(Y) = P(X, Y) \quad (4.18)$$

The procedure to obtain the  $P(Y)$  probabilities has already been described for the standalone Markov Chains and it is the same for HMMs. But, to obtain the  $P(X|Y)$  probabilities, we must make another assumption. That is:

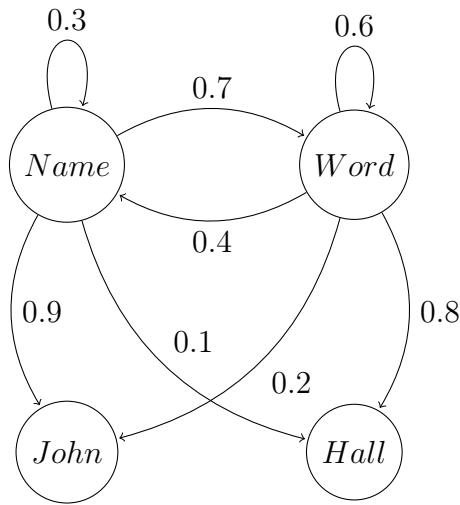
$$P(X_i = x_i | Y_i = y_i, Y_{i-1} = y_{i-1}, X_{i-1} = x_{i-1}, \dots) = P(X_i = x_i | Y_i = y_i) \quad (4.19)$$

the probability of observing word  $x_i$  depends only on the current label  $y_i$ , the hidden state (e.g. B-PER, I-PER, O, etc.). With this assumption, we are stating that once we know a token's assigned label, then we can reliably predict what is the probability that this token takes any value from a limited vocabulary. Conditioned on the assigned label, the current word is independent of the other words in the sequence. This assumption is problematic when we want to use observations from other positions in the sequence to predict labels at the current position, because these observations are likely to be correlated. For this reason, other models such as CRFs (explored in the next section) try to overcome this issue by taking different assumptions.

Now we can finally calculate the probability of a sequence  $Y$  given a sequence of observations  $X$ :

$$P(Y|X) \propto \prod_{i=1}^n P(X_i|Y_i)P(Y_i|Y_{i-1}) \quad (4.20)$$

This is the main equation for the HMM. To model the emission probability distributions



**Figure 4.2.** Finite state machine for a Hidden Markov Model.

$P(X_i|Y_i)$ , we assume that the probabilities are time invariant and that  $X_i$  takes its value from a fixed vocabulary with size  $|V|$ . Similar to the transition matrix  $\theta$ , we introduce a  $|V| \times |L|$  emission matrix  $\mu$ , where  $|L|$  is the number of possible labels and each cell  $\mu_{a,b}$  represents the probability that, given label  $a$ , we will observe word  $b$ . For example, consider the finite state-machine in Figure 4.2 that extends our previous example and defines a HMM with two states (Name, Word), and a vocabulary of two words (John, Hall). The edges between the hidden states and the words represent the emission probabilities, and the emission matrix for this HMM takes the form:

$$\mu = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}$$

Now the log-likelihood for the HMM is defined as:

$$\ell(\theta, \mu) = \sum_{a \in L, b \in L} \eta_{a,b} \cdot \log(\theta_{a,b}) + \sum_{c \in L, d \in V} \eta'_{c,d} \cdot \log(\mu_{c,d}) \quad (4.21)$$

where  $\eta'_{c,d}$  is a function that counts the number of emissions from state  $c$  to word  $d$ . The matrices  $\theta$  and  $\mu$  are independent, so we can optimize for the transition and emission matrices separately, therefore the procedure to find  $\hat{\theta}$  is still the one described in Equation 4.16. Also, notice that if we do the same procedure with Lagrangian Multipliers that we employed to find the MLE for Markov Chains, but this time we take the partial derivatives relative to  $\mu_{c,d}$  and set the constraints such that each line

in the emission matrix must sum up to one, we will find that:

$$\hat{\mu}_{c,d} = \frac{\eta'_{c,d}}{\sum_{d' \in V} \eta'_{c,d'}} \quad (4.22)$$

This is the expected number of times that we observed word  $d$  when we were at the hidden state  $c$  in the training set. A closed form expression that is easy to calculate from the training data. With that, we conclude the maximum likelihood parameter estimation for the HMM. This procedure is only usable when we have access to labeled data, since we need to observe the correct labels to estimate  $\hat{\theta}$  and  $\hat{\mu}$ . If we only have unlabeled data or partially labeled data, we can do the parameter estimation with the Baum-Welch Algorithm by Baum et al. [1970], though the results for NER become much less reliable.

### 4.1.1 Smoothing

The problem with the Maximum Likelihood Estimator derived above is that the emission probability for words that were not observed in the training set will always be zero. Considering that we only have a small dataset and even if it were much bigger we would still observe new words when running predictions for unseen data, this problem needs to be addressed. We use Laplace smoothing [Manning and Schütze, 1999] to change the estimates for  $\theta$  and  $\mu$  attributing some probability mass to unseen tokens:

$$\hat{\theta}_{a,b} = \frac{\eta_{a,b} + 1}{\sum_{b' \in L} \eta_{a,b'} + |L|} \quad (4.23)$$

$$\hat{\mu}_{c,d} = \frac{\eta'_{c,d} + 1}{\sum_{d' \in V} \eta'_{c,d'} + |V|} \quad (4.24)$$

where  $|L|$  is the number of labels and  $|V|$  is the size of the vocabulary. If we consider other smoothing methods we will get different estimators, but this already properly ensures that unobserved words will not receive a zero probability. The main shortcoming with Laplace Smoothing is that it gives too much probability mass to unseen words relative to other smoothing methods.

### 4.1.2 Predicting sequences

The last task that needs to be done is to calculate the most likely sequence of labels given a sequence of observations. With estimates for transition and emission matrices we can easily calculate the conditional probability  $P(Y = y|X = x)$  of each label se-

quence  $y$ , given the observed sequence  $x$ . But, instead of computing probabilities for each possible label configuration and taking the one with maximum probability, we can obtain the label sequence with maximum probability with the Viterbi algorithm by Forney [1973], which is a dynamic programming algorithm that calculates the probabilities for sequences with the Markov Property exactly and efficiently. As we increase the order of our HMM, this computation becomes exponentially more expensive, but we found experimentally that for HMMs up to third order, the Viterbi algorithm is still viable. For larger windows, a beam-search strategy can be used.

### 4.1.3 Self-Training

The self-training strategy is a semi-supervised machine learning technique to improve the performance of HMMs in web sequence labeling tasks. A HMM for the sequence labeling task has the form:

$$P(y|x) \propto \prod_{i=1}^n P(x_i|y_i)P(y_i|y_{i-1}) \quad (4.25)$$

where  $x$  is a sequence of words and  $y$  is a sequence of labels, both with size  $n$ , and the initial probabilities  $P(y_1|y_0)$  follow a uniform distribution across all label assignments. Also, to simplify the notation, we omit the random variables, that is,  $P(Y = y|X = x) \equiv P(y|x)$ . To construct this model from the data, we need to estimate the following probabilities:

- $P(x_i|y_i)$ : the emission probability of word  $x_i$  given label  $y_i$ .
- $P(y_i|y_{i-1})$ : the transition probability of going from label  $y_{i-1}$  to label  $y_i$ .

We may consider  $x_i$  to be a vector of binary features, i.e.  $x_i = \{f_{i,1}, f_{i,2}, \dots, f_{i,k}\}$  as long as all the feature distributions are independent, conditioned on  $y_i$ . That is:

$$P(x_i|y_i) = P(f_{i,1}, f_{i,2}, \dots, f_{i,k}|y_i) = \prod_{j=1}^k P(f_{i,j}|y_i) \quad (4.26)$$

Also, due to the time invariance assumption, the  $P(f_{i,j}|y_i)$  probabilities are independent of the timestep  $i$ . Therefore, for each binary feature  $f_j$  we need only estimate the parameters  $\hat{P}(f_j|y)$ , relative to each possible assignment of  $y$ . The maximum likelihood estimators for these feature parameters can be obtained from their relative frequencies, just as we did in Section 4.1 for single words (Equation 4.22). For example, consider a binary feature  $F_C$  that indicates if a word is capitalized and label  $Y$  which takes a

value from the set  $\{PER, O\}$  indicating if the current word is a person name (PER) or something else (O), then we can get the feature estimator with the expression:

$$\hat{P}(F_C = 1|Y = \text{PER}) = \frac{\text{Count}(f_C = 1, y = \text{PER})}{\text{Count}(f_C = 0, y = \text{PER}) + \text{Count}(f_C = 1, y = \text{PER})} \quad (4.27)$$

where  $\text{Count}(f_C = A, y = B)$  is a function that counts the number of times feature  $f_C$  took value  $A$  at the same time that label  $y$  took value  $B$  in the training set.

This approach yields good experimental results for textual features such as the capitalization of a word, but if we try to use HTML structural features such as the HTML tags, we will find that they are not very useful, contradicting our intuition. The problem is that there is not much similarity between the HTML structure in different webpages. For example, only because a named entity label occurs more often inside `<div>` tags in a page, this does not mean that will be the case for most webpages. Consider for example a faculty webpage that shows researcher names in a table (`<td>` tags) in contrast to a webpage that organizes researcher names in a list (`<li>` tags). If there are many cases like the former page in our training set, and we use the HTML tag feature in a model that assigns labels to the latter page in our test set, we will probably get many wrong predictions. Nonetheless, the HTML features are not useless. Inside a single webpage, the HTML tag is a good predictor of the correct labels. Words with a similar HTML context tend to have similar labels. For example, words that happen together in a list have very similar HTML contexts and are very likely to belong to the same category. The question is how to obtain better parameter estimators for HTML features.

Consider a set  $T$  of  $m$  textual features unrelated to the HTML structure:

$$T = \{f_1^T, f_2^T, \dots, f_m^T\}$$

And a set  $H$  of  $k$  HTML features that are related to the HTML structure:

$$H = \{f_1^H, f_2^H, \dots, f_k^H\}$$

Given a HMM trained with only the first set of features  $T$ , we can use this model to predict the labels for an unlabeled document. Consider an HTML binary feature  $F_{td}$  representing if a token's parent tag is `<td>`, then the estimator for this HTML feature probability becomes:

$$\hat{P}(F_{td} = 1|Y = \text{PER}) = \frac{\text{Count}(f_{td} = 1, \tilde{y} = \text{PER})}{\text{Count}(f_{td} = 0, \tilde{y} = \text{PER}) + \text{Count}(f_{td} = 1, \tilde{y} = \text{PER})} \quad (4.28)$$

Where  $\tilde{y}$  is the label predicted by the HMM trained with only textual features and  $f_{td}$  is the feature value in the training set. We use this formulation to calculate feature probabilities for only the HTML features. Next, we incorporate the estimators for the HTML feature probabilities in the original HMM and predict the final labels with the whole set of features  $T \cup H$ . This process could be repeated multiple times. If the predictions are becoming more accurate with each iteration, then the estimates are also likely to improve.

In a simpler explanation, the self-training strategy can be divided into the following steps:

- Train the HMM without any HTML features.
- Compute labels for a website with the trained HMM.
- Use the computed labels as a proxy for the actual labels in the website and estimate HTML feature frequencies for this website alone.
- Recompute the labels now using the HTML feature probabilities.

To understand why this strategy could help labeling named entities on the Web, consider the example shown in Figure 4.3. This example shows a snippet from a webpage and how a naive HMM would probably attribute labels to each word. Despite being rather uncommon, North West is an actual person name, what becomes obvious in this case once we know the HTML context where it happens (inside a list with other person names), but this is not so obvious if we consider the sentence in isolation. The naive HMM could be improved if we calculated feature probabilities for the parent tags using proxies for the correct labels. We would find that four times in a total of six occurrences, the correct label for a word that happened inside a  $\langle li \rangle$  tag was *I-PER*. By incorporating this knowledge in the model using the self-training strategy, we would improve the probability that words North and West were labeled correctly as a person name in this case.

The self-training strategy must be employed individually over each webpage. This means that, for each webpage, we discard the previously calculated HTML feature probabilities and calculate new ones. The reason for this is that the webpage layouts vary significantly between different sources, so general feature probabilities are likely to introduce too much noise in the model. In the above mentioned example, we cannot assume that only because tokens inside a  $\langle li \rangle$  tag are more likely to be names in the context of that particular webpage, this will also be the case for other webpages. HTML feature probabilities cannot be generalized because of the great variation in

```
<div>Three (0) people (0) : (0)</div>
<ul>
  <li>John (I-PER) Smith (I-PER)</li>
  <li>Kate (I-PER) Clark (I-PER)</li>
  <li>North (0) West (0)</li>
</ul>
```

**Figure 4.3.** Example of a webpage snippet with labels attributed by a HMM.

webpage layout. For example, another webpage may be formatted in such a way that named entities never occur inside a list, and all the tokens inside tables are named entities. Therefore, HTML feature probabilities calculated outside the scope of the webpage are likely to lead to bad predictions.

The self-training strategy could possibly be incorporated in the Baum-Welch algorithm to provide a more statistically solid argument for its effectiveness, but this heuristic already yields a consistent improvement to sequence labeling on the Web. In theory, this strategy could be used with any sequence tagger, however retraining a classifier with new features can become prohibitively expensive in the case of CRFs or Neural Networks. In HMMs, retraining the features is fast, because we assume feature independence and the maximum likelihood estimator can be obtained with a simple closed form expression.

#### 4.1.4 Experimental Considerations

HMM based taggers have been successfully applied in many NLP and WDE tasks such as in the works by Rabiner [1990], and Freitag and McCallum [2000]. The closed-form parameter estimation makes them incredibly fast to train, also their parameters are very interpretable (because the model is simple), making them a good choice for a first approximation to NER. However, as we will discuss in the experiments section, these models are highly dependent on the right selection of features, what may outweigh the benefit of a small training cost.

## 4.2 Conditional Random Fields

With HMMs, we tried to model the joint probability between words and labels  $P(X, Y)$  and derive the conditional probability  $P(Y|X)$  with Bayes Theorem. However, this is a waste of modelling effort, since we need to assume the shape of the prior distribution  $P(Y)$ , but we only really care about the conditional problem. Thus, modelling  $P(Y|X)$  instead of the joint distribution would presumably be a more direct and simple



approach. Also, to assure that the computation of  $P(X, Y)$  was feasible, we needed to assume feature independence, what inhibits the use of overlapping features that could potentially be useful in addition to words. Maximum Entropy models for sequence labeling were created to address these issues and, among them, Linear Chain CRFs [Lafferty et al., 2001] stand out as the discriminative analogue to HMMs. CRFs estimate the conditional probability while HMMs, being generative models, estimate the joint probability. In this section, we derive the Maximum Entropy Classifier and the Linear Chain CRF from basic principles.

The concept of Information Entropy established by Shannon [1948] quantifies the amount of information expressed in a statistical distribution. The entropy  $H$  of a random variable  $Z$  with a probability mass function  $P$  is given by:

$$H(Z) = \sum_{z \in Z} P(z) \cdot \log \left( \frac{1}{P(z)} \right) \quad (4.29)$$

The logarithm in the entropy function can be taken in any base. The change of base will only provide a linear scaling of the same entropy function. When using base two,  $H(Z)$  is essentially the expected number of bits necessary to encode the value of a random outcome drawn from  $Z$ , that is  $E \left[ \log \left( \frac{1}{P(Z)} \right) \right]$ . The more bits we need to use to encode the possible outcomes of a probability mass function, the more uncertain we are about the underlying distribution. For example, if an event has only one possible outcome, we always know the value of a random sample, therefore  $H(Z) = 0$  and the entropy is minimum. On the other end, if the probability distribution is uniform, all outcomes have the same probability, and therefore we are as uncertain as we can be and the entropy is maximum.

The principle of maximum entropy set forward by Jaynes [1957] states that, when we do not know exactly what probability distribution generated a sample, the best estimate for the parameters of this probability distribution is the one that makes the least assumptions, or the one with maximum entropy. This is the distribution that is closest to the uniform distribution. A similar argument can be made to limit the number of free parameters in the model. That is, when comparing two models with similar predictive power, the one with the least degrees of freedom should be preferred.

The principle of maximum entropy is similar to a principle in the philosophy of science called Occam's Razor<sup>1</sup>, that states that, when there are two explanations for an outcome, the one that makes the least number of assumptions is the best. Which is a

---

<sup>1</sup> This idea is attributed to the English scholar William of Ockham that lived in the thirteenth century.

good rule of thumb for science in general, even though, ultimately, nothing guarantees that simpler models will be better.

The task of labeling word sequences can be thought of as a multinomial classification task where each label  $Y$  is predicted according to its context  $X$ . The random variable  $Y$  takes values from a finite set of labels  $L = \{L_1, L_2, \dots, L_k\}$  and the random variable  $X$  takes a value from a finite set of possible contexts  $C$ . These contexts are defined by a vector of observations expressing features (useful pieces of evidence) such as: the last word is an honorific, the current word is capitalized, the current word is "that", etc. Also, we define  $k$  binary feature functions  $f_j \forall j \in [1, k]$  such that:

$$f_j(x, y) = \begin{cases} 1 & \text{if } y = L_a \text{ and the feature is present in } x \\ 0 & \text{otherwise} \end{cases} \quad (4.30)$$

Where the feature function is only true when the observed label is  $L_a$  and the feature occurs in context  $x$ . So, at any position in the sequence, the truth statement of the feature functions can be determined in terms of the predicted label  $y$  and the context  $x$ . At this moment, we treat the task of attributing labels to tokens at each position independently, so we consider that the context  $x_i$  and label  $y_i$  for a token at position  $i$  can simply be represented generically by  $x$  and  $y$ .

Now, the Maximum Entropy framework provides a compelling way for estimating the probability of a label given its linguistic context, that is  $P(Y|X)$ . In the sequence labeling problem, we actually want to maximize the conditional entropy:

$$H(Y|X) = - \sum_{x \in C, y \in L} P(X = x, Y = y) \cdot \log(P(Y = y|X = x)) \quad (4.31)$$

It is usual to approximate  $P(X = x, Y = y)$  with  $P(Y = y|X = x)\tilde{P}(X = x)$  [Ratnaparkhi, 1998], because the model probability  $P(X = x)$  is hard to obtain since its probability space is the entire range of context configurations, which is usually very big. However, the observed probability  $\tilde{P}(X = x)$  is defined over a fixed training sample, so we only need to calculate probabilities for context configurations that actually occur in this sample.

If we maximize  $H(Y|X)$  subject to no restrictions, we will find that the Maximum Entropy estimator will yield an uniform distribution, meaning that we are no better than simply predicting labels at random. So, it becomes necessary to establish some constraints for the optimization problem.

The choice is more or less arbitrary, but under the Maximum Entropy frame-

work, there is a strong motivation for setting the constraints in a way that makes the Maximum Entropy estimator equivalent to the Maximum Likelihood estimator for our model. By doing it, we can make sure that the model fits the data as tightly as possible (Maximum Likelihood), while making as few assumptions as it needs to (Maximum Entropy). We can get this estimator by setting the constraints such that, given the  $k$  binary feature functions that we determined earlier, there are also  $k$  constraints such that:

$$E_P[f_i] = E_{\tilde{P}}[f_i] \quad \forall i \in [1, k] \quad (4.32)$$

where  $E_P[f_i]$  is the model expectation of  $f_i$  and  $E_{\tilde{P}}[f_i]$  is the empirical expectation of  $f_i$ . Or more clearly:

$$\begin{aligned} E_P[f_i] &= \sum_{x \in C, y \in L} \tilde{P}(x) P(y|x) f_i(x, y) \\ E_{\tilde{P}}[f_i] &= \sum_{x \in C, y \in L} \tilde{P}(x, y) f_i(x, y) \end{aligned}$$

From this point forward, assume that, when the random variable is omitted,  $P(x) \equiv P(X = x)$ . Additionally, we need to make sure that the probabilities sum up to one, so:

$$\sum_{y \in L} P(y|x) = 1 \quad \forall x \in C \quad (4.33)$$

We can optimize the conditional entropy subject to these constraints with Lagrangian Multipliers by defining the Lagrangian:

$$\Lambda(P, \lambda, \beta) = H(Y|X) + \sum_{i=1}^k \lambda_i (E_P[f_i] - E_{\tilde{P}}[f_i]) + \sum_{x \in C} \beta_x \left( \sum_{y \in L} P(y|x) - 1 \right) \quad (4.34)$$

and solving the unconstrained optimization problem to obtain the maximum entropy distribution:

$$P^* = \arg \max_P \Lambda(P, \lambda, \beta) \quad (4.35)$$

The derivatives relative to  $\lambda_i$  and  $\beta_x$  yield the initial constraints, but setting the derivative relative to  $P(y|x)$  to zero, we get:

$$\frac{\partial \Lambda}{\partial P(y|x)} = -\tilde{P}(X) - \tilde{P}(X) \cdot \log P(y|x) + \beta_x + \tilde{P}(X) \sum_{i=1}^k \lambda_i f_i(x, y) = 0 \quad (4.36)$$

$$P(y|x) = \exp\left(\frac{\beta_x}{\tilde{P}(x)} - 1\right) \cdot \exp\left(\sum_{i=1}^n \lambda_i f_i(x, y)\right) \quad (4.37)$$

By replacing  $P(y|x)$  in our initial probability constraints defined in Equation 4.33, we get:

$$\sum_y \exp\left(\frac{\beta_x}{\tilde{P}(x)} - 1\right) \cdot \exp\left(\sum_{i=1}^n \lambda_i f_i(x, y)\right) = 1 \quad (4.38)$$

$$\exp\left(\frac{\beta_x}{\tilde{P}(x)} - 1\right) = \frac{1}{\sum_y \exp(\sum_{i=1}^n \lambda_i f_i(x, y))} \quad (4.39)$$

By replacing the Equation 4.39 in Equation 4.37 we finally get:

$$P(y|x) = \frac{1}{Z(x)} \cdot \exp\left(\sum_{i=1}^n \lambda_i f_i(x, y)\right) \quad (4.40)$$

$$Z(x) \equiv \sum_y \exp\left(\sum_{i=1}^n \lambda_i f_i(x, y)\right) \quad (4.41)$$

This defines a family of exponential functions with parameters  $\lambda$ . To obtain the parameters that maximize the entropy of our model subject to the initial set of constraints, we can replace this definition back in the Lagrangian  $\Lambda$  and optimize in terms of  $\lambda$ . This yields the following optimization problem:

$$\Lambda(\lambda) = \sum_{i=1}^k \lambda_i E[f_i(x, y)] - \sum_x \tilde{P}(x) \cdot \log Z(x) \quad (4.42)$$

which is identical to the log-likelihood:

$$\ell(P) = \sum_{x,y} \tilde{P}(x, y) \cdot \log P(y|x) \quad (4.43)$$

With this, we conclude that a model with the parametric form defined in Equation 4.40 has identical maximum likelihood and maximum entropy estimators. Also, it can be proven that this model is unique and its optimization function is concave [Brown, 1986]. The classifier based on this model is known as a Logistic Classifier or Maximum Entropy Classifier. There is no closed-form expression for finding the optimal set of parameters, but we can obtain this optimal set with numerical optimization methods such as Stochastic Gradient Descent. Normally, this is achieved by minimizing the cross-entropy for the exponential model, which is identical to maximizing the likelihood.

For the particular case of sequence labeling, the simplest conceivable classifier inside the Maximum Entropy Framework would be a multinomial logistic regression that decodes each label independently. That is, given a context  $x$ , the classifier would predict a label  $y$  without taking in consideration any of the previous or forward labels. This would be the discriminative analogue to the Naive Bayes model. However, the independence assumption is too restrictive for some language related tasks, as we attested in the experiments for this dissertation.

Linear Chain CRFs<sup>2</sup> relax this assumption by jointly decoding the entire sequence of labels. That is, they normalize the probabilities over the entire sequence of labels and the feature functions also take the previous label in consideration. The CRF equation is very similar to the generic maximum entropy equation derived earlier. The only difference being the assumptions about the feature functions and the joint decoding of label sequences. For a sequence of contexts  $x = \{x_1, x_2, \dots, x_n\}$  and labels  $y = \{y_1, y_2, \dots, y_n\}$  with size  $n$ , and  $k$  feature functions, the Linear Chain CRF is defined by:

$$P(Y = y|X = x) = \frac{1}{Z(x)} \prod_{i=1}^n \exp \left( \sum_{j=1}^k \theta_j f_j(y_{i-1}, y_i, x_i) \right) \quad (4.44)$$

$$Z(x) = \sum_{y'} \prod_{i=1}^n \exp \left( \sum_{j=1}^k \theta_j f_j(y_{i-1}, y_i, x_i) \right) \quad (4.45)$$

Where  $y'$  is the set of all possible label combinations with size  $n$ . This is a huge combinatorial space with  $|L|^n$  combinations, where  $|L|$  is the number of possible labels. But this quantity can be calculated efficiently with the sum-product algorithm, because of the assumptions regarding the feature functions. The most likely label sequence can be calculated with the Viterbi algorithm, as was the case for HMMs.

Our analysis only discussed Linear Chain CRFs. For a thorough analysis of Conditional Random Fields that is not restricted to Linear Chains, consult Sutton and McCallum [2012].

## 4.2.1 Experimental Considerations

CRFs are more general than HMMs, because the transitions from  $y_{i-1}$  to  $y_i$  can depend on the context  $x_i$  and the feature functions may be correlated. This flexibility of feature

---

<sup>2</sup> When we talk about CRFs in other contexts, unless stated otherwise, we will be talking about Linear Chains CRFs.

functions, and the fact that CRFs model the conditional probability instead of the joint probability allows for a wider range of possibilities. In general, CRFs have performed better than HMMs in sequence labeling problems, but recently, pure CRF models have been largely replaced by neural networks. However, CRFs are still employed as the output layer of complex neural architectures and the maximum entropy framework is of great relevance to Neural Networks.

### 4.3 Neural Networks

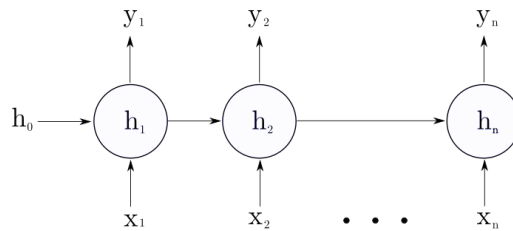
The recent upsurge in the popularity of neural networks owes to the increasing computational capacity brought by Graphical Processing Units and discoveries such as the fast learning algorithm for Deep Belief Networks [Hinton et al., 2006], Convolutional Neural Networks [LeCun et al., 1989], Long Short-Term Memory [Hochreiter and Schmidhuber, 1997], and Word Embeddings [Bengio et al., 2003] which helped overcome the limitations of earlier neural architectures.

Neural networks are a family of classification algorithms that were vaguely inspired in the functioning of the human brain. They consist of a weighted graph of artificial neurons, which are functions that receive an input from other neurons or from an input vector and produce an output according to their activations. Neural networks are a general framework that can encompass other machine learning approaches. For example, as proven by Cox [1958], a single layer neural network with a Softmax activation function (a Perceptron) is equivalent to the multinomial logistic classifier, that we already explored in the previous section. For a thorough view of the topic of neural networks, consult Bengio [2009].

An extension of the logistic classifier to a neural network with multiple layers can still be trained by minimizing the cross entropy function. However, once we add multiple layers on top of the logistic classifier, the optimization problem no longer retains its convexity. This means that numerical optimization methods can get trapped on local minima and miss the best set of parameters. For a long time, this limitation hindered the development of neural architectures. Nonetheless, in many practical cases, the rugged shape of the cost function does not prevent the optimization algorithm from finding a good set of parameters, as long as we make use of clever optimization strategies to avoid local minima, such as the strategy of adding momentum to gradients [Sutskever et al., 2013].

In contrast to earlier probabilistic models that were more statistically oriented, many improvements to neural networks derive from empirical results obtained in spe-

cific applications. In the probabilistic modelling of text, we are especially interested in the subclass of Recurrent Neural Networks (RNN). RNNs have been successfully employed on numerous NLP tasks such as language modelling, POS tagging, speech recognition and NER. In RNNs, some of the neural layer activations become inputs to the same layer at the next timestep. Additionally, different from feed-forward neural networks, RNNs can retain information in their internal state. This characteristic can function as a memory cell, preserving long distance relationships across the chain, and making them more suitable for processing sequences, and consequently for solving text related tasks. Figure 4.4 describes a RNN for sequence labeling unrolled through multiple timesteps.



**Figure 4.4.** RNN for NER

At each timestep, the neural network computes a hidden state  $h_t$  using an input vector  $x_t$  and the previous hidden state  $h_{t-1}$ , that retains information from past iterations. The input vector  $x_t$  for the RNN could consist, for example, of word features indicating if the word is an honorific, if it is the word "that", if it is capitalized, etc. However, pretrained word embeddings usually provide better input vectors for language related tasks, as we will discuss in Section 4.3.2. Finally, the RNN produces an output vector  $y_t$  representing the label for that timestep. A common definition for an RNN cell is given by the equations:

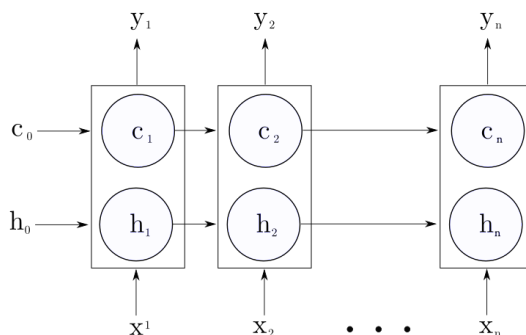
$$h_t = \tanh(W_x x_t + W_h h_{t-1})$$

$$y_t = \text{softmax}(W_y h_t)$$

where  $W_x$ ,  $W_h$  and  $W_y$  are weight matrices that can be trained with the Backpropagation Through Time (BPTT) algorithm [Werbos, 1990]. On a sequence labeling task,  $W_y$  is a  $|H| \times |Y|$  weight matrix where  $H$  is the size of the hidden layer, defined experimentally according to the task, and  $Y$  is the number of classification labels for our problem. With the Softmax activation, the RNN will generate a probability distribution across the range of possible labels and we can simply select the most probable label at each

timestep, assuming independence between labels. Theoretically, RNNs are capable of learning and retaining long term dependencies with their internal state  $h_t$ . However, in practice, it becomes difficult due to the vanishing gradient problem [Bengio et al., 1994], as we will now discuss.

The vanishing/exploding gradient problem is one example of an empirical problem that constrains theoretically sound network designs. When training neural networks, the weight updates are obtained from the partial derivatives of a loss function. As these derivatives are propagated with the chain rule to distant layers, their values tend to get vanishingly small or explode exponentially, what makes training less effective. This problem is particularly critical for RNNs, because there is the need to propagate weights over a substantial number of timesteps. LSTMs were created by Hochreiter and Schmidhuber [1997] with this problem in mind and they have been popularized because of their effectiveness. LSTMs incorporate a long term memory cell in the form of a vector  $c$  in addition to the hidden state already present in the conventional RNN. This new memory is preserved or updated conditionally based on the observed input. A generic description of this architecture is given in Figure 4.5.

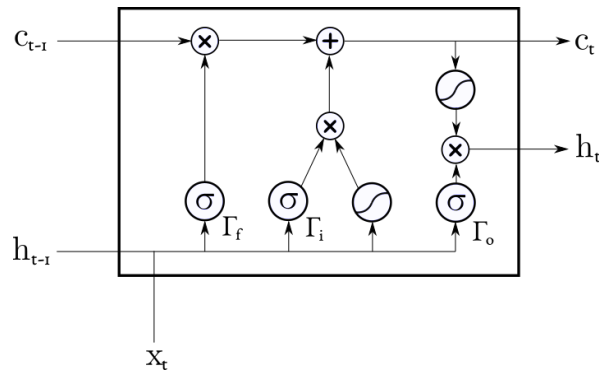


**Figure 4.5.** RNN with a long term memory cell  $c$ .

The LSTM has three gates to control the flow of information that comes in and out of the memory cell  $c$ . The input gate  $\Gamma_i$  controls the amount of new information that will flow into the memory cell, the forget gate  $\Gamma_f$  controls the amount of previous information that will be retained in the memory cell (despite the misleading name), and the output gate  $\Gamma_o$  controls the amount of information stored in the memory cell that will be used to compute the output activation of the LSTM unit. LSTM implementations vary slightly in the literature. A visual description of our LSTM unit is provided in Figure 4.6.

Consider that the vector  $[x_t, h_{t-1}]$  is formed by concatenating the current input vector  $x_t$  and the hidden vector from a previous timestep  $h_{t-1}$ . Also, consider that  $*$





**Figure 4.6.** LSTM Cell

represents the Hadamard Product (element-wise multiplication) of two matrices with the same dimensions. That is:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} * \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{bmatrix}$$

Then, the equations for the LSTM unit are:

$$\begin{aligned} \Gamma_i &= \sigma(W_i \cdot [x_t, h_{t-1}] + b_i) \\ \Gamma_f &= \sigma(W_f \cdot [x_t, h_{t-1}] + b_f) \\ \Gamma_o &= \sigma(W_o \cdot [x_t, h_{t-1}] + b_o) \\ c_t &= \Gamma_f * c_{t-1} + \Gamma_i * \tanh(W_c \cdot [x_t, h_{t-1}] + b_c) \\ h_t &= \Gamma_o * \tanh(c_t) \end{aligned}$$

where  $\sigma$  is the logistic sigmoid function.  $\Gamma_i$ ,  $\Gamma_f$ , and  $\Gamma_o$  are the input, forget and output gates, respectively, and  $W_i$ ,  $W_f$ ,  $W_o$  are the weight matrices corresponding to each gate.  $c_t$  is the cell state at time  $t$  and  $h_t$  is the hidden state at time  $t$ .

### 4.3.1 BI-LSTM-CRF

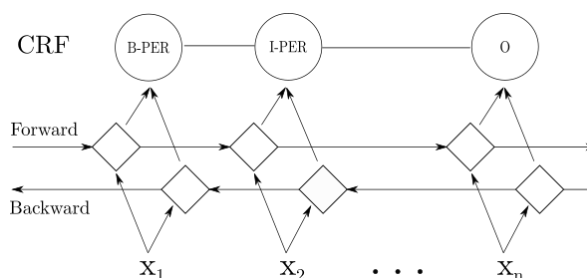
On NER tasks, knowing both past and future words could be important to attribute a label at time  $t$ <sup>3</sup>. For example, in the sentences "Summer Smith is a lawyer" and "Summer came early this year", at the first timestep, it might be useful to know the words after Summer to correctly label the sequences. When jointly decoding label

<sup>3</sup> The timesteps in the context of LSTMs for sequence labeling refer to positions in a sequence. So future words are words that come later in the sequence and past words are words that happen earlier in the sequence.

sequences with the Viterbi algorithm in HMMs or CRFs, these codependencies are taken into account, however a regular LSTM network only takes past hidden states into consideration.

A bidirectional LSTM solves this problem by stacking two regular LSTMs, and feeding them with observations in opposite directions. The first LSTM receives forward states and the second LSTM receives backward states. The hidden states from both networks can then be concatenated at each timestep to produce output labels. With this architecture, LSTM units may use information from past and future timesteps to decide the label at time  $t$ . However, labels are still decoded individually at each timestep, because we are looking at past and future information encoded in the LSTM state but not the actual label predictions.

Huang et al. [2015] proposed a bidirectional LSTM with a CRF<sup>4</sup> layer (BI-LSTM-CRF) on the output to overcome this obstacle. The main benefit of adding a CRF layer in the neural sequence model is that the labels are jointly decoded for a whole sentence instead of being predicted individually. Some predicted labels should be highly correlated in a NER task (e.g. I-PER is very likely to happen after B-PER), so it is desirable to predict sequences conjointly. The BI-LSTM-CRF is described in Figure 4.7.



**Figure 4.7.** Bidirectional LSTM-CRF

### 4.3.2 Word Embeddings

In language-related tasks such as sequence labeling, training data is usually scarce. Because of the huge number of word combinations that may occur in natural language, most combinations will never be observed in the training data. For this reason, it is useful to know to which extent two words are related. For example, if the sentence "John is a person" occurs multiple times in the training data, but the sentence "Joe is a person" never occurs, it may be useful to know that the words "John" and "Joe"

<sup>4</sup> The definition for the Linear-Chain Conditional Random Fields (CRF) layer was given in Section 4.2.

are somewhat related when predicting labels for this new sentence. This is one of the motivations for using word embeddings as input vectors for the recurrent neural network architectures presented earlier in this chapter.

Word embeddings are lower-dimensional representations of words in continuous space. That is, each word is represented by a vector of continuous features, typically 50 to 300 dimensions, containing *Ad Hoc* variables whose values are determined with unsupervised clustering methods for words in large corpora. Word embeddings are lower-dimensional in comparison with the size of the vocabulary, which usually consists of a few hundred thousand words. If we were to use binary features to signal the occurrence of a word (one-hot vectors), we would need one feature for each word, making network training ineffective and the comparison of similarity between different words impossible. With word embeddings, the number of parameters in the sequence model is dramatically reduced. Also, the mapping of words into feature vectors makes possible the comparison of different words according to their shared features (usually using cosine similarity). And lastly, word embeddings can be pre-trained without supervision on large corpora and be used on tasks for which there is little labeled data.

Pre-trained word embeddings are a powerful form of transfer learning. Transfer learning is a group of techniques concerned with bootstrapping the learning of model parameters by obtaining information from another dataset not directly related to the problem. This is especially useful in NER, because named entities that belong to the same class tend to have similar word embeddings. Thus, with good word embeddings, a neural network might predict the correct label for a word that was never seen during training because this label is similar to other words that occurred in the training set.

There are multiple methods to produce word embeddings, and most of the official implementations also provide pre-trained word embeddings for the English language. In this work, we explored three sets of pre-trained embeddings: Word2Vec [Mikolov et al., 2013], GloVe [Pennington et al., 2014] and ELMo [Peters et al., 2018]. The characteristics of each set of embeddings is given in Table 4.1.

Model	Dimensions	Training Set	Tokens (billions)	Vocab.
Word2Vec	300	Google News	100	3M
GloVe	300	Gigaword5 + Wikipedia	42	400K
ELMo	512	Wikipedia	5.5	-

**Table 4.1.** Statistics of pre-trained word embeddings.

Most breakthroughs in sequence labeling in the past few years came with the in-

roduction of novel methods for constructing word embeddings, as was the case for the three methods mentioned earlier. Different from earlier methods such as Word2Vec and GloVe that produced static vectorial representations of words, more recent approaches such as ELMo produce context dependent embeddings for a specific dataset with a neural network pre-trained on a language modelling task. Also, the word representations in ELMo are entirely character based, therefore there are no out-of-vocabulary words.

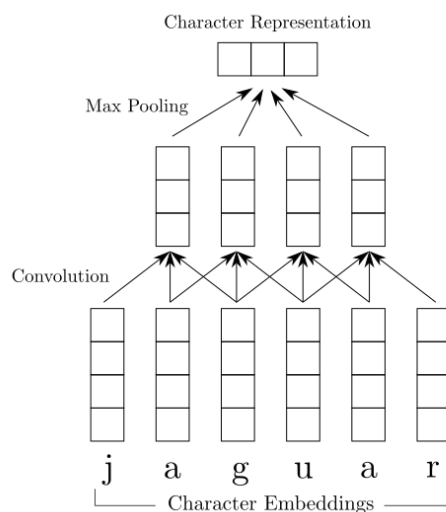
### 4.3.3 Character Representations

When creating deep neural architectures, one of the objectives is to allow the neural network to discover useful features by itself instead of feeding it with what we think is relevant. In NER, morphological features are very useful to classify named entities. For example, knowing that a word ends in "ing" means that it is probably a verb and therefore it is unlikely to be a proper name. We can add a character representation layer at the bottom of the Bi-LSTM-CRF to extract these morphological features automatically. The character representations, which are fixed size vectors that encode morphological features, are concatenated to word embeddings and fed to the LSTM.

In the next Subsection, we describe two methods to build character representations. Both of them receive character embeddings as inputs. Character embeddings are fixed size dense vectors that work the same way as word embeddings, except that the lookup function maps characters to embeddings instead of words to embeddings. This means that each word generates a character embedding matrix with size  $|W| \times |C|$ , where  $|W|$  is the word length and  $|C|$  is the character embedding size (determined experimentally). Also, the character embeddings are trained directly in the target dataset, since the vocabulary size is much smaller than in the case of word embeddings.

#### 4.3.3.1 CNN character representations

The CNN method proposed by Ma and Hovy [2016] uses a CNN layer at the bottom of the Bi-LSTM-CRF architecture to encode character-level information. The CNN layer is described visually in Figure 4.8. The CNN receives character embeddings as inputs. For example, if the character embeddings have 50 dimensions, then the input matrix representing word "jaguar" would be a  $50 \times 6$  matrix. The one dimensional convolution operation works by sliding fixed size kernels over the input matrix producing scalar values at each position. These kernels, also called filters, try to detect morphological patterns in the input matrix such as "ing" or "son". In this example, we define three



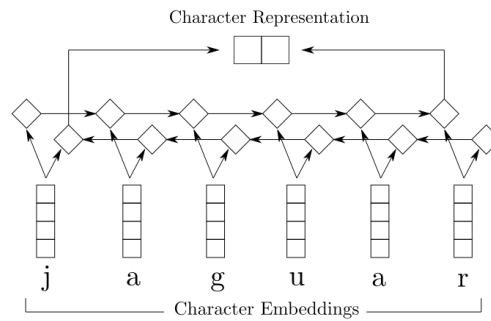
**Figure 4.8.** CNN-based character representations for the word "jaguar".

kernels with window size 3 (which is usually a good number for detecting useful prefixes and suffixes), each kernel consists of a  $50 \times 3$  matrix. At each window position in the word (i.e. "jag", "agu", "gua" and "uar"), we perform a convolution over each of the 50 channels (character embedding features) adding up the results to produce a scalar value. This way, after running the three kernels over the input matrix, we get a  $4 \times 3$  matrix indicating at each window position the extent to which each filter was activated. Finally, we perform a global max pooling operation that collapses the filter dimension taking its maximum value over the 4 window positions, producing a vector with 3 dimensions that indicates for each filter, if it was triggered in that word. This means, that when a filter gets a good match at any position in the sequence, the max pooling layer output relative to that filter will be triggered, therefore this type of character representation detects position invariant morphological features. For example, the word "finger" would trigger a filter for "ing" the same way the word "writing" would trigger this filter.

#### 4.3.3.2 LSTM character representations

The LSTM method proposed by Lample et al. [2016] models character-level representations with a Bi-LSTM similar to the one described in Figure 4.7, however instead of receiving words as inputs, it receives sequences of character embeddings. Also, instead of producing an output at each timestep, we are only interested in obtained the hidden state at the first timestep for the backward LSTM and the hidden state for the last timestep for the forward LSTM. The combined forward and backward states are

concatenated to form the character representation, as described visually in Figure 4.9.



**Figure 4.9.** LSTM-based character representations for the word "jaguar".

The forward state is expected to be a better representation of the suffix of a token, and the backward state is expected to be a better representation of the prefix of a token. This differentiates the architecture from the CNN method, because the LSTM based representations are not positional invariant.

# Chapter 5

## Researcher Name Extraction Dataset

This chapter describes the dataset used to evaluate the performance of sequence labeling models on the task of RNE. We call it the RNE dataset. We attested experimentally that models trained in a popular NER dataset (the CoNLL-2003 English dataset) do not perform well in the RNE task and this is likely due to the differences between plain text and HTML. The lack of other NER datasets built for HTML name extraction imposes the need to construct a dataset specific for RNE.

The RNE task consists of finding researcher names in faculty listings from university webpages across the world, mainly from Computer Science departments. This would be a necessary step when linking researcher profiles from university websites to their entries in public databases. Unlike many WDE datasets, each webpage in the RNE dataset comes from a different website, and therefore has a different format, what makes many WDE approaches impractical. The idea is to explore systems that are general enough to allow accurate entity extraction from different sources while requiring no supervision between different websites.

We collected 145 Computer Science and Engineering faculty pages from 42 different countries in multiple languages, although the English version was favored whenever it was available. We gathered faculty webpages in proportion to the number of universities in each country<sup>1</sup>. The dataset size was limited to a number of pages that could be viably labeled manually and that could still represent many different countries. Also, university pages that contained corrupt HTML or JavaScript code that performed lazy loading of data records were not included in this dataset.

---

<sup>1</sup>A detailed list of universities can be found in <https://univ.cc/world.php>

This chapter is divided as follows. Section 5.1 describes the dataset format and how it was split in three data files. Section 5.2 describes how the evaluation of results in this dataset was carried on. Section 5.3 describes how we obtained a dictionary of relevant named entities for this dataset, and discusses some baselines. Lastly, Section 5.4 compares the characteristics of the RNE dataset with another popular NER dataset, the CoNLL-2003 English NER dataset.

## 5.1 Data Description

Each of the 145 faculty pages was preprocessed and converted to the CoNLL-2003 data format. That is, one word per line with empty lines representing sentence boundaries. Sentence boundaries were determined by line break HTML tags (`div`, `p`, `table`, `li`, `br`, etc.) in contrast to inline tags (`span`, `em`, `a`, `td`, etc.). For example, the HTML sentence:

```
<p>E. <b>Smith</b> is a <a href="/link">professor</a>.</p>
```

becomes:

E. Smith is a professor.

Sentences that were more than fifty tokens long were also split according to the punctuation. Some algorithms have trouble handling very big sentences, so we verified experimentally that fifty tokens provide a large enough context and allow efficient training using a GPU. A few sentences in the dataset would be more than a thousand tokens long if this step was not performed, what would make batch training for neural networks less efficient.

A robust tool for HTML segmentation poses many challenges by itself, but the simple approach adopted here has proved to be sufficient to perform RNE adequately. Also, it is best to evaluate NER models without relying on any sophisticated data record segmentation system. In many cases, entity annotation may precede the segmentation phase on WDE methods. Also, depending on the task (as is the case for researcher name extraction), a good annotator that is able to work with raw HTML provides a good solution to the problem.

Finally, all tokens were tagged using the IOB scheme put forward by Ramshaw and Marcus [1999] and used in CoNLL-2003, this is:

Words tagged with O are outside of named entities and the I-XXX tag is used for words inside a named entity of type XXX. Whenever two entities of type XXX are immediately next to each other, the first word of the



second entity will be tagged B-XXX in order to show that it starts another entity [Tjong Kim Sang and De Meulder, 2003].

The RNE dataset only has entities of type person PER, therefore a classifier has to label each token with one of the labels: O, B-PER, or I-PER. The following example sentence, obtained from the dataset, illustrates proper token labeling:

Token	Correct Label
Kasper	I-PER
Rasmussen	I-PER
Associate	O
Professor	O
,	O
Royal	O
Society	O
Research	O
Fellow	O

The RNE dataset was divided in a training, a validation and a test set, as described in Table 5.1. The sizes of the data files were divided in approximately the same proportion adopted in other sequence labeling datasets such as CoNLL-2003, that is described in Table 5.2. Only the training set was used to fit model parameters. Models that relied on numerical optimization methods such as neural networks were evaluated successively on the validation set during training to avoid overfitting. That is, the validation set was never observed during training, but it provided an unbiased evaluation of model performance. We kept training parameters until the performance in the validation set no longer improved. This is called the early stopping validation strategy. The comparison of different models was conducted by comparing their performance in the test set, which is never observed during training or used in the validation strategy.

Data file	Documents	Sentences	Tokens	Names
Training	85	24,728	110,269	5,822
Validation	30	8,743	36,757	1,788
Test	30	10,399	44,795	2,708
Total	145	43,870	191,821	10,318

**Table 5.1.** Description of the data files in the RNE dataset.

Data file	Documents	Sentences	Tokens	LOC	MISC	ORG	PER
Training	946	14,987	203,621	7,140	3,438	6,321	6,600
Validation	216	3,466	51,362	1,837	922	1,341	1,842
Test	231	3,684	46,435	1,668	702	1,661	1,617
Total	1,393	22,137	301,418	10,645	5,062	9,323	10,059

**Table 5.2.** Description of the CoNLL-2003 English dataset

Most webpages in this dataset are faculty directories with informative text in small passages, even if long prose is not absent. Size and structure varies wildly, therefore some documents may contain up to a few hundred names whereas other documents may contain only a dozen names. This difference in document size and characteristics may be problematic when comparing different extraction systems, because a system that performs well on some types of pages may perform poorly on other types. To avoid this problem, we tried to keep the proportion between the number of tokens and the number of documents roughly the same for all data files.

## 5.2 Evaluation

The performance of classifiers in the RNE dataset was evaluated according to their precision (P), recall (R) and F-scores in the test set. The precision and recall measures are defined in terms of the number of true positives, false negatives, and false positives made by the classifier when extracting named entities. That is:

$$\text{Precision} = \frac{\text{TruePos}}{\text{TruePos} + \text{FalsePos}} \quad \text{Recall} = \frac{\text{TruePos}}{\text{TruePos} + \text{FalseNeg}}$$

Precision accounts for the proportion of named entities found by the model that are correct relative to all predicted named entities. Recall is the proportion of named entities correctly predicted by the model relative to all named entities in the dataset. Precision measures Type I errors (false positives) and recall measures Type II errors (false negatives). Partial matches are not considered, so a classification only counts as a true positive if the entire named entity has been correctly extracted. Additionally, the F-score, proposed by Rijsbergen [1979], is a composite measure that combines precision

and recall, defined as:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}} \quad (5.1)$$

The choice of  $\beta$  depends on the relative importance attributed to precision over recall. This formula “measures the effectiveness of retrieval with respect to a user who attaches  $\beta$  times as much importance to recall as precision” [Rijsbergen, 1979]. A common choice for the value of  $\beta$  is 1, in which case the measure is called the  $F_1$ -score (F1). Whenever the  $F_1$ -score is used, we attribute as much importance to recall as to precision.

In our experiments, we measured the precision, recall and  $F_1$ -scores over the entire data files. Considering that each webpage has a different number of named entities, this naturally privileges models that work well for pages with more named entities. A different approach might be to consider the averaged precision, recall and  $F_1$ -scores per webpage, privileging systems that have more regularity between different websites. However, this approach would also have negative implications. That is, the impact of errors in pages with many named entities would diminish and the impact of errors in pages with few named entities would increase. So, the former approach was preferred.

### 5.3 Dictionary

A dictionary of named entities can be a powerful aid to sequence labeling systems, especially when considering traditional statistical methods. For the RNE task, we extracted a list of 1,595,771 researcher names from the DBLP database and annotated tokens in the RNE dataset with exact and partial match tags. That is, if a sequence of tokens corresponded exactly to a name from the DBLP list, the entire sequence was annotated as an exact match. Otherwise, if only some tokens in the sequence matched a name from the DBLP list partially, the matching tokens were annotated with a partial match tag. This dictionary was used by some of the models that will be discussed in the experiments chapter, however it must be stated that one of the main advantages of Deep Neural Networks relative to other approaches is the fact that they can perform well even without the aid of a dictionary.

To understand how this dictionary can be useful for some models in the RNE, we measured the precision, recall and  $F_1$ -scores for an exact dictionary matching strategy in each RNE data file. The results, shown in Table 5.3, provide a baseline with which to compare other methods of sequence labeling. Any useful extraction method should at least improve upon this approach. Recall is very low because the dictionary is

Data file	Precision	Recall	F1	Correct names
Training	0.7316	0.2303	0.3504	1,341 of 5,822
Validation	0.8474	0.2858	0.4274	511 of 1,788
Test	0.8717	0.3268	0.4754	890 of 2,723

**Table 5.3.** DBLP dictionary coverage in each data file of the RNE dataset, when an exact dictionary matching strategy is used.

incomplete and precision falls short of top performing extraction methods. This last result is surprising since we are only extracting exact matches. The reason for the low precision is that exact dictionary matches may actually only correspond to partial matches in the dataset. For example, if the dictionary has an entry for "Ann Smith", it may partially overlap an entry in the dataset for "Mary Ann Smith", yielding a false positive.

## 5.4 Comparison with CoNLL-2003

The CoNLL-2003 dataset was introduced in the NER shared-task in 2003 and is frequently used to attest the performance of state-of-the-art sequence labeling systems [Huang et al., 2015; Lample et al., 2016; Ma and Hovy, 2016; Peters et al., 2018]. The English data, which is composed of news stories extracted from the Reuters Corpus, provides annotations for four types of entities: people (PER), organizations (ORG), locations (LOC), and miscellaneous (MISC), i.e., entities that cannot be classified in one of the former groups. The statistics for the CoNLL-2003 dataset were already described in Table 5.2.

If the CoNLL-2003 English dataset is used to both train and evaluate state-of-the-art models, it is to be expected that models trained in this dataset will be able to extract the same named entities in other documents with reasonable effectiveness. However, this is hardly the case. This is because tabular HTML documents, such as faculty listings, are very different from plain text. Using our labeled dataset for NER on HTML, we can understand how different news stories written in free text format and faculty listings in HTML format really are.

We trained three classifiers to perform the task of person name extraction using the CoNLL-2003 training set. The CoNLL-2003 dataset was relabeled so that classifiers only had to extract person entities (PER labels), thus other entity types (LOC, ORG, and MISC) were relabeled as not an entity (O label). Next, we trained the classifiers

with the relabeled CoNLL-2003 training set, and validated the results with the relabeled validation set. The models were tested with the relabeled CoNLL-2003 test set and the RNE test set. In Table 5.4, we present the results for a first order HMM, a Linear Chain CRF, and a Bi-LSTM-CRF classifier with GloVe embeddings [Huang et al., 2015] in the aforementioned experimental setting.

Model	CoNLL-2003			RNE		
	P	R	F1	P	R	F1
HMM	0.777	0.452	0.571	0.189	0.180	0.184
CRF	0.774	0.754	0.764	0.138	0.129	0.133
Bi-LSTM-CRF	0.969	0.931	0.950	0.282	0.258	0.269

**Table 5.4.** Performance of three classifiers trained with the CoNLL-2003 training set and tested in the CoNLL-2003 and RNE test sets.

The very low  $F_1$ -score obtained by all models on the RNE test set is much lower and even worse than the established dictionary baseline in Table 5.3. We cannot use these results to properly compare the relative performances between the models (this topic will be explored in the next chapter), but it is possible to conclude that machine learning models trained on plain text do not necessarily perform well in HTML extraction tasks. The Bi-LSTM-CRF model obtained a 0.269  $F_1$ -score in the RNE test set, what is surprising considering that it is a very efficient approach to sequence labeling in plain text. By looking at some of its mistakes we can shed some light on this result. The Bi-LSTM-CRF model extracted "Dean Emeritus" and "14101 INF" as names, and it missed the last names in "George E. Elliott" and "Carl J. Galligan". These mistakes are representative of some of the major flaws with the systems trained with the CoNLL-2003 training set. The names in the RNE dataset are usually longer than in CoNLL-2003 and the word distributions are vastly different, what may account for some of the mistakes of the Bi-LSTM-CRF model.

Let us examine some differences between both datasets. The number of documents in the RNE dataset is 145, which is much smaller than the 1,393 documents in the ConLL-2003 dataset. However, the number of sentences in the RNE dataset is higher, that is 43,870 sentences against 22,137 in the ConLL-2003 dataset. Also, the number of tokens in the RNE dataset is roughly half the number present in the CoNLL-2003 dataset. These numbers show that the HTML documents in the RNE dataset are longer than news stories and, more importantly, they are composed of much shorter sentences when compared to the text from news corpora. Indeed, there are 3.46 words

per sentence in the RNE dataset against 13.62 words per sentence in the CoNLL-2003 dataset. Most faculty webpages have a lot of boilerplate text that contributes to their size, while the news stories in CoNLL-2003 are usually rather short. These numbers show that HTML provides far less context to named entity recognition by sequence models. One key implication is that any good classifier needs to seek other sources of evidence in addition to the text, such as dictionaries and the HTML structure.

Another property that is quite distinct among different NER datasets is their word distributions. Word frequencies tend to vary considerably in documents with different topics. Table 5.5 shows the ten most frequent words for each dataset, including punctuation signs. Punctuation signs are frequently present in proper names, as in: Mary B. Smith, Susan (Susie) Williams, John Al-Azzawi, etc. Also, punctuation signs can be useful to detect boundaries between named entities. The CoNLL-2003 English dataset contains a more generic selection of terms whereas the subject of the RNE dataset becomes evident with words such as "professor" and "university" happening with a high frequency in the corpus. Also, punctuation signs are much more frequent in the RNE dataset.

Lastly, in Figure 5.1 we plot the word frequencies for the hundred most frequent words in the CoNLL-2003 and the RNE datasets. In both plots we have a few very frequent words and a long tail of infrequent words. Most named entities are located in this long tail, and therefore, a good sequence labeling method needs to be able to handle the labeling of infrequent tokens effectively. The probabilities attributed to unseen tokens is another aspect where models vary significantly between different datasets.

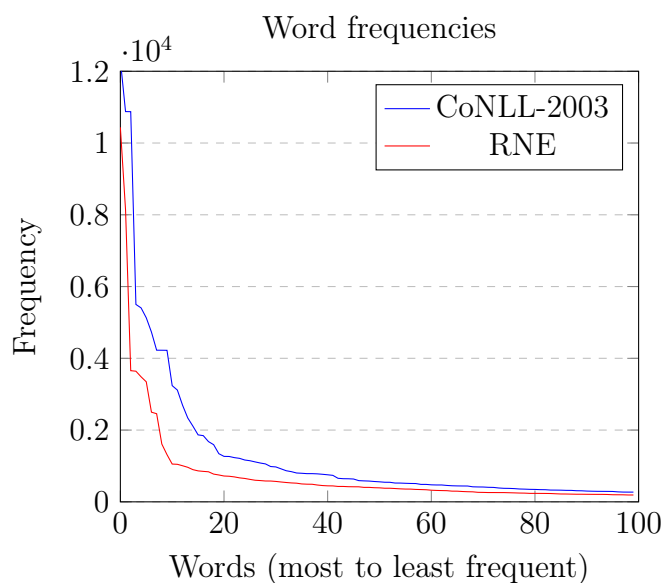
## 5.5 Dataset Size

The reduced size of the RNE dataset (only 145 webpages) may prevent us from drawing conclusive judgements about the expected performance of different NER models in the RNE task. The matter of sample size is difficult to address because there are no alternative data sources to confirm the claim that the RNE dataset is in fact a representative sample of faculty directories across the world. The number of tokens in the RNE dataset is roughly half the number present in the CoNLL-2003 dataset, a dataset that is often used to attest the performance of NER models in plain text. However, the size of the CoNLL-2003 dataset is also small and presents an obstacle in the effort of drawing general conclusions about model performance.

Some facts may contribute to ascertain the validity of the RNE dataset as a

CoNLL-2003		RNE	
Word	Frequency	Word	Frequency
the	12310	,	10439
,	10876	-	8140
.	10874	)	3655
of	5502	(	3641
in	5405	:	3484
to	5129	of	3345
a	4731	and	2499
(	4226	professor	2456
)	4225	university	1611
and	4223	research	1315

**Table 5.5.** Ten most frequent words for the CoNLL-2003 and the RNE datasets.



**Figure 5.1.** Word frequencies plot for the CoNLL-2003 dataset and the RNE dataset.

representative sample of faculty directories. The RNE dataset contains webpages from 145 universities from 42 countries. This amounts to approximately 1.5% of the total number of universities listed in the Universities Worldwide Database.<sup>2</sup> Each country contributes to the RNE dataset roughly in proportion to the number of universities in that country relative to the total number of universities in the world. In Table 5.6,

<sup>2</sup><https://univ.cc/world.php>

we present the number of documents from each country for the top ten countries with most universities in the RNE dataset. For comparison, we also present the number of universities from each country in the Universities Worldwide Database. This database lists 9623 universities.

The geographical distribution of the documents in the RNE dataset is roughly proportional to the number of universities in larger countries. One of the principal limitations of the RNE dataset is that the documents from countries with few universities (below the top ten countries) represent only 29.66% of all documents in the dataset, while these universities actually represent more than half of the universities in the world (55.01%) according to the Universities Worldwide Database.

Despite the significant variation in university webpage layouts, the RNE dataset contain a very diverse range of data representations and languages, therefore it is not likely that many university faculty pages outside the RNE dataset will present content in a way that deviates considerably from what is already present in the dataset. We expect that the results obtained with the RNE dataset will remain stable when the models are employed over larger samples.

Country	RNE	%	# Universities	%
USA	35	24.14%	2051	21.31%
Japan	11	7.59%	568	5.90%
Germany	11	7.59%	281	2.92%
China	10	6.90%	392	4.07%
UK	8	5.52%	165	1.71%
Canada	7	4.83%	145	1.51%
Australia	6	4.14%	50	0.52%
France	5	3.45%	265	2.75%
Italy	5	3.45%	92	0.96%
Russia	4	2.76%	320	3.33%
Other	43	29.66%	5294	55.01%
Total	145	100.00%	9623	100.00%

**Table 5.6.** Number of documents per country for the ten most frequent countries in the RNE dataset and the number of universities in each country according to the Universities Worldwide Database.



# Chapter 6

## Experiments

The main objective of this dissertation is to find the best methods for performing NER on the Web considering the RNE task. The value of a specific method is determined not only by its accuracy, but also by the amount of feature engineering required, the training time required, and the overall complexity of the model when we consider its gains relative to simpler alternatives. With this objective in mind, we pose the following research questions:

1. How effective are HMMs for solving the RNE task?
  - a) Does the order of the HMM influence its performance?
  - b) Can we improve a HMM by adding textual features?
  - c) Can we improve a HMM with the self-training strategy?
2. How effective are CRFs for solving the RNE task?
  - a) Are CRFs better than HMMs?
  - b) Can we improve the CRFs by adding textual features?
  - c) Are CRFs more resilient to bad features than HMMs?
3. How effective are Neural Networks for solving the RNE task?
  - a) How does a Bi-LSTM-CRF compare with HMMs and CRFs?
  - b) Can character representations boost the performance of the Bi-LSTM-CRF?
  - c) Does the selection of word embeddings influence the performance of the Bi-LSTM-CRF?
4. All things considered what is the best model?

a) What model has the best F1-score?

All models in this dissertation were trained and tested in the dataset described in Chapter 5. When relevant, the technical specifications for the experiments will be given in each section. Section 6.1 establishes a baseline in addition to the dictionary matching baseline that was described in Chapter 5 for the RNE task. Section 6.2 discusses the experimental results that concern Research Question 1. Section 6.3 discusses the experimental results that concern Research Question 2. Section 6.4 discusses the experimental results that concern Research Question 3. Finally, Section 6.5 discusses the experimental results that concern Research Question 4.

## 6.1 Simple Baseline

A model that works well despite the absence of a good feature selection is arguably more valuable than a method that requires a lot of feature engineering or a big dictionary of named entities, as long as its accuracy is sufficiently high for the intended purpose. For this reason, we wanted to establish some baseline results in addition to the dictionary matching results presented in Chapter 5. We trained a simple generative model and a simple discriminative model to provide a measure of the expected difficulty of solving this task. The simple generative model is a Naive Bayes classifier, which is essentially a HMM, as presented in Section 4.1, with the only difference being that it assumes label independence. That is,  $P(Y_i|Y_{i-1}) = P(Y_i)$  for all timesteps  $i$ . The simple discriminative model is a Logistic Classifier, as presented in Section 4.2.

The “Naive-Bayes and Logistic Classifiers form a generative-discriminative pair”, as explained by Ng and Jordan [2001], in the sense that the essential difference between the models is that the first estimates the joint probability between words and labels  $P(X, Y)$  and the latter models the conditional probability  $P(Y|X)$ . The same analogy could be made for HMMs and CRFs. The Naive Bayes model and the Logistic Classifier are very simple and fast to train, so they provide a good first approximation to a solution for this task. Both models used only the current word as a feature. Table 6.1 shows a comparison between these models and the exact dictionary matching approach<sup>1</sup> presented in Chapter 5. We show results for the models in the validation and test sets of the RNE dataset. The difference between the validation and test sets is only of significance when discussing neural networks, which are trained in the training set until the F1-score in the validation set stops improving. For the other models, the data

---

<sup>1</sup> That is, using a dictionary of researcher names extracted from DBLP, a sequence of tokens was labeled as a name if it corresponded exactly to an entry in this dictionary.

from the validation and test sets is never observed during training. When comparing different models we will always consider results in the test set.

Model	Validation			Test		
	P	R	F1	P	R	F1
Dictionary Matching	0.847	0.285	0.427	0.871	0.328	0.477
Naive Bayes	0.068	0.053	0.059	0.101	0.075	0.086
Logistic Classifier	0.952	0.171	0.122	0.116	0.156	0.133

**Table 6.1.** Naive Bayes and Logistic Classifier results for the researcher name extraction task using only the current word as a feature.

The Logistic Classifier performs a little better than the Naive Bayes model in this problem, but both models are rather ineffective. The problem is that these classifiers always assign an O tag to unknown tokens <sup>2</sup>, what is understandable since the proportion of O tags in the dataset is approximately 86% and each token is classified independently. Table 6.2 shows the number of unknown tokens in the validation and test sets, and the number of named entities that contain at least one unknown token. Roughly four in every five named entities contain at least one unknown token, therefore, a classifier that only considers the current token as a feature and attributes labels independently can obtain at best 20% recall. Also, the proportion of unknown tokens in the validation set is higher than in the test set, what may account for the drop in performance for both classifiers in the validation set relative to the test set.

The Naive Bayes and the Logistic Classifier performed significantly worse than the dictionary matching approach. These models are not suited for sequence classification problems and that is why we need approaches such as HMMs, CRFs, and Neural Networks. However, these baselines provide a very useful insight. That is, the amount of probability mass attributed to unknown observations is of great importance to the performance of NER methods.

## 6.2 Hidden Markov Models

There are a couple of parameters that need to be considered when we employ Hidden Markov Models to a sequence labeling task on the Web. First, the number of previ-

<sup>2</sup> Unknown tokens are tokens that occur at least once in the validation or test sets but do not occur in the training set.

Tokens / Named Entities (NE)	Valid	%	Test	%
Unknown Tokens	12076	26.96%	8324	22.65%
Known Tokens	32719	73.04%	28433	77.35%
Total	44795	100.00%	36757	100.00%
NEs with Unknown Tokens	1446	80.87%	2205	80.98%
NEs without Unknown Tokens	342	19.13%	518	19.02%
Total	1788	100.00%	2723	100.00%

**Table 6.2.** Unknown tokens in the RNE dataset.

ous states (the order of the HMM), second, the features, and third, the self-training strategy.

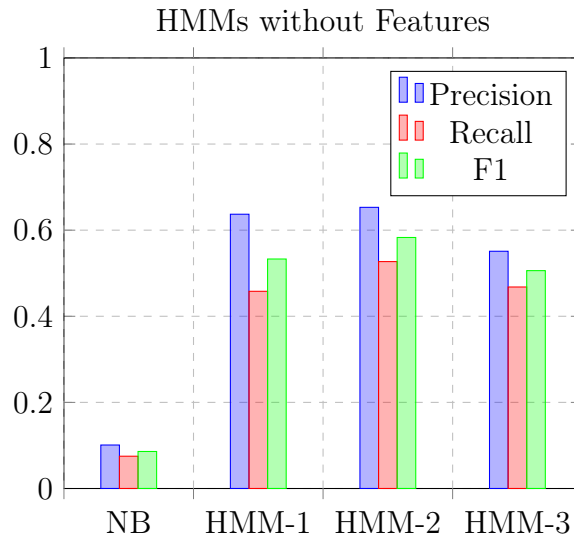
### 6.2.1 HMM order

We hypothesized that the performance of HMMs on sequence labeling tasks for NLP can be improved by increasing the number of previous states checked at each timestep, i.e., the order of the HMM. Figure 6.1 presents the performance of *HMMs* up to third order in the test sets of the Researcher Name dataset, and Table 6.3 presents the same results numerically. To allow comparison with the baselines, we also add the results of a Naive Bayes classifier. The Naive Bayes classifier can be thought of as a HMM of zeroth-order.

Model	Validation			Test		
	P	R	F1	P	R	F1
Naive Bayes	0.068	0.053	0.059	0.101	0.075	0.086
HMM-1	0.693	0.581	0.632	0.637	0.458	0.533
HMM-2	0.703	0.630	0.665	<b>0.653</b>	<b>0.527</b>	<b>0.583</b>
HMM-3	0.616	0.618	0.617	0.551	0.468	0.506

**Table 6.3.** Performance of the Naive Bayes classifier and Hidden Markov Models with no features besides the current word on the validation and test sets of the RNE dataset.

Considering label dependencies over multiple timesteps increases the classifier performance by a considerable margin relative to the Naive Bayes model even without any additional features. The second order HMM (HMM-2) achieved a  $F_1$ -score of 0.583 at the task, while the Naive Bayes classifier achieved a  $F_1$ -score of only 0.086. All



**Figure 6.1.** Performance of the Naive Bayes classifier and Hidden Markov Models with no features besides the current word on the test set of the RNE dataset.

Hidden Markov Models are also better than the dictionary matching approach, which only achieved an  $F_1$ -score of 0.477. Nevertheless, we already observe a relative decline in performance with the third order HMM (0.506  $F_1$ -score), suggesting that increasing the number of previous timesteps is not always advantageous. As we increase the window of previous labels at each timestep, the label combinations get less common in the training set, and therefore, the probability estimates get less reliable.

### 6.2.2 Feature selection

Eleven discrete features were extracted from the dataset. These were selected from a larger pool of features, considering their aid to the performance of the extraction systems. Deep learning architectures can work incredibly well without any of these features, however they are of critical importance to traditional approaches such as HMMs and CRFs. The selected features are presented in Table 6.4.

The only features that demand further explanation are features 10 and 11. Feature 10 is the token’s enclosing HTML tag and its parent tag concatenated. Feature 11 is the CSS class for the token’s HTML tag. Both of these features are only useful in the self-training strategy for HMMs and the attention architectures for Bi-LSTM-CRFs. In other models, we use features 1 to 9.

HMMs can benefit considerably from a good feature selection, but too many features can lead to a drop in performance because of the feature independence as-

Feature	Description	Type
1	Unaccented lowercase token	Categorical
2	Exact dictionary match	Binary
3	Partial dictionary match	Binary
4	Email	Binary
5	Number	Binary
6	Honorific (Mr., Mrs., Dr., etc.)	Binary
7	Matches a URL	Binary
8	Is capitalized	Binary
9	Is a punctuation sign	Binary
10	HTML tag + parent	Categorical
11	CSS class	Categorical

**Table 6.4.** Features used in the RNE dataset. Feature 10 is the token’s enclosing HTML tag and its parent tag concatenated.

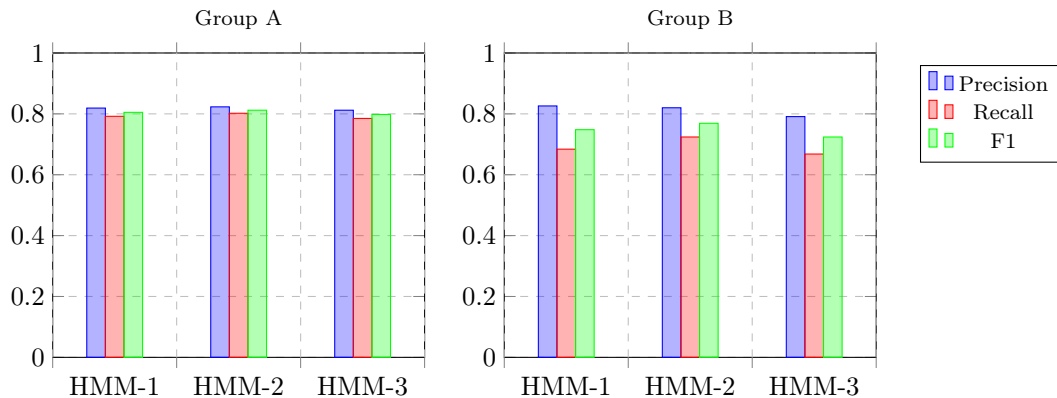
sumption. Correlated features may contribute more to the final prediction than should be the case if the correlation was taken into consideration. For example, if two features have complete correlation and both are introduced in the model, the effect is the same as giving twice the weight to a single of these features. If too many features are correlated, we can inadvertently give too much weight to bad features, what will introduce noise in the predictions. To understand how a feature selection can help or hurt the Hidden Markov Model performance we tested the model using three groups of features:

- Group A: Current word, Exact Match, Partial Match, URL, Capitalized
- Group B: All features except for the HTML Tag and the CSS Class
- Group C: The HTML Tag and the CSS Class

Group A is composed of a small selection of indicative features while Group B is composed of all the textual features. Group C will only be used in the Self-Training strategy.

Figure 6.2 compares the performance of HMMs using the features from Group A and Group B. Table 6.5 presents the numerical results for the same models.

The Group A models were better overall showing that too many features can hurt the performance of HMMs. However, when these features are carefully selected, they can improve the performance of the HMM significantly (0.812 F1 for the HMM-2)



**Figure 6.2.** Hidden Markov Models trained with the features in Group A and Group B.

Model	Validation			Test		
	P	R	F1	P	R	F1
HMM-1 (Group A)	0.813	0.816	0.814	0.819	0.792	0.805
HMM-2 (Group A)	0.787	0.820	0.803	0.823	<b>0.802</b>	<b>0.812</b>
HMM-3 (Group A)	0.774	0.816	0.795	0.812	0.785	0.798
HMM-1 (Group B)	0.730	0.714	0.722	<b>0.826</b>	0.684	0.748
HMM-2 (Group B)	0.720	0.710	0.715	0.820	0.724	0.769
HMM-3 (Group B)	0.721	0.702	0.711	0.791	0.667	0.724

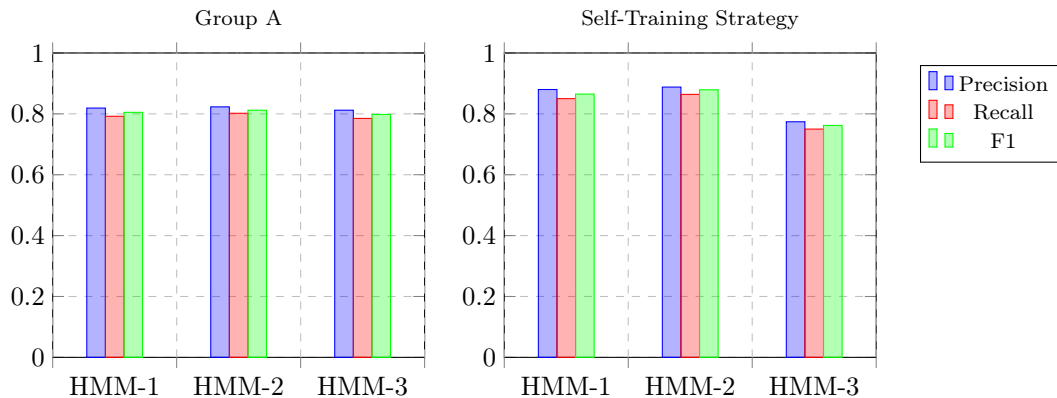
**Table 6.5.** Performance on the validation and test sets for Hidden Markov Models trained with the features in Group A and Group B.

relative to the performance of the HMMs that used only the current word as a feature (0.583 F1 for the HMM-2).

### 6.2.3 Self-training strategy

In the next experiment, we wanted to understand if the self-training strategy described in Section 4.1.3 is effective for improving the performance of HMMs. Figure 6.3 compares the performance of HMMs up to third order with features from Group A and HMMs that were trained with Group A features and Self-Trained with Group C features. The numerical results for the same models are presents in Table 6.6.

The self-trained models show a marked improvement in comparison to the models with no self-training at the test set except for the third order *HMM*. With this, we conclude that the best *HMM* for the name extraction task is a second-order *HMM*



**Figure 6.3.** HMMs trained with the features in Group A and HMMs trained with features from Group A and self-trained with features from Group C.

Model	Validation			Test		
	P	R	F1	P	R	F1
HMM-1 (Group A) + Self-Training	0.752	0.876	0.810	0.880	0.851	0.865
HMM-2 (Group A) + Self-Training	0.784	0.892	0.835	<b>0.888</b>	0.864	<b>0.879</b>
HMM-3 (Group A) + Self-Training	0.789	0.891	0.837	0.774	0.750	0.762
HMM-1 (Group B) + Self-Training	0.747	0.906	0.819	0.866	0.867	0.866
HMM-2 (Group B) + Self-Training	0.771	0.912	0.835	0.885	<b>0.869</b>	0.877
HMM-3 (Group B) + Self-Training	0.788	0.917	0.847	0.826	0.803	0.814

**Table 6.6.** HMMs trained with the features in Group A and HMMs trained with features from Group A and self-trained with features from Group C.

using *Group A* features and the self-training strategy using *Group C* features. This model achieves a *F1* of 0.879.

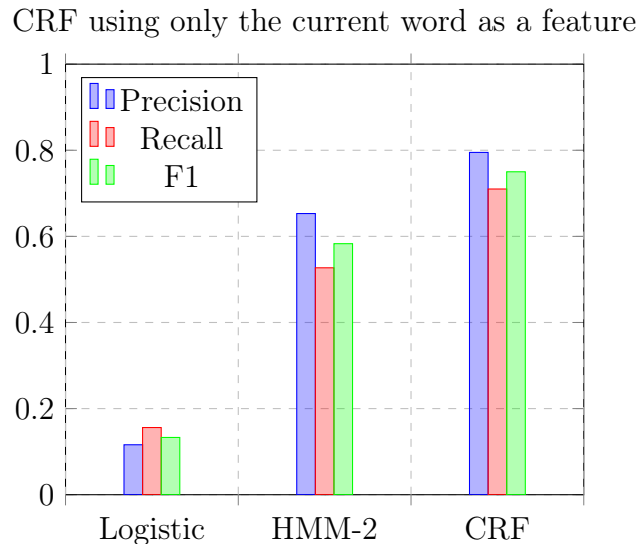
### 6.3 Conditional Random Fields

When we compared a Logistic Classifier with a Naive Bayes model, we found that the Logistic Classifier performed a little better. In this section we want to check if CRFs are also better than HMMs when using only the current words as features. Also, CRFs provide a much more flexible way to incorporate features in comparison to HMMs. When considering the application of this class of models to the Researcher Name Extraction task, we want to understand how the feature selection impacts the performance of CRFs.



### 6.3.1 Only the Current Word Feature

Figure 6.4 shows a comparison between the Logistic Classifier (that assumes label independence), the best HMM with no features besides the current word (HMM-2) and a CRF with no features besides the current word in the RNE task. Table 6.4 shows the numerical results for the same models.



**Figure 6.4.** Performance of the Logistic Classifier, second-order HMM and CRF with no features besides the current word on the test set of the RNE dataset.

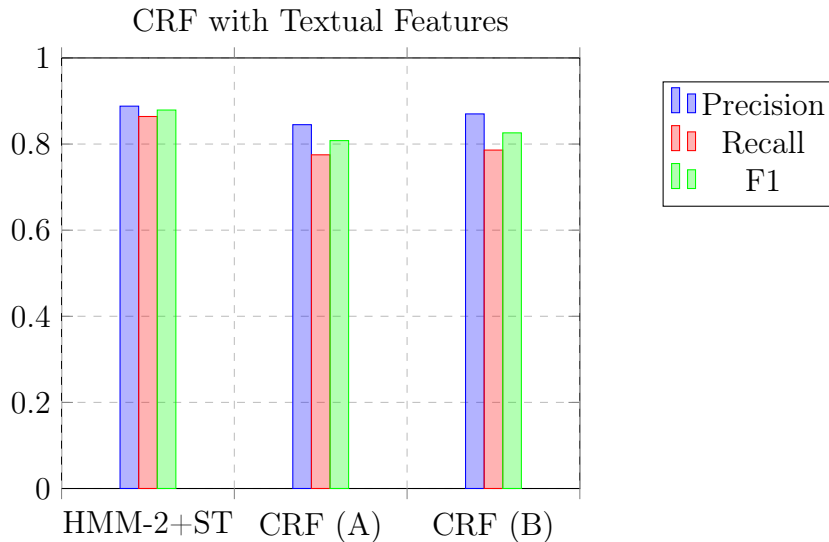
Model	Validation			Test		
	P	R	F1	P	R	F1
Logistic Classifier	0.952	0.171	0.122	0.116	0.156	0.133
HMM-2i	0.703	0.630	0.665	0.653	0.527	0.583
CRF i	0.806	0.805	0.806	<b>0.795</b>	<b>0.710</b>	<b>0.750</b>

**Table 6.7.** Performance of the Logistic Classifier, second-order HMM and CRF with no features besides the current word on the validation and test sets of the RNE dataset.

The CRF shows a significant improvement in comparison to the other models that used only the current word as a feature, achieving a F1-score of 0.750. Next, we proceed to understand if the addition of textual features can improve this performance further.

### 6.3.2 Feature Selection

To allow comparison between the HMMs from the last section, we consider CRFs using features from Group A and Group B. Figure 6.5 shows a comparison between the best HMM and the CRFs using the features from Group A and Group B. Table 6.8 presents the numerical results.



**Figure 6.5.** CRFs trained with the features from Group A and Group B and the HMM-2 trained with features from Group A and self-trained with features from Group C.

Model	Validation			Test		
	P	R	F1	P	R	F1
HMM-2 (Group A) + Self-Training	0.784	0.892	0.835	<b>0.888</b>	<b>0.864</b>	<b>0.879</b>
CRF (Group A)	0.857	0.875	0.866	0.845	0.775	0.808
CRF (Group B)	0.881	0.903	0.892	0.870	0.786	0.826

**Table 6.8.** Conditional Random Fields using features from Group A and Group B.

The CRF is more robust to variations in the feature selection. It performs similar to the second-order HMM in Group A and slightly better in Group B when we consider the HMMs without the self-training strategy. However, when we consider the self-trained HMM-2, it still shows a better overall performance with an F1 score of 0.879.

## 6.4 Neural Networks

A neural network architecture that has had remarkable success at solving sequence labeling tasks is the Bi-LSTM-CRF described in Section 4.3.1. In Section 6.4.1, we investigate how Bi-LSTM-CRFs compare to HMMs and CRFs in the RNE task. In Section 6.4.2, we check if CNN-based or LSTM-based character representations can boost the performance of Bi-LSTM-CRFs in the same task. Lastly, in Section 6.4.3, we check if the choice of pre-trained word embeddings alters the results significantly.

Different from the previous models, that converged to an optimal set of parameters, the neural networks must resort to numerical optimization methods over a rugged optimization function, therefore they may get trapped in local minima and never find the best set of parameters. The results for the models may vary between different runs. Therefore we present the average results for each measure over five runs of each model variation.

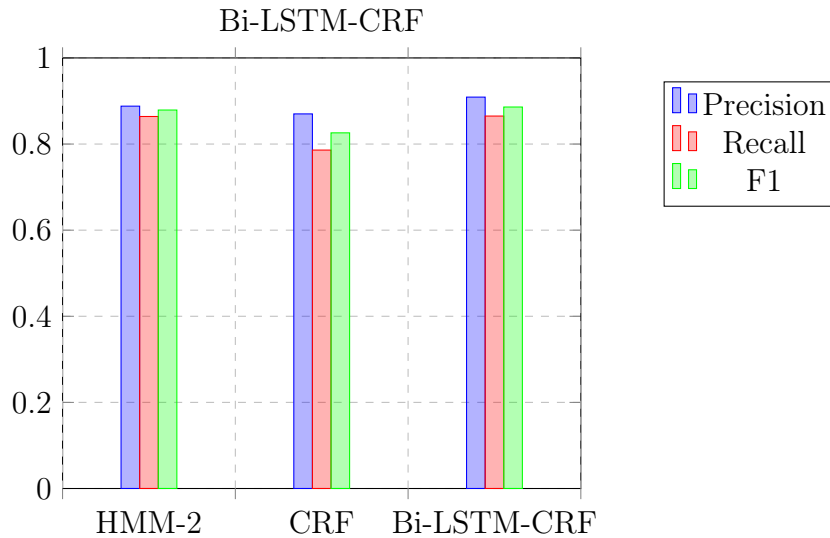
### 6.4.1 Bi-LSTM-CRF

One of the advantages of deep neural networks relative to earlier sequence labeling methods is that they usually work without any feature engineering. Figure 6.6 compares the performance of a BI-LSTM-CRF on the test set of the researcher name extraction task with the best HMMs and CRFs. We only use GloVe-300 word embeddings as inputs to the Bi-LSTM-CRF model. Table 6.9 presents the numerical results for the same models.

Model	Validation			Test		
	P	R	F1	P	R	F1
HMM-2 (Group A) + Self-Training	0.784	0.892	0.835	0.888	0.864	0.879
CRF (Group B)	0.881	0.903	0.892	0.870	0.786	0.826
Bi-LSTM-CRF	0.906	0.938	0.922	<b>0.909</b>	<b>0.865</b>	<b>0.886</b>

**Table 6.9.** Results for the HMM using features from Group A and self-training, the CRF with features from Group B and the Plain Bi-LSTM-CRF.

The Bi-LSTM-CRF is better than the previous models, though the comparison is not completely fair, since we fed the model with word embeddings. However, these pre-trained embeddings are static and general to any language related task, so their incorporation in the model does not require any substantial effort. Without feature engineering, the Bi-LSTM-CRF model already achieves an 0.886 F1-score, surpassing



**Figure 6.6.** Performance of the Bi-LSTM-CRF with only GloVe embeddings in comparison to the HMM-2 (Group A) with self-training and the CRF (Group B).

the best model discussed in the previous experiments (the HMM-2 with self-training, which achieved 0.879 F1-score).

## 6.4.2 Character Representations

Morphological features can help identifying named entities. We presented in Section 4.3.3 two methods for incorporating character representations in Recurrent Neural Networks, the CNN-based method, and the LSTM-based method. In Table 6.10, we compare the performance of CNN character representations (CNNc) and LSTM character representations (LSTMc) with the plain Bi-LSTM-CRF. The character embeddings had 50 weights, the CNN character representations used 50 filters with a window of size 3 and the LSTM-character representations used a bi-LSTM with 25 hidden states in each direction. Both techniques produced character representations with 50 weights.

The LSTM character representations improved the  $F_1$ -score by 0.015 points relative to the plain Bi-LSTM-CRF that only used GloVe-300 word embeddings. Also, both the CNN-based and the LSTM-based representations have a similar performance, yet the LSTM-based representations are slightly better, owing probably to the fact that they can represent prefixes and suffixes while the CNN-based filters are positional invariant.

Model	Validation			Test		
	P	R	F1	P	R	F1
Bi-LSTM-CRF	0.906	0.938	0.922	0.909	0.865	0.886
Bi-LSTM-CRF+CNNc	0.929	0.946	0.938	<b>0.921</b>	0.881	0.901
Bi-LSTM-CRF+LSTMc	0.928	0.950	0.939	0.920	<b>0.886</b>	<b>0.902</b>

**Table 6.10.** Bi-LSTM-CRF with Character Representations.

### 6.4.3 Word Embeddings

All neural architectures used in the experiments discussed in the previous Sections used GloVe-300 pre-trained word embeddings. However, the choice of embeddings is of great importance to the success of neural sequence models. In fact, most recent developments in NER models come from the incorporation of better pre-trained word embeddings to state-of-the-art models in many NLP tasks [Peters et al., 2018; Devlin et al., 2018]. We considered three sets of pre-trained word embeddings in our experiments: GloVe-300, Word2Vec-300 and ELMo. The characteristics of these word embeddings is described in Table 4.1. Different from GloVe-300 and Word2Vec-300, which are static, ELMo embeddings are context dependent and need to be recalculated for the specific dataset. This adds significant overhead to the model training. While the Bi-LSTM-CRF with LSTM character embeddings and GloVe-300 embeddings took approximately 4,412 seconds to train and run predictions, the same model with ELMo embeddings took approximately 9,437 seconds. Table 6.11 compares the performance of Bi-LSTM-CRF models with LSTM-based character representations using different sets of pre-trained word embeddings.

Model	Validation			Test		
	P	R	F1	P	R	F1
GloVe	0.928	0.950	0.939	<b>0.920</b>	<b>0.886</b>	<b>0.902</b>
Word2Vec	0.925	0.926	0.926	0.899	0.831	0.864
ELMo	0.917	0.953	0.937	0.876	0.810	0.842

**Table 6.11.** Bi-LSTM-CRF with LSTM characters and different sets of word embeddings.

GloVe embeddings are superior to Word2Vec in our NER setting, agreeing with the reported results for neural models in the CoNLL-2003 English dataset for

NER [Huang et al., 2015; Lample et al., 2016; Ma and Hovy, 2016]. But, ELMo embeddings showed a very poor performance, contrasting with the results reported in the CoNLL-2003 English dataset [Peters et al., 2018]. This is probably due to the fact that, different from static embeddings, ELMo considers the context when generating word representations. The neural language model that originates the embeddings was trained by Peters et al. [2018] in text extracted from Wikipedia, so the contextual representations generated by this language model are specifically tuned for plain text. ELMo seems to be highly reliant on this contextual information, but the HTML sentences in the RNE dataset provide very little context. Apparently, this specificity of the RNE dataset contributes negatively to the quality of the word embeddings.

The results obtained with different sets of embeddings are important because they show how the choice of word embeddings may influence the performance of a sequence model considerably. In fact, if we used Word2Vec embeddings instead of GloVe embeddings, the Bi-LSTM-CRF would actually lose to the self-trained HMM-2, which reached a 0.879  $F_1$ -score. Also, we cannot trust that only because an embedding shows superior performance in related tasks that it will also be effective in the present task.

#### 6.4.4 Technical Details for Neural Networks

All neural models were trained with the Adam Optimizer using a learning rate of 0.001 over 20 epochs on mini batches of size 10. We used early stopping [Caruana et al., 2000] to select the best parameters, considering the F1 measure in the validation set. All Bi-LSTM-CRF models contained a single LSTM layer with 100 weights. Dropout layers with 0.5 dropout rates were added after the LSTM, the character representations, and the attention matrix when applicable. The neural models were trained on Amazon G3.4xlarge instances, which have NVIDIA Tesla M60 GPUs with 8GB internal memory. The implementations were made entirely in Python using Google’s Tensorflow deep learning library <sup>3</sup>.

### 6.5 What is the best model?

We reach the point where we try to answer the main question of this dissertation. We have experimented with multiple sequence labeling methods in the previous Sections and now we provide an overview of the best variants of each category. In Table 6.12,

---

<sup>3</sup><https://www.tensorflow.org/>.

we compare the performance of the best HMMs, CRFs and Neural Networks for the RNE dataset.

Model	Validation			Test		
	P	R	F1	P	R	F1
HMM-2 (Group A) + Self-Training	0.784	0.892	0.835	0.888	0.864	0.879
CRF (Group B)	0.881	0.903	0.892	0.870	0.786	0.826
Bi-LSTM-CRF + LSTMc	0.928	0.950	0.939	<b>0.920</b>	<b>0.886</b>	<b>0.902</b>

**Table 6.12.** Overview of the best models for the name extraction task in the RNE dataset.

Taking the F1-score as the comparison criterion, the Bi-LSTM-CRF model with LSTM character representations, and Glove-300 embeddings is the winning model with a F1 of 0.902.

Despite the superior performance of Neural Networks in this task, there remains an important consideration to be made. As we increase the complexity and training time required by our models, there is only a scant gain in terms of precision and recall. For example, the Bi-LSTM-CRF achieved an 0.902 F1-score, while the HMM-2 using a Self-Training strategy achieved a 0.879 F1-score. This is a 0.023 increase, which is not insignificant, but it comes to mind if the additional complexity is entirely justifiable.

The Neural Network approach does not demand any feature engineering or self-training strategy, contrasting with the HMM approach, which is highly reliant on the right selection of features and the self-training strategy. However, the absence of human engineering in Neural Network training is questionable to some degree since we need to define many hyper parameters and many details about the neural architecture, such as the number of LSTM layers and hidden weights, where to add dropout layers, which word embeddings to use, etc. In summary, there are many choices to be made, none of which have definitive theoretical justification, so the model variations need to be tested empirically. This means many iterations between setting parameters, obtaining results, and tuning parameters once again. Considering that each of these iterations can take at least a couple of hours to finish, it is not clear if selecting features is really a more arduous struggle.

To the best of our abilities, the Tensorflow implementation of the Bi-LSTM-CRF + LSTM chars running on an AWS GPU instance took approximately 4,412 seconds to train and run predictions in the test set. While, the HMM-2 did the same in approximately 64 seconds, running in a conventional CPU with unoptimized code.

That is roughly seventy times faster. Finally, if we consider the intellectual cost of understanding and implementing both models and the code maintainability of both implementations, the difference is even more profound.

Then, what is the best model? It depends on the end goal. If the goal is to achieve as much accuracy as possible, then definitely go for Deep Neural Architectures. However, in most ordinary implementations, simpler models should probably be preferred. In both cases, models trained on publicly available sequence tagging datasets will probably perform poorly, so most of the effort would certainly be better spent on constructing task specific datasets with a high quality. This points to the necessity of searching for good unsupervised approaches to sequence labeling. This way, we would have models with better flexibility and good accuracy in many extraction tasks. Unfortunately, unsupervised and semi supervised approaches to this problem are still far from serviceable.



# Chapter 7

## Improvements to Neural Networks

In the previous chapters, we discussed neural networks that can be generally employed in a diverse range of NLP tasks. However, there are some specificities to the problem of sequence labeling on the Web that can be explored to allow further improvement of our models. Taking into consideration the success of the self-training strategy for HMMs, which is a form of incorporating knowledge about the HTML structure in the model, we propose two experimental methods to boost the performance of neural networks when sequence labeling Web documents. In Section 7.1, we propose two attention models to incorporate HTML structural information in our neural sequence taggers. In Section 7.2, we propose a new optimization objective for neural networks trained for sequence labeling tasks that can make them prioritize recall over precision, what can be useful in a name extraction task on the Web. Lastly, in Section 7.3, we discuss some experimental results for these strategies.

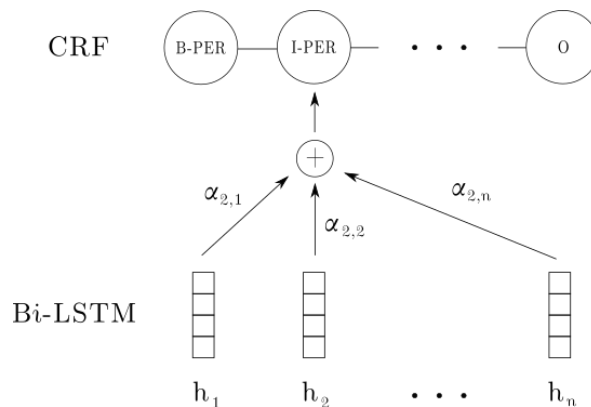
### 7.1 Attentions Models

The self-training strategy for HMMs demonstrates a way to incorporate HTML features in sequence models, however it is not clear how to apply the same intuition to Neural Networks. Ultimately, we want the model to consider the predictions that it made for words in similar HTML contexts when constructing the neural representation for the current word in a sequence. A natural way to incorporate this intuition into the neural architectures described in Section 4.3 is with the use of attention mechanisms, similar to the one proposed by Vaswani et al. [2017]. An attention mechanism is a way to combine inputs from multiple timesteps in a sequence to perform an operation at the current timestep. Originally, attention mechanisms in NLP were developed with the goal of solving sentence alignment for neural machine translation [Bahdanau et al.,

2015], but, since then, they found other uses.

In a Bi-LSTM-CRF model, the bidirectional LSTM layer produces output representations at each timestep, producing a  $T \times H$  matrix, where  $T$  is the number of timesteps in the sequence and  $H$  is the LSTM hidden layer size, which is defined experimentally. This matrix constitutes a neural representation for a sentence with vectors of size  $H$  representing words at each timestep. Also, if the sequence length in the dataset varies, we can simply pad the short sentences with zero vectors.

In the original Bi-LSTM-CRF model without an attention layer, the neural representations would be passed directly to the CRF decoder, but in the Self-Attended Bi-LSTM-CRF, we add an attention mechanism between the LSTM output and the CRF input as described in Figure 7.1.



**Figure 7.1.** Attention mechanism for the Bi-LSTM-CRF model.

Now we need to find a way to combine the vectors at each timestep in a way that transforms the representations according to the similarity between HTML contexts. Essentially, we want to compute a  $T \times T$  attention matrix  $\alpha$  where  $\alpha_{i,j}$  is the weight attributed to the word representation  $h_j$  at timestep  $i$ . In other words,  $\alpha_{i,j}$  is measuring the amount of attention that we pay to each word  $j$  in a sentence at timestep  $i$ . With the  $\alpha$  matrix, we can calculate new representations  $h'_i$  for each timestep by performing a linear combination of the hidden states according to their attention values:

$$h'_i = \sum_j \alpha_{i,j} h_j \quad (7.1)$$

Also, consider a set of  $n$  context vectors  $c_i$  that contain representations for the HTML features at timestep  $i$ . This representation can be a binary feature vector, a dense

vector or even the hidden states  $h_i$ . The attention matrix is then calculated as:

$$\alpha_{i,j} = \frac{e^{A(c_i, c_j)}}{\sum_k e^{A(c_i, c_k)}} \quad (7.2)$$

where  $A(c_i, c_j)$  is an attention function that computes the similarity between HTML contexts at timesteps  $i$  and  $j$  and outputs a real number. The exponentials are a Softmax normalization function to ensure that  $1 \geq \alpha_{i,j} \geq 0$  and  $\sum_j \alpha_{i,j} = 1$ . Next, we propose two ways for defining the attention function  $A$ . The Hard Attention Function and the Soft Attention Function.

### 7.1.1 Hard Attention Function

The hard attention function is a binary similarity function that either outputs one when contexts are identical or zero when they are different. This definition only makes sense when the contexts  $c_i$  at timestep  $i$  are discrete feature vectors, because otherwise we would need a softer comparability criterion. So, if  $c_i = \{f_{i,1}, f_{i,2}, \dots, f_{i,m}\}$  is a context vector where each feature  $f_{i,j}$  assumes a definite value from a discrete set  $\gamma_j$ , we can define the attention function:

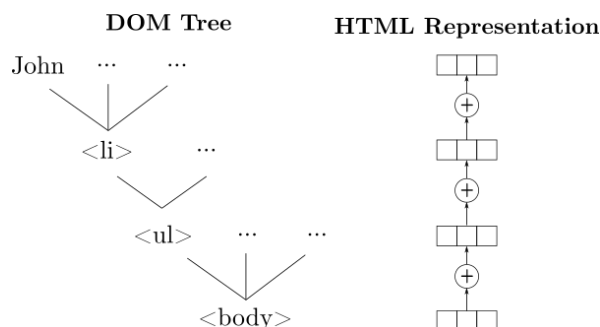
$$A(c_i, c_j) = \begin{cases} 1, & \text{if } f_{i,k} = f_{j,k} \quad \forall k \in [1, m] \\ 0, & \text{otherwise} \end{cases} \quad (7.3)$$

The combination of features must be sufficiently restrictive so that the mixture of hidden states does not introduce too much noise. We have determined experimentally that considering the enclosing HTML tag, its parent tag, and the CSS class as features is sufficient to allow a consistent comparison between similar HTML contexts. Ideally, the choice of features could be performed automatically with the incorporation of a feed forward neural network in the comparison function and a softer similarity criterion. A possibility for doing this is the Soft Attention Function.

### 7.1.2 Soft Attention Function

Different from the Hard Attention Function, the Soft Attention Function outputs a real number that represents the degree of similarity between HTML contexts. Now, instead of considering discrete feature vectors we resort to dense vectorial representations for the HTML context. We propose to extract HTML structural features with DOM path representations. To do this, consider a DOM tree (which is an acyclic directed graph), a word's HTML context can be represented by the path we take in this graph starting

from the root element. If we consider the path for each word in our sequence, we can create HTML dense representations by training a matrix of fixed size embeddings for each HTML tag, then we construct HTML representations by averaging the embeddings in a final vector representation. The process is described in Figure 7.2.



**Figure 7.2.** Building HTML representations by climbing the DOM tree.

Experimentally, we only considered the last two HTML embeddings, since the HTML tag information gets less relevant as we get farther from the leaves. The vocabulary of HTML tags is very small, so we can train HTML embeddings effectively in the target dataset. We could also learn HTML representations with CNNs or LSTMs as we did with character representations, instead of averaging the embeddings.

Now to measure the similarity between different HTML contexts, we resort to an attention function similar to the scaled dot product proposed by Vaswani et al. [2017]. That is:

$$A(c_i, c_j) = \frac{Wc_i \cdot Wc_j}{\sqrt{n}} \quad (7.4)$$

where  $W$  is a weight matrix to be learned and  $\sqrt{n}$  is a normalization factor with  $n$  being the size of the context vectors  $c$ . This function assumes a larger value when the contexts are similar and a smaller value when they are different. With the soft attention mechanism, our model can learn HTML context representations from the training data and produce generalization patterns such as "words happening together in a list probably belong to the same class".

### 7.1.3 Dataset Split and Experimental Considerations

In sequence labeling tasks, it is common to split the dataset in sentences and then train them in independent steps. However, if we want to make use of attention mechanisms to perform the comparison of words in different HTML contexts, we want to compare labels attributed to words that share a similar context in different sentences. To allow

the comparison of words in different sentences, instead of splitting the dataset into independent shuffled sentences, we combine multiple sentences in a single training step and separate them with a segmentation token. Also, with the addition of many parameters to the model, the risk of overfitting increases. To prevent this problem from occurring, we added dropout layers with a 0.5 dropout rate before and after the attention mechanism.

## 7.2 F-score Optimization

In the Bi-LSTM-CRF model, we estimate parameters by maximizing the log-likelihood. The likelihood is intimately associated with the accuracy in a classification task, but in NER tasks, models are usually evaluated according to their attained results in terms of precision, recall, or the F-score. By maximizing the likelihood, we are essentially increasing accuracy with the expectation that this will lead to an improvement of the F-score, what is most often true. However, in extraction tasks there is usually a trade-off between precision and recall, meaning that we can trade a little less precision for an increase in recall and vice-versa. For example, if we want better precision, it may be useful to throw away entities that were not classified with a high degree of confidence. Contrarily, if we want better recall, we can relax the classification criterium and extract entities that otherwise would be ignored.

In NER on the Web, we could argue that recall is slightly more important than precision, because when parsing a huge amount of data, it is much easier to manually filter false positives than to manually find false negatives that were ignored by the classifier. This preference for recall could be expressed in the evaluation by setting the F-score parameter  $\beta$  in a way that values recall twice as much as precision, for example. However, when maximizing the accuracy of a classifier with the cross-entropy function, we have no way to tell the optimizer to choose parameters that maximize the F-score with a particular  $\beta$ . We could try to maximize the F-score directly, but:

While the F-score of a classifier evaluated on a single supervised instance is well defined, the overall F-score on a larger dataset is not a function of the F-score evaluated on each instance in the dataset. This is in contrast to ordinary loss/utility, whose grand total (or average) on a dataset can be computed by direct summation [Jansche, 2005].

This makes the usage of the F-score as an optimization objective impractical for large datasets trained in batches. However, with a few simplifications, we can

replace our loss function and instead maximize the F-score in terms of a expected utility function. We take the method proposed by Jansche [2005], that approximately maximizes the F-score of a classifier based on a logistical regression model, and apply it to our neural architectures. The F-score function can then be calculated in terms of a triple (A,B,C), as follows:

$$F_\alpha(A, B, C) = \frac{A}{A + \alpha B + (1 - \alpha)C} \quad (7.5)$$

This is equivalent to the  $\beta$ -weighted harmonic mean defined in Equation 5.1. Variable  $A$  is the number of true positives,  $B$  is the number of false negatives, and  $C$  is the number of false positives. Table 7.1 describes the relationship between the predictions and the actual labels.

		Predicted		Total
		pos	neg	
Actual	pos	A	B	$n_{pos}$
	neg	C	D	$n_{neg}$
Total		$m_{pos}$	$m_{neg}$	$n$

**Table 7.1.** Relationship between positive and negative matches.

That is,  $n_{pos}$  is the number of positive examples in the dataset (true positives plus missed positives) and  $m_{pos}$  is the number of positive examples that were predicted (true positives plus false positives). With this, we can rewrite  $F_\alpha(A, B, C)$  as:

$$F_\alpha(A, n_{pos}, m_{pos}) = \frac{A}{\alpha n_{pos} + (1 - \alpha)m_{pos}} \quad (7.6)$$

With this expression in mind, Jansche [2005] proposes the following optimization objective for the expected F-score:

$$\tilde{F}_\alpha(z, y) = \frac{\tilde{A}(z, y)}{\alpha \tilde{n}_{pos} + (1 - \alpha) \tilde{m}_{pos}(\theta)} \quad (7.7)$$

where  $y = \{y_1, y_2, \dots, y_n\}$  is a vector of actual labels,  $z = \{z_1, z_2, \dots, z_n\}$  is a vector of predictions, and:

$$\theta_i \equiv \hat{P}(z_i = True) \quad \forall i \in [1, n] \quad (7.8)$$

is the model probability that label  $z_i$  is a *True* label, assuming that we are dealing

with a binary classification problem with only *True* and *False* labels. Finally,

$$\begin{aligned}\tilde{A}(y, z) &\equiv \sum_{i=1}^n I_{y_i=z_i=1} \cdot \theta_i \\ \tilde{m}_{pos} &\equiv \sum_{i=1}^n \theta_i \\ \tilde{n}_{pos} &\equiv \sum_{i=1}^n I_{y_i=1} - \tilde{A}(y, z)\end{aligned}$$

where  $I_C$  is an identity function that is equal to one when the clause  $C$  is true and zero otherwise. With that, we have an optimization objective that can be optimized with standard numerical optimization methods. As a simplification, we consider that each label is independent and belongs to a different entity. Further work is needed to resolve this simplification and to adapt this optimization function to multi-label classification tasks. But, as we will see in the experiments section, this formulation is sufficient for the RNE task.

## 7.3 Experiments

The techniques proposed in this chapter are still very experimental, but we will show some preliminary results by posing some additional research questions:

1. Can the hard-attention and soft-attention mechanisms boost the performance of the Bi-LSTM-CRF?
2. Can we control how much the Bi-LSTM-CRF values recall over precision?
3. Can we improve the results of the best models with a simple filtering strategy?

In Section 7.3.1, we discuss the experiments concerning Research Question 1, in Section 7.3.1, we discuss the experiments concerning Research Question 2, and lastly, in Section , we discuss the experiments concerning Research Question 3.

### 7.3.1 Attention Mechanisms

The Self-Training strategy for HMMs showed that there is a lot to gain from incorporating features related to the HTML structure in our models. The HMM-2 model improved from a 0.805  $F_1$ -score to 0.879  $F_1$ -score with the self-training strategy. The

hard and soft attention mechanisms proposed in Section 7.1 are techniques for neural networks that attempt to bring improvements to the Bi-LSTM-CRF in a manner similar to the self-training strategy for HMMs. In Table 7.2, we compare the performance of the Bi-LSTM-CRF with LSTM-based character representations using a Hard Attention layer and a Soft Attention layer.

Model	Validation			Test		
	P	R	F1	P	R	F1
Bi-LSTM-CRF + LSTM <sub>c</sub>	0.928	0.950	0.939	0.920	0.886	0.902
+Hard Attention	0.944	0.961	0.952	<b>0.925</b>	<b>0.890</b>	<b>0.907</b>
+Soft Attention	0.894	0.961	0.926	0.884	0.849	0.866

**Table 7.2.** Hard and Soft Attention

The Hard Attention layer improved the original model by roughly 0.005 F1-score. However, the Soft Attention layer actually decreased the performance, what demands further explanation. The Hard Attention layer captures if the HTML context between two tags is exactly the same, but it does not look at the context itself. What the Hard Attention layer learns is essentially how much it can trust predictions made for other words in the same HTML context, no matter what is the actual context. However, the Soft Attention layer, also learns features about the HTML context, because it transforms the contextual representations with a feed forward neural network and then compares these neural representations. For example, the model could learn that tokens happening inside a list are more likely to be names, but since each document comes from a different website, this reasoning is unlikely to be correct for most cases. We cannot trust specific HTML configurations to be meaningful outside the document where they happen. This does not mean that the Soft Attention layer is useless, but for it to grasp abstract structural patterns that could be generally useful, we would need a much larger dataset. A possibility would be to pre-train unsupervised HTML contextual embeddings to detect useful HTML patterns in a large collection and then use this knowledge in a specific sequence labeling setting.

### 7.3.2 F-score Optimization

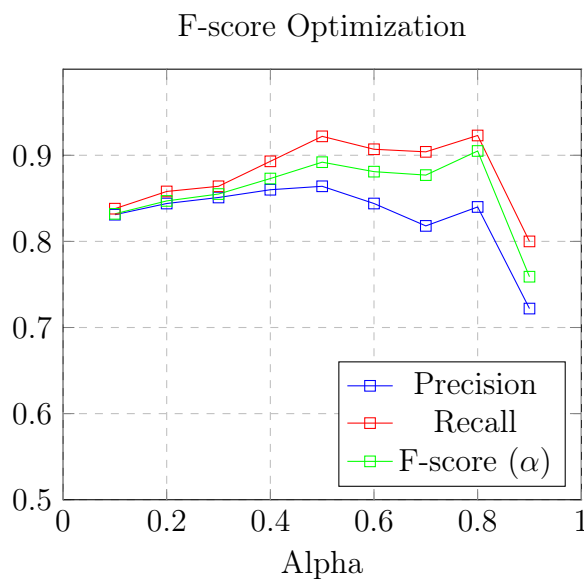
When training Deep Neural Networks for sequence labeling, we usually minimize the cross entropy. However, what we ultimately care about is the F-score. In Section 7.2, we proposed to change the optimization objective for Neural Networks and optimize



the expected F-score function to control how much our model prioritizes recall over precision.

The results presented in Chapter 6 show that Neural Architectures tend to value precision over recall in the name extraction task, however it is arguably better to improve recall and lose a little precision in this case, since it is easier to filter out false positives from the results than to find the false negatives manually.

Figure 7.3 presents the results on the test set for a Bi-LSTM-CRF model with LSTM character representations trained with the expected F-score optimization objective and varying the F-score  $\alpha$ . For example,  $\alpha = 0.5$  means that we value recall as much as precision,  $\alpha = 0.2$  means that we value precision twice as much as recall, and  $\alpha = 0.8$  means that we value recall twice as much as precision. Take note that setting  $\alpha = 0.5$  does not mean that we are using the same model that minimizes the cross-entropy, because the optimization objective is different. The exact formula for the  $F_\alpha$ -score was given in Equation 7.5. No attention layers were added to these models.



**Figure 7.3.** Results for the Bi-LSTM-CRF + LSTMc that optimized the expected  $F_\alpha$ -score function. A larger  $\alpha$  means preference for recall rather than precision.

This experiment shows that the capacity to control how much the model optimizes for one measure over the other is limited. When giving more priority to precision ( $\alpha = 0.8$ ), the model is still less precise than the variant that optimizes for the  $F_1$ -score ( $\alpha = 0.5$ ). Actually, the model that achieves top precision (0.864) and the second best recall (0.922) is the one with  $\alpha = 0.5$ . The model with  $\alpha = 0.8$  achieves the best recall

(0.923) and the best  $F_\alpha$  (0.905), though it should theoretically prioritize precision.

A useful result is that all of the models that optimized the expected F-score function valued recall over precision, contrasting with the Neural models that minimized the cross-entropy, which prioritized precision over recall. However, it is inconclusive if the direct optimization of the F-score function will lead to an improvement of recall over precision in other extraction tasks. The Bi-LSTM-CRF that optimized the expected  $F_1$ -score obtained a recall of 0.922 in the test set without losing a lot of precision (0.864), contrasting with the best model presented in the previous sections, the Hard-Attention Bi-LSTM-CRF, that achieved a 0.890 recall and 0.925 precision. Table 7.3 shows the detailed results for the models discussed in this Subsection.

$\alpha$	Validation			Test		
	P	R	$F_\alpha$	P	R	$F_\alpha$
0.1	0.879	0.925	0.884	0.831	0.838	0.832
0.2	0.876	0.939	0.888	0.844	0.858	0.847
0.3	0.881	0.935	0.896	0.851	0.864	0.855
0.4	0.873	0.962	0.907	0.860	0.893	0.873
0.5	0.823	0.970	0.890	<b>0.864</b>	0.922	0.892
0.6	0.839	0.976	0.916	0.844	0.907	0.881
0.7	0.810	0.975	0.919	0.818	0.904	0.877
0.8	0.814	0.980	0.942	0.840	<b>0.923</b>	<b>0.905</b>
0.9	0.680	0.900	0.871	0.636	0.752	0.738

**Table 7.3.** Bi-LSTM-CRF with LSTM character embeddings and F-score optimization objective.

### 7.3.3 Filtering False Positives

In this dissertation, we argue that recall is more valuable than precision in NER tasks on the Web. To verify this empirically, we consider the results of the best models presented in Table 6.12, the Hard Attention Bi-LSTM-CRF and the Bi-LSTM-CRF + LSTMc optimized for the F1-score, and apply a simple filter to the predicted labels in the test set. Despite their variations, all models tend to commit some similar prediction mistakes.

We tried a simple strategy to test how difficult it would be to filter out false positives from the results. The filtering strategy consists of automatically relabeling

tokens that were labeled as a person ("B-PER" or "I-PER") to an outside ("O") label if:

1. It was an honorific (Mr., Dr., P.hD., etc.).
2. It contained a number.
3. It was a punctuation sign just before or after a name (e.g. "- Mark" or "Emma ;").
4. It was an isolated name label.
5. It belonged to a name that was repeated at least three times.

The results of this simple filtering strategy is presented in Table 7.4. It improved the precision of all models considerably, showing that filtering out all false positives can probably be accomplished with only a small effort.

Model + Filter (F)	Precision	Recall	F1
HMM-2 (Group A) + Self-Training + F	0.9240	0.8795	0.9012
CRF (Group B) + F	0.9236	0.7951	0.8545
Bi-LSTM-CRF + LSTMc + F	<b>0.9387</b>	0.8887	0.9130
Bi-LSTM-CRF + LSTMc + Hard Attention + F	0.9376	0.8997	0.9183
Bi-LSTM-CRF + LSTMc + F1 Optimization + F	0.9239	<b>0.9409</b>	<b>0.9323</b>

**Table 7.4.** Overview of the best models for the name extraction task in the RNE test set after the filtering strategy.

Lastly, in Table 7.5 we present the training times and the subjective complexity for the best models in the RNE task. The subjective complexity tries to describe how much effort a researcher has to put in order to implement and run tests with each model, considering our particular experience in running the experiments for this dissertation. As we increase the model complexity, we can raise the  $F_1$  up to 0.9323 with the F1 optimized Bi-LSTM-CRF + LSTMc after filtering false positives, what leaves small room for further improvement. However, with a much simpler model (the self-trained HMM-2) we can get a 0.9012 F1 after filtering false positives, which is not significantly lower. These results are somewhat comparable to what has been happening with the reported results for the CoNLL-2003 English NER in the past few years (already discussed in Chapter 3). That is, as model complexity increases, the gains in terms of objective measures are not very substantial.

Model	F1	F1 (after filter)	Time (seconds)	Complexity
HMM-2 (Group A) + Self-Training	0.879	0.9012	64	Easy
CRF (Group B)	0.826	0.8545	965	Medium
Bi-LSTM-CRF + LSTMc	0.902	0.9130	4412	Hard
+Hard Attention	0.907	0.9183	6075	Very Hard
+F1 Optimization	0.892	0.9323	8065	Very Hard

**Table 7.5.** Overview of the subjective complexity and training times of the best models for the RNE task.

# Chapter 8

## Conclusions and Future Work

Existing WDE methods are useful for extracting simple entities from templated web-pages, but they do not handle cross website extraction tasks so well. Some techniques from machine learning such as NER are much more flexible, but the existing models are usually concerned with extraction tasks in plain text, so there is an absence of datasets for Web named entity extraction tasks. In this dissertation, we introduced a novel dataset that handled the researcher name extraction task, the RNE dataset, and discussed the applicability of different machine learning approaches to this problem.

We assessed the performance of different types of HMMs, CRFs and Neural Networks in the RNE task, considering their accuracy and complexity. We also proposed the self-training strategy for HMMs and the attentions models for Bi-LSTM-CRFs, inspired on WDE systems, to make use of HTML structural features and improve the model performances on the Web. Lastly, we proposed to use a expected utility function based on the F-score as an optimization objective for Neural Networks and prioritize recall over precision.

A natural extension of this work is to test the accuracy of the proposed models on other Web extraction tasks. Naturally, this would require access to labeled datasets that concern these other tasks.

### 8.1 Summary of Conclusions

In this dissertation, we learned that machine learning methods for NER are useful for solving some cross-website data extraction tasks and, particularly, the RNE task. We discussed the relative advantages and disadvantages of HMMs, CRFs, and Neural Networks. HMMs and CRFs can solve the RNE task with very good accuracy as long as we feed them with the correct features. Contrastingly, Neural Networks demand no

feature engineering and can obtain even better accuracy, however their complexity and training time is not entirely justifiable since the gains are not very substantial. In fact, Neural Networks cannot be considered entirely automatic, since we still have to tune hyper parameters and choose the right set of word embeddings.

Additionally, both traditional methods and neural networks can profit from making use of HTML structural features when used for sequence labeling in webpages. We proposed the self-training strategy and the attention models to incorporate this knowledge in HMMs and Neural Networks, respectively. And finally, we argued that we should value recall a little more than precision in Web extraction tasks, because filtering out false positives is easier than finding false negatives in the corpus. A way to incorporate this intuition on Neural Network is through the optimization of the expected F-score utility function. Also, we confirmed with the filtering strategy that filtering false negatives in the RNE task was indeed uncomplicated.

## 8.2 Future Work

This dissertation left many open ideas that could be explored in future research:

- **The self-training strategy** was very useful for improving the performance of HMMs, but variations of this strategy could also be incorporated in neural architectures. The hard-attention and soft-attention layers described in this dissertation were an early approach for doing that, but they require further improvement. To make soft-attention layers more reliable, we could pre-train HTML context embeddings in a large collection and derive structural patterns that can be useful in many sequence labeling tasks in HTML.
- **Good unsupervised methods** for WDE are necessary if we want to solve this task definitively. NER methods trained in plain text become ineffective when they are applied to documents of a different type, so we end up having to construct new datasets for each extraction task. This effort is very time-consuming, so we need more flexible and accurate approaches. The Baum-Welch algorithm is an unsupervised method for training HMMs, but it is not effective in NER. If we could devise an efficient way to train semi supervised neural networks especially for Web extraction tasks, we could make an end-to-end system that solves NER, relationship extraction and named entity linking in a single step.

### 8.3 Final Remarks

The preliminary results of some of the Hidden Markov Models discussed in this dissertation were published in Veneroso and Ribeiro-Neto [2018], though the results in this research article were not obtained with the exact same dataset. The other contributions in this dissertation generated other two research articles that are still being reviewed. The first article makes a comparison of the P-score measure and the H-index, that was made possible by the collection of affiliation information from university websites. The second article discusses the NER techniques proposed in this dissertation.

# Bibliography

- Abdessalem, T., Cautis, B., and Derouiche, N. (2010). ObjectRunner: Lightweight, targeted extraction and querying of structured web data. *Proc. VLDB Endow.*, 3(1-2):1585--1588. ISSN 2150-8097.
- Adelberg, B. (1998). NoDoSE—a tool for semi-automatically extracting structured and semistructured data from text documents. *ACM SIGMOD Record*, 27(2):283--294. ISSN 01635808.
- Akbik, A., Blythe, D., and Vollgraf, R. (2018). Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638--1649, Santa Fe, New Mexico, USA. Association for Computational Linguistics.
- Arocena, G. O. and Mendelzon, A. O. (1998). WebOQL: Restructuring documents, databases, and webs. In *Proceedings of the Fourteenth International Conference on Data Engineering, ICDE '98*, pages 24--33, Washington, DC, USA. IEEE Computer Society.
- Baevski, A., Edunov, S., Liu, Y., Zettlemoyer, L., and Auli, M. (2019). Cloze-driven pretraining of self-attention networks. *Preprint arXiv:1903.07785*.
- Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.
- Baum, L. E., Petrie, T., Soules, G., and Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164--171.
- Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1--127. ISSN 1935-8237.



- Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137--1155. ISSN 1532-4435.
- Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning Long-term Dependencies with Gradient Descent is Difficult. *Trans. Neur. Netw.*, 5(2):157--166. ISSN 1045-9227.
- Bikel, D. M., Schwartz, R., and Weischedel, R. M. (1999). An algorithm that learns what's in a name. *Machine Learning*, 34(1):211--231. ISSN 1573-0565.
- Borthwick, A., Sterling, J., Agichtein, E., and Grishman, R. (1998). Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In *Proceedings of the Sixth Workshop on Very Large Corpora*, pages 152--160. Association for Computational Linguistics.
- Brown, L. D. (1986). *Fundamentals of Statistical Exponential Families: With Applications in Statistical Decision Theory*. Institute of Mathematical Statistics, Hayworth, CA, USA. ISBN 0-940-60010-2.
- Califf, M. E. and Mooney, R. J. (1999). Relational learning of pattern-match rules for information extraction. *Computational Linguistics*, 4:9--15. ISSN 15324435.
- Caruana, R., Lawrence, S., and Giles, L. (2000). Overfitting in neural nets: Back-propagation, conjugate gradient, and early stopping. In *Proceedings of the 13th International Conference on Neural Information Processing Systems, NIPS'00*, pages 381--387, Cambridge, MA, USA. MIT Press.
- Chang, C., Chang, C., Lui, S., and Lui, S. (2001). IEPAD: information extraction based on pattern discovery. *Proceedings of the 10th international conference on World Wide Web*, pages 681--688.
- Chang, C.-H., Kaye, M., Girgis, M. R., and Shaalan, K. F. (2006). A Survey of Web Information Extraction Systems. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1411--1428. ISSN 1041-4347.
- Chang, C. H. and Kuo, S. C. (2004). OLERA: Semisupervised Web-data extraction with visual support. *IEEE Intelligent Systems*, 19(6):56--64. ISSN 15411672.
- Chinchor, N. (1998). Overview of MUC-7. In *Proceedings of the Seventh Message Understanding Conference, MUC-7*.
- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493--2537. ISSN 1532-4435.

- Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, 20(2):215--242. ISSN 00359246.
- Crescenzi, V., Mecca, G., and Merialdo, P. (2001). Roadrunner: Towards automatic data extraction from large web sites. *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 109--118. ISSN 10477349.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. *Preprint arXiv:1810.04805*.
- Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., and Zhang, W. (2014). Knowledge Vault: a web-scale approach to probabilistic knowledge fusion. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, pages 601--610. ISSN 0893-6080.
- Ferrara, E. and Baumgartner, R. (2011). Automatic wrapper adaptation by tree edit distance matching. *Smart Innovation, Systems and Technologies*, 8:41--54. ISSN 21903018.
- Ferrara, E., De Meo, P., Fiumara, G., and Baumgartner, R. (2014). Web data extraction, applications and techniques: A survey. *Knowledge-Based Systems*, 70:301--323. ISSN 09507051.
- Florian, R., Ittycheriah, A., Jing, H., and Zhang, T. (2003). Named entity recognition through classifier combination. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 168--171.
- Forney, G. D. (1973). The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268--278. ISSN 0018-9219.
- Freitag, D. (1998). Information Extraction from HTML: Application of a General Machine Learning Approach. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, AAAI '98/IAAI '98*, pages 517--523, Menlo Park, CA, USA. American Association for Artificial Intelligence.
- Freitag, D. and McCallum, A. (2000). Information Extraction with HMM Structures Learned by Stochastic Optimization. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 584--589. AAAI Press.

- Freitag, D. and Mccallum, A. K. (1999). Information Extraction with HMMs and Shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction*, pages 31--36. AAAI Press.
- Furche, T., Gottlob, G., Grasso, G., Orsi, G., Schallhart, C., and Wang, C. (2012). AMBER: Automatic Supervision for Multi-Attribute Extraction. *Preprint arXiv:1210.5984*.
- Grishman, R., Huttunen, S., and Yangarber, R. (2002). Information extraction for enhanced access to disease outbreak reports. *Journal of Biomedical Informatics*, 35(4):236 – 246. ISSN 1532-0464.
- Grishman, R. and Sundheim, B. (1996). 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. Association for Computational Linguistics.
- Hammer, J., McHugh, J., and Garcia-Molin, H. (1997). Semistructured Data: The TSIMMIS Experience. In *Proceedings of the First East-European Conference on Advances in Databases and Information Systems*, ADBIS'97, pages 22--22, Swindon, UK. BCS Learning & Development Ltd.
- Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527--1554. ISSN 0899-7667.
- Hirsch, J. E. (2005). An index to quantify an individual's scientific research output. *Proceedings of the National Academy of Sciences*, 102(46):16569--16572.
- Hirschman, L., Yeh, A. S., Blaschke, C., and Valencia, A. (2005). Overview of BioCre-AtIvE: critical assessment of information extraction for biology. *BMC Bioinformatics*, 6:S1.
- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735--1780. ISSN 0899-7667.
- Hogue, A. and Karger, D. (2005). Thresher: Automating the Unwrapping of Semantic Content from the World Wide Web. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 86--95, New York, NY, USA. ACM.

- Hsu, C. N. and Dung, M. T. (1998). Generating finite-state transducers for semi-structured data extraction from the Web. *Information Systems*, 23(8):521--538. ISSN 03064379.
- Huang, Z., Xu, W., and Yu, K. (2015). Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991.
- Jansche, M. (2005). Maximum Expected F-measure Training of Logistic Regression Models. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing, HLT '05*, pages 692--699, Stroudsburg, PA, USA. Association for Computational Linguistics.
- Jaynes, E. T. (1957). Information theory and statistical mechanics. *Physical Review*, 106:620--630.
- Krüpl, B., Herzog, M., and Gatterbauer, W. (2005). Using visual cues for extraction of tabular data from arbitrary HTML documents. *Special interest tracks and posters of the 14th international conference on World Wide Web - WWW '05*, pages 1000--1001.
- Kushmerick, N. (2000). Wrapper induction: efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15--68. ISSN 00043702.
- Laender, A. H. F., Ribeiro-Neto, B., and da Silva, A. S. (2002a). DEByE - Date extraction by example. *Data Knowledge Engineering*, 40(2):121--154. ISSN 0169-023X.
- Laender, A. H. F., Ribeiro-Neto, B. A., da Silva, A. S., and Teixeira, J. S. (2002b). A brief survey of web data extraction tools. *SIGMOD Record*, 31(2):84--93. ISSN 0163-5808.
- Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01*, pages 282--289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. *CoRR*, abs/1603.01360.
- Lawrence, S., Lee Giles, C., and Bollacker, K. (1999). Digital libraries and autonomous citation indexing. *Computer*, 32(6):67--71.

- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541--551. ISSN 0899-7667.
- Liu, L., Pu, C., and Han, W. (2000). XWRAP: an XML-enabled wrapper construction system for Web information sources. *Proceedings of 16th International Conference on Data Engineering*, pages 611--621. ISSN 1063-6382.
- Ma, X. and Hovy, E. (2016). End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064--1074, Berlin, Germany. Association for Computational Linguistics.
- Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA. ISBN 0-262-13360-1.
- McCallum, A. (2005). Information extraction: Distilling structured data from unstructured text. *Queue*, 3(9):4:48--4:57. ISSN 1542-7730.
- McCallum, A., Freitag, D., and Pereira, F. C. N. (2000). 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. Morgan Kaufmann Publishers Inc.
- McCallum, A. and Li, W. (2003). 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. Association for Computational Linguistics.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2, NIPS'13*, pages 3111--3119, USA. Curran Associates Inc.
- Muslea, I., Minton, S., and Knoblock, C. (1999). A Hierarchical Approach to Wrapper Induction. In *Proceedings of the Third Annual Conference on Autonomous Agents, AGENTS '99*, pages 190--197, New York, NY, USA. ACM.
- Ng, A. Y. and Jordan, M. I. (2001). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Proceedings of the 14th Interna-*

- tional Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pages 841--848, Cambridge, MA, USA. MIT Press.
- Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532--1543.
- Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227--2237, New Orleans, Louisiana. Association for Computational Linguistics.
- Rabiner, L. R. (1990). *Readings in Speech Recognition. Chapter: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1-55860-124-4.
- Ramshaw, L. A. and Marcus, M. P. (1999). *Text Chunking Using Transformation-Based Learning*, pages 157--176. Springer Netherlands, Dordrecht.
- Rastegar-Mojarad, M., Liu, S., Wang, Y., Afzal, N., Wang, L., Shen, F., Fu, S., and Liu, H. (2018). Biocreative/ohnlp challenge 2018. In *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, BCB '18, pages 575--575, New York, NY, USA. ACM.
- Ratnaparkhi, A. (1998). *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, Philadelphia, PA, USA. AAI9840230.
- Ribas, S., Ribeiro-Neto, B., de Souza e Silva, E., Ueda, A. H., and Ziviani, N. (2015). Using reference groups to assess academic productivity in computer science. In *Proceedings of the 24th International Conference on World Wide Web, WWW '15 Companion*, pages 603--608, New York, NY, USA. ACM.
- Rijsbergen, C. J. V. (1979). *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition. ISBN 0408709294.
- Sahuguet, A. and Azavant, F. (1999). Building Light-Weight Wrappers for Legacy Web Data-Sources Using W4F. In *Proceedings of the 25th International Conference on Very Large Data Bases, VLDB '99*, pages 738--741, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

- Sarawagi, S. (2008). Information extraction. *Foundations and Trends in Databases*, 1(3):261--377. ISSN 1931-7883.
- Schulz, A., Lässig, J., and Gaedke, M. (2016). Practical web data extraction: Are we there yet? — A short survey. *IEEE/WIC/ACM International Conference on Web Intelligence (WI), 2016*, pages 562---567.
- Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379--423.
- Shi, S., Liu, C., Shen, Y., Yuan, C., and Huang, Y. (2015). AutoRM: An effective approach for automatic Web data record mining. *Knowledge-Based Systems*, 89:314-331. ISSN 09507051.
- Soderland, S. (1999). Learning Information Extraction Rules for Semi-Structured and Free Text. *Machine Learning*, 34(1):233--272. ISSN 0885-6125.
- Sundheim, B. M. (1991). Overview of the third message understanding evaluation and conference. In *Proceedings of the 3rd Conference on Message Understanding, MUC3 '91*, pages 3--16, Stroudsburg, PA, USA. Association for Computational Linguistics.
- Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ICML'13*, pages 1139--1147. JMLR.org.
- Sutton, C. and McCallum, A. (2012). An Introduction to Conditional Random Fields. *Foundations and Trends in Machine Learning*, 4(4):267--373. ISSN 1935-8237.
- Tjong Kim Sang, E. F. and De Meulder, F. (2003). 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, CONLL '03*, pages 142--147, Stroudsburg, PA, USA. Association for Computational Linguistics.
- Varlamov, M. I. and Turdakov, D. Y. (2016). A survey of methods for the extraction of information from Web resources. *Programming and Computer Software*, 42(5):279--291. ISSN 0361-7688.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, pages 5998--6008. Curran Associates, Inc.

- Veneroso, J. M. F. and Ribeiro-Neto, B. (2018). Entity name extraction from faculty directories. In *Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries*, JCDL '18, pages 389--390, New York, NY, USA. ACM.
- Werbos, P. (1990). Backpropagation through time: what does it do and how to do it. In *Proceedings of IEEE*, volume 78, pages 1550--1560.
- Zhu, J., Nie, Z., Wen, J.-R., Zhang, B., and Ma, W.-Y. (2005). 2D Conditional Random Fields for Web Information Extraction. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 1044--1051, New York, NY, USA. ACM.
- Zhu, J., Nie, Z., Wen, J.-R., Zhang, B., and Ma, W.-Y. (2006). Simultaneous Record Detection and Attribute Labeling in Web Data Extraction. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 494--503, New York, NY, USA. ACM.



# Appendix A

## Cross Validation

Considering the reduced size of the RNE dataset, we must acknowledge the limitations concerning generalizations about the relative performance of the models studied in this dissertation. The naive division of the labeled data into three static data files (training, validation and test) may harm the validity of our analysis if the data files were randomly selected in a way that favors a model in place of others due to the natural volatility in the process of parameter estimation.

When training a machine learning model for a particular prediction task, the model is expected to fit the training data as closely as possible as the training progresses. If the training is kept going indefinitely, the model will probably learn spurious patterns (i.e. overfitting the training data), what will lead to poor generalizations. That is why the expected performance of a predictive model over unseen data must be checked with the aid of an independent sample drawn from the same population as the training data, which we called the test set in previous chapters. In addition to the test set, we also separated a validation data file. The validation data file is never observed directly by the models. Rather, its purpose is to prevent overfitting by monitoring the model performance in the validation dataset and stop the training when a given statistic, in this case the F1-score, decreases over several epochs. This is called the early stopping strategy [Caruana et al., 2000].

The problem with splitting the dataset in three files is that the independent sample used to test the models may not be a faithful representation of the complete population. This problem is less likely to lead to wrong conclusions if the dataset is sizable, but in the case of the RNE dataset, the risk of incorrect results increases due to the diminished size of the dataset. We can mitigate this problem by employing the strategy of K-fold cross-validation. K-fold cross-validation consists in splitting the complete dataset into K folds, where  $(K - 1)$  folds are used to train the model and

one fold is used to test it. The test fold is rotated  $K$  times until we have trained  $K$  variants of the same model and tested it in each fold separately. The averaged statistics obtained in this process will be more trustworthy than the results obtained with the static split since each model variation will have been tested against all data points.

In this appendix, we discuss the results for the models presented in the preceding chapters by performing five fold cross validation in a combination of the training, validation and test data files of the RNE dataset and discussing the results. The results for HMMs are presented in Section A.1, the results for CRFs are presented in Section A.2 and the results for Neural Networks are presented in Section A.3.

## A.1 Hidden Markov Models

Table A.1 presents the results for all HMM variations when performing five fold cross validation in the RNE dataset. The F1 obtained only in the test set is also presented for ease of comparison. The F1 (test set) is the same one presented in the earlier chapters. Each cell shows the precision, recall and F1 statistics averaged over five folds and their standard deviations inside parenthesis.

The results for the models that employed only the current word as a feature did not change dramatically, except for the HMM-3, which had a much better average result in the cross validation runs. An important aspect revealed by cross validation is that most tested HMMs showed a significant F1 standard deviation. For example, the HMM-2 with the self-training strategy showed a 0.099 standard deviation in the F1 across five folds. This important information poses some doubt over some earlier conclusions. In fact, models that employed Group B features<sup>1</sup> performed better than models with Group A features<sup>2</sup> contrasting with the results for the models tested using only the test datafile. Because of the high standard deviation, it is difficult to ascertain the relative superiority of different feature selections and different types of HMMs. But, it is possible to notice that the addition of more features contributes a lot to the reduction of the standard deviations since models trained with features from Group B presented consistently less variation in performance. The self-training strategy was still useful in improving all models with a substantial margin. Overall, the best HMMs were able to achieve an F1 around 0.8 in the RNE dataset.

---

<sup>1</sup> All features except for the HTML Tag and the CSS Class.

<sup>2</sup> Current word, Exact Match, Partial Match, URL, Capitalized

Model	P	R	F1 (CV)	F1 (Test)
NB	0.082 (0.006)	0.072 (0.004)	0.077 (0.005)	0.086
HMM-1	0.631 (0.066)	0.516 (0.097)	0.566 (0.083)	0.533
HMM-2	0.606 (0.141)	0.552 (0.114)	0.577 (0.126)	0.583
HMM-3	0.599 (0.111)	0.609 (0.114)	0.603 (0.105)	0.506
HMM-1 (Group A)	0.773 (0.094)	0.729 (0.055)	0.750 (0.073)	0.805
HMM-2 (Group A)	0.762 (0.085)	0.749 (0.057)	0.754 (0.062)	0.812
HMM-3 (Group A)	0.777 (0.073)	0.763 (0.061)	0.769 (0.065)	0.798
HMM-1 (Group B)	0.811 (0.048)	0.793 (0.030)	0.801 (0.032)	0.748
HMM-2 (Group B)	0.779 (0.042)	0.781 (0.082)	0.779 (0.055)	0.769
HMM-3 (Group B)	0.771 (0.075)	0.787 (0.042)	0.778 (0.055)	0.724
HMM-1 (Group A) + ST	0.745 (0.060)	0.787 (0.093)	0.764 (0.069)	0.865
HMM-2 (Group A) + ST	0.771 (0.123)	0.816 (0.082)	0.791 (0.099)	0.879
HMM-3 (Group A) + ST	0.772 (0.098)	0.802 (0.072)	0.786 (0.085)	0.762
HMM-1 (Group B) + ST	0.812 (0.041)	0.828 (0.028)	0.819 (0.018)	0.866
HMM-2 (Group B) + ST	0.783 (0.081)	0.836 (0.033)	0.807 (0.052)	0.877
HMM-3 (Group B) + ST	0.774 (0.073)	0.821 (0.044)	0.796 (0.053)	0.814

**Table A.1.** Precision (P), Recall (R) and F1 for Hidden Markov Models and Naive Bayes in the RNE task using five fold cross validation (CV).

## A.2 Conditional Random Fields

Table A.2 presents the results for all CRF variations when performing five fold cross validation in the RNE dataset. Each cell shows the precision, recall and F1 statistics averaged over five folds and their standard deviations in parenthesis.

All CRF variations performed worse when considering the average results across five folds in comparison to the models tested using only the test set. The variant using Group B features still achieved the top performance between the three variants. As was the case for the HMMs, CRFs show a significant standard deviation, what prevents us from drawing strong conclusions about the relative performances of different model variations.

## A.3 Neural Networks

Table A.3 presents the results for all Neural Network variations when performing five fold cross validation in the RNE dataset. Each cell shows the precision, recall and F1

Model	P	R	F1 (CV)	F1 (Test)
Logistic Classifier	0.095 (0.027)	0.122 (0.027)	0.104 (0.018)	0.133
CRF	0.685 (0.086)	0.680 (0.071)	0.677 (0.031)	0.750
CRF (Group A)	0.794 (0.074)	0.742 (0.081)	0.767 (0.075)	0.808
CRF (Group B)	0.802 (0.068)	0.753 (0.074)	0.776 (0.067)	0.826

**Table A.2.** Precision (P), Recall (R) and F1 for CRFs and Logistic Classifier in the RNE task using five fold cross validation (CV).

statistics averaged over five folds and their standard deviations. All neural networks were trained over ten epochs in each fold. No early stopping strategy was used.

Almost all neural network variations showed about the same average performance using either only the test set or cross validation. However, it is noteworthy to mention that the choice of the test set hindered the performance of ELMo embeddings quite significantly. Overall, ELMo embeddings showed better performance than Word2Vec embeddings but despite adding a lot of complexity to the models, they still performed suboptimally when compared to static GloVe embeddings.

The Hard Attention model had the best average F1 among all neural network variations either in the test set or the cross validation runs. The soft attention layer was detrimental to the performance of the Bi-LSTM-CRF with LSTM characters on both experiments.

When compared to simpler statistical models, neural networks show a lower standard deviation. Still, when considering these ranges of variation, the introduction of Neural Networks in the RNE task does not provide a massive gain in performance. However, this gain is more significant than previously stated when we compared only the performances in the test set. It could be argued that the main benefits of employing Bi-LSTM-CRFs in this task, are the improved predictability provided by a smaller standard deviation and a better overall performance.

### A.3.1 F-score Optimization

In Table A.4, we present the cross-validation results for the Bi-LSTM-CRF with an  $F-\alpha$  optimization objective, where  $\alpha$  ranges from 0.1 to 0.9. The previous  $F-\alpha$  scores obtained using only the test set are also presented in the table.

The cross validation results for the Bi-LSTM-CRFs that optimized the  $F-\alpha$  objective did not change tremendously when compared to the results using only the test set. All variants privileged recall over precision and the change in the value of  $\alpha$  did

Model	P	R	F1 (CV)	F1 (Test)
Bi-LSTM-CRF	0.866 (0.036)	0.844 (0.043)	0.855 (0.036)	0.886
+CNNc	0.885 (0.050)	0.925 (0.023)	0.904 (0.034)	0.901
+LSTMc	0.889 (0.051)	0.924 (0.029)	0.906 (0.036)	0.902
+LSTMc (Word2Vec)	0.870 (0.056)	0.898 (0.047)	0.883 (0.044)	0.864
+LSTMc (Elmo)	0.911 (0.044)	0.891 (0.036)	0.901 (0.038)	0.841
+Hard Attention	0.905 (0.032)	0.912 (0.032)	0.908 (0.019)	0.907
+Soft Attention	0.875 (0.051)	0.924 (0.039)	0.899 (0.044)	0.865

**Table A.3.** Precision(P), Recall (R) and F1 for Bi-LSTM-CRF variations in the RNE task using five fold cross validation (CV). All the models used GloVe-300 embeddings except if stated otherwise.

not impact the end result significantly. The best F-score ( $\alpha$ ) was achieved by the variant that optimized the F-score with  $\alpha = 0.3$  in contrast with the runs in the test set, where the best model was the network that optimized the F1-score. It is noteworthy to mention that, different from the results obtained using only the test set, most Neural Network variants showed a preference for Recall over Precision in the cross validation results even when optimizing the regular cross entropy function, what weakens the case for a custom optimization objective such as the F-score utility function.

$\alpha$	P	R	F- $\alpha$ (CV)	F- $\alpha$ (Test)
0.1	0.832 (0.055)	0.871 (0.048)	0.867 (0.048)	0.832
0.2	0.837 (0.047)	0.901 (0.052)	0.887 (0.048)	0.863
0.3	0.842 (0.049)	0.911 (0.053)	0.888 (0.039)	0.873
0.4	0.836 (0.068)	0.929 (0.037)	0.887 (0.023)	0.881
0.5	0.807 (0.033)	0.904 (0.036)	0.853 (0.031)	0.881
0.6	0.818 (0.050)	0.926 (0.065)	0.857 (0.053)	0.855
0.7	0.798 (0.090)	0.900 (0.037)	0.825 (0.077)	0.877
0.8	0.821 (0.044)	0.926 (0.029)	0.840 (0.041)	0.850
0.9	0.644 (0.320)	0.808 (0.254)	0.655 (0.321)	0.734

**Table A.4.** Precision(P), Recall (R) and F1 for Bi-LSTM-CRF variations optimizing the F- $\alpha$  objective in the RNE task using five fold cross validation (CV).