UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ALAN PINTO SOUZA

Metadata extraction from Scientific Documents in PDF

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Prof. Dr. Carlos Alberto Heuser
Advisor

Prof. Dra. Viviane Moreira
Coadvisor

Porto Alegre, June 2014

“Facts are the air of scientists.
Without them you can never fly.”
— LINUS PAULING

CIP – CATALOGING-IN-PUBLICATION

Souza, Alan Pinto

Metadata extraction from Scientific Documents in PDF / Alan Pinto Souza. – Porto Alegre: PPGC da UFRGS, 2014.

59 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2014. Advisor: Carlos Alberto Heuser; Coadvisor: Viviane Moreira.

1. Metadata Extraction. 2. PDF. 3. Machine Learning. I. Heuser, Carlos Alberto. II. Moreira, Viviane. III. Título.

AGRADECIMENTOS

Para explicar os meus agradecimentos, eu preciso contar um pouco da minha trajetória. Fazer o mestrado em uma Universidade Federal nunca esteve em meus planos. Não que me faltasse vontade, mas eu nunca achei que tivesse competência suficiente para ser aprovado. Eu digo isso devido ao fato que durante boa parte da minha graduação eu estudei em uma Faculdade em que, sinceramente, não me orgulho da forma com que me dediquei a mesma. Eu tinha que dividir o tempo de estudo com a minha profissão de músico, o que fazia com que eu tivesse que faltar muitas aulas para viajar a trabalho. Depois que eu me formei, decidi que era necessário ingressar em um Mestrado para consertar todos os erros que eu cometi ao decorrer da minha Graduação. Logo, durante um ano, eu estudei por conta própria todas as matérias da Graduação novamente. Contei com a ajuda de um professor particular de Matemática (Gustavo Viegas), pessoa a quem dedico meu primeiro agradecimento. Fica aqui o meu reconhecimento pelo excelente trabalho que o Gustavo prestou durante os nossos dois meses de encontros.

Faltando apenas dois dias para acabarem as incrições para o Mestrado da UFRGS eu decidir que iria participar. Fica aqui meu muito obrigado a Adolfo Duran e Edeyson Gomes pelas cartas de recomendação. Também, gostaria de agradecer aos meus orientadores Carlos Heuser e Viviane Moreira por terem apostado em meu potencial acadêmico, mesmo sem terem um histórico relevante da minha vida como estudante. Ao Heuser, gostaria de agradecer ao desafio que me foi dado de ser, provavelmente, o teu último mestrando antes da tua aposentadoria. À Viviane, por ter sido muito mais que uma co-orientadora e contribuindo de igual importância para a qualidade do nosso trabalho. Gostaria também de agradecer a todos os professores que eu tive durante o primeiro ano de aulas. Em especial, fica o meu muito obrigado a Aline Villavicencio, Renata Galante, Markus Ritt e Karin Becker. Eu aprendi muito com vocês e a sabedoria de todos eu quero carregar para sempre comigo.

Gostaria de agradecer a todos os meus colegas de laboratório. Em especial, fica o meu muito obrigado a Solange Pertile, Matheus Cadori, Marcelo Caggiani, Diego Tumitan e Bruno Laranjeira por terem me ajudado durante todo o meu mestrado. Também, gostaria de lembrar todo o suporte que vocês me deram para que eu conseguisse ser aprovado na matéria de Complexidade de Algoritmos com o professor Markus Ritt. Com certeza essa foi uma das matérias mais interessantes e complexas na qual eu participei e, sem o conhecimento de vocês, tudo teria sido mais difícil.

Eu gostaria de agradecer aos meus colegas de trabalho que me incentivaram para a inscrição na UFRGS e me apoiaram na realização desse sonho. Em especial agradeço a Marlon Parizzotto, Tales Chaves, Karina Kohl, Leonardo Tavares, Cristiano Galina, Joan Bernardes, Guilherme Rotta, Sagiane D'Avila, Farlon Souto e Leandro Farinati.

Eu gostaria de agradecer a minha mulher Driele Sanches que me incentivou 100% do

tempo durante esses 3 anos. Obrigado pela paciência e suporte nos momentos em que a minha atenção ficou comprometida por conta das incontáveis horas de estudo que eu precisei dedicar ao Mestrado.

Por fim, gostaria de agradecer aos meus familiares por terem investido na minha educação e por terem me apoiado sempre. O meu muito obrigado a minha mãe Maria Cristina, ao meu pai Sérgio Souza, e a minha irmã Aline Naiana.

ABSTRACT

Most scientific articles are available in PDF format. The PDF standard allows the generation of metadata that is included within the document. However, many authors do not define this information, making this feature unreliable or incomplete. This fact has been motivating research which aims to extract metadata automatically. Automatic metadata extraction has been identified as one of the most challenging tasks in document engineering. This work proposes Artic, a method for metadata extraction from scientific papers which employs a two-layer probabilistic framework based on Conditional Random Fields. The first layer aims at identifying the main sections with metadata information, and the second layer finds, for each section, the corresponding metadata. Given a PDF file containing a scientific paper, Artic extracts the title, author names, emails, affiliations, and venue information. We report on experiments using 100 real papers from a variety of publishers. Our results outperformed the state-of-the-art system used as the baseline, achieving a precision of over 99%.

Keywords: Metadata Extraction, PDF, Machine Learning.

Extração de Metadados em Artigos Científicos no Formato PDF

RESUMO

A maioria dos artigos científicos estão disponíveis no formato PDF. Este padrão permite a geracão de metadados que são inclusos dentro do documento. Porém, muitos autores não definem esta informação, fazendo esse recurso inseguro ou incompleto. Este fato tem motivado pesquisa que busca extrair metadados automaticamente. A extração automática de metadados foi classificada como uma das tarefas mais desafiadoras na área de engenharia de documentos. Este trabalho propõe Artic, um método para extração de metadados de artigos científicos que aplica um modelo probabilístico em duas camadas baseado em Conditional Random Fields. A primeira camada visa identificar as secões principais com possíveis metadados. Já a segunda camada identifica, para cada secão, o metadado correspondente. Dado um PDF contendo um artigo científico, Artic extrai título, nome dos autores, emails, afiliações e informações sobre a conferência onde o paper foi publicado. Os experimentos usaram 100 artigos de conferências variadas. Os resultados superaram a solução estado-da-arte usada como baseline, atingindo uma precisão acima de 99%.

Palavras-chave: Extração de Metadados, PDF, Aprendizagem de Máquina.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS AND ACRONYMS

HMM	Hidden Markov Models
CRF	Conditional Random Fields
PDF	Portable Document Format
IDM	Independent Document Model
OCR	Optical Character Recognition
POS	Part of Speech
DP	Dynamic Programming
ML	Machine Learning
JSON	JavaScript Object Notation

CONTENTS

1 INTRODUCTION

The metadata of a document are all the information describing the document itself. In scientific articles, this data usually includes: title, author, affiliation, date of publication, place of publication, etc. Collecting metadata is a crucial step for assembling a document repository, which in turn is very important in the document engineering area.

The Portable Document Format (PDF) is a file format that was created with the initial goal of being independent of application, hardware, and operating system (ROSENTHOL, 2013). This format was developed in the '90s and is widely used in the scientific literature as the standard format for publications. PDF allows the generation of the aforementioned metadata which is directly included within the document. Therefore, it is not necessary to use an additional file containing the metadata for PDF-based articles.

With the popularization of Internet, many scientific articles have been made available on the Web. While in the beginning, the articles were scanned and provided as image-based PDFs, more recently, documents are directly created as text-based PDFs. The biggest limitation is the lack of metadata or, even when present, it does not provide complete and reliable information. As a result, many information retrieval and document engineering systems have difficulties in indexing these files. These facts have been motivating research that aims at automatically identifying metadata. COUNCILL; GILES; KAN (2008) rated automatic metadata extraction as one of the most difficult tasks in document engineering. Research in this topic typically applies one of these three methods: template matching, web-based lookup (knowledge base), and machine learning (ML) techniques. A comparison of these methods with an evaluation of existing tools is presented in (LIPINSKI et al., 2013).

Recent techniques are increasingly using ML algorithms to try to achieve better results. They usually deal with the problem of metadata extraction as a sequence labeling task. In (SEYMORE; MCCALLUM; ROSENFELD, 1999; YIN et al., 2004), the authors address the metadata extraction problem using Hidden Markov Models (HMM). LUONG; NGUYEN; KAN (2010) created SectLabel, a metadata extraction tool that defines a single CRF model to identify 23 different classes, such as: address, affiliation, author, email, equation, figure, title, etc. SectLabel defines a set of features that allows the CRF model to identify each of the aforementioned classes. Having a single layer model may affect the metadata identification process since all features are naturally generic.

In this work we propose Artic, a two-layer CRF model that allows features to be metadata-specific. The first layer aims at identifying the main sections that may contain metadata information. For each of the given sections, a second layer will extract the desired metadata with a more granular level. Experiments yielded an overall precision of 99.84%, which represents a F1 improvement of 6.92% compared to the state-of-the-art baseline. We summarize our contributions as follows:

- Artic employs a two-layer CRF model. We believe that having an additional layer will improve the metadata extraction process as it allows the use of line-level features (first layer) and word-level features (second layer).
- Artic is able to identify the relationship between authors, emails, and affiliations. This functionality is not provided by the classification model, which is limited to identify the classes only (e.g Author Name, Affiliation).
- As opposed to a single classification tool, Artic provides the metadata output in a well-defined format (JSON). This allows anyone to use the metadata as it is, without the need for extra components to organize the classification results.

The remainder of this work is organized as follows. Chapter 2 gives required background to understand the techniques for metadata extraction. Chapter 3 explains state-of-the-art algorithms proposed for each of the tree areas of information extraction. Chapter 4 explains Artic, the two-layer CRF approach proposed in this work. Chapter 5 evaluates Artic against a baseline and the expected JSON output. Chapter 6 concludes the work with a summary of the contributions, future works and results.

2 BASIC CONCEPTS

This chapter introduces general concepts required to fully understand the techniques for automatic document metadata extraction.

2.1 Document Metadata

A document metadata is a component that describes information about the document itself. The term metadata is usually referred as "data about data". This assumption is ambiguous due to the fact that metadata can also describe structural components. Structural metadata defines the design and specification about data structures, leading to a non-ambiguous description of "data about the structure of the data". On the other hand, a descriptive metadata, also known as metacontent, describes data content (instances) of the application data itself, with the non-ambiguous description of "data about data content". The concept of document metadata, in the context of this work, is equivalent to metacontent. Instances of document metadata are: title, author, affiliation and email (BRETHERTON; SINGLEY, 1994).

The metadata was motivated by the need to manage exponentially increasing streams of data from automated systems, which also required interdisciplinary collaboration and sharing. For example, a book stored in a digital library system usually contains metadata such as: title, author, year of publication, ISBN, location in the library, etc. To better enable the user to find the book, searches are usually performed over the metadata contents. As a result, the book itself remains unchanged and is not required for the search process. Now, suppose that the library wants to share all their books with another library for a collaboration strategy. If the library has been built using metadata, a simple query over all the books metadata entities would be enough. Hence, metadata is the key component to ensure that a resource will be easily accessible in the future.

Metadata can be embedded in the document or it can be stored separately. The advantages of storing metadata with the object it describes are: ensure that it will not be lost, ensure that the document and the metadata will be updated together, and eliminate the problem of linking between data and metadata (PRESS, 2004). However, some specific types of documents do not allow embedded metadata. Additionally, storing metadata separately can simplify metadata management, as it will not need to access the document to retrieve the desired information. Hence, the most common approach is to store the metadata in a separate database which is usually referred as metadata repository or metadata registry.

2.2 Dynamic Programming

Dynamic Programming (DP) is a powerful algorithm technique that implicitly explores the space of all possible solutions by carefully decomposing bigger problems into a series of sub-problems. Then, it starts building-up correct solutions to larger and larger problems (KLEINBERG; TARDOS, 2006). It is drawn from the intuition behind divide and conquer and it is essentially the opposite of the greedy strategy. DP follows the principle of optimality which says that the optimal value of a problem can be obtained by the optimal values of its subproblems. The core feature of DP is the repeated usage of pre-computed values.

One very useful problem that can be solved by DP is LCS (Longest Common Subsequence). Given two distinct sequence of characters, such that, $X = x_1, x_2, ..., x_n$ and $Y = y_1, y_2, ..., y_m$, where n is the total number of characters from the sequence X and m is the total number of characters from the sequence Y. We need to find the longest subsequence length of commons characters in-between X and Y. For example, the LCS solution for "Stable" and "Table" is 5 and the actual sequence is "table". As we have just explained, to solve this problem using DP we need to explore the space of all possible solutions. Two extra indexes (i, j) are required to specify where the start of the common character is in X and Y, respectively ($x_1, x_2, ..., x_i, ..., x_n$ and $y_1, y_2, ..., y_j, ..., y_n$). In this case x_i represents the start common character index in sequence X and, analogously, y_j represents the start common character index in sequence Y. Said that, there are four possible solutions:

1. if $i < n$ and $j < m \rightarrow x_i = x_{n-1}$ and $y_j = y_{m-1}$
2. if $i = n$ and $j < m \rightarrow x_i = x_n$ and $y_j = y_{m-1}$
3. if $i < n$ and $j = m \rightarrow x_i = x_{n-1}$ and $y_j = y_m$
4. if $i = n$ and $j = m \rightarrow x_i = x_n$ and $y_j = y_m$

We can easily join items 1 and 4, as $n = m$ and $n - 1 = m - 1$ holds. The optimal function $LCS(i, j) = max(LCS(i, j-1), LCS(i-1, j), LCS(i-1, j-1) + [x_i = y_j])$ is the function that returns the optimal length of the LCS with $x_1 = i$ and $y_1 = j$. At each iteration of LCS recursion, we get the maximum value from $LCS(i, j-1)$ or $LCS(i-1, j)$ or $LCS(i-1, j-1)$. Each recursion call represents the three subproblems we've raised above (joining items 1 and 4). The $[x_i = y_j]$ adds one when $x_i = y_j$. The complexity of this solution is $O(nm)$, where n is the number of characters of X and m is the number of characters of Y.

2.3 Hidden Markov Models

Hidden Markov Model (HMM) is a technique used for solving sequence labeling problems. The task of document metadata extraction can be considered as an instance of this problem (HAN et al., 2003). Hence, some proposals evaluated the use of HMM to automatically extract metadata information from any sort of document (HETZNER, 2008; SCHEFFER; DECOMAIN; WROBEL, 2001; SCHEFFER et al., 2002).

2.3.1 Theoretical Foundations

Natural events usually do not have a standard behavior, but they present random outputs thus not allowing trivial predictions. On the other hand, certain regularities can be defined even for random processes. The probability theory defines mathematical techniques to model regularities in random processes. Mathematical statistics considers the problem of how the parameters of probabilistic models can be derived from observations. It is important to emphasize that regularities can only be derived when considering long-term observations (FINK, 2007).

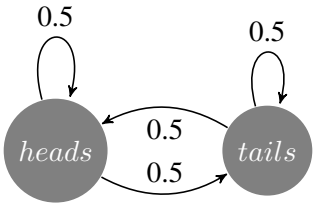
A Markov chain (also referred as n-gram model) is used especially for the statistical modeling of chronologically organized data (FINK, 2007; SKULJ, 2009). There are two types of Markov chains: discrete and continuous. A discrete Markov chain has a finite or countable set of states. A continuous Markov chain considers time or the space continuous and is also called continuous-time Markov process. In the context of this work, only discrete Markov chains will be considered. For the sake of simplicity, further references of this model will omit the "discrete" prefix.

Markov chains model the state transitions of random processes where the next stage depends exclusively on the current stage. In other words, given the present, the future is independent of the past. This "memoryless" feature is known as Markov property. A random process defines a system which is in a certain stage at some time (t), and the state transition occurs randomly. The probability associated with each state transition is formally defined below:

$$Pr(X_{n+1} = x | X_1 = x_1, X_2 = x_2, ..., X_n = x_n) = Pr(X_{n+1} = x | X_n = x_n)$$

Markov chains can be easily represented as a directed graph, where the edges are labeled with the transition probabilities and the nodes are all possible states. Figure 2.1 defines the Markov chain for a "fair coin" tossing experiment. The transition probabilities are all 0.5 (50%), hence with no bias of any sort.

Figure 2.1: The Markov chain for the fair coin model.

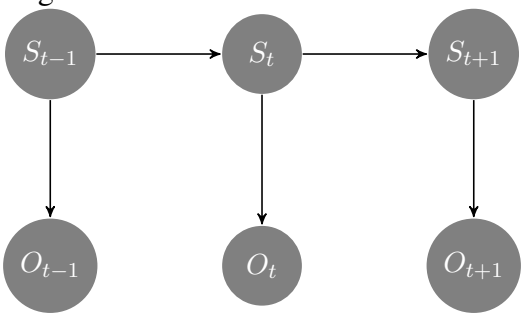


There are some fundamental problems with Markov chains which have motivated the emergence of a more robust model. One limitation is that the majority of the real-world problems cannot expect to perfectly observe the complete true state of a system. The evolvement started with the perception that there are some hidden information that are not being observed. Hence, the idea is to break up the state of the system into observed states and hidden (latent) states.

A Hidden Markov Model (HMM) is a two-stage statistical Markov model. The first state is a discrete random process with a finite state-space, exactly as the Markov chain model. In the second stage, for each instant of time (t) an emission O_t is generated (FINK, 2007; RABINER, 1989; RABINER; JUANG, 1986). The probability distribution for the emissions is dependent only on the current state S_t and not on previous states or emissions. Figure 2.2 graphically shows the evolution of a Hidden Markov Model. The

edge from S_{t-1} to S_t is the same transition probability described for the Markov chain. And, the edge from S_t to O_t represents the emission probability. From this diagram, it is clear that the transition probability of the hidden state S_t at time t, given the values of the hidden state S at all times, only depends on the value of the hidden variable S_{t-1}: the values at time $t-2$ and before have no influence (Markov property). Additionally, the observation value O_t only depends on the value of the hidden state S_t (both at time t).

Figure 2.2: The Hidden Markov Model.



The formal definition of the emission probability is given below:

$$P(O_t|O_1, O_2, ..., O_{t-1}, S_1, S_2, ..., S_t) = P(O_t|S_t)$$

A HMM has three required parameters. The first is the emission probabilities, which is the only observable entity of a HMM. The second parameter is the transition probabilities which remain "hidden" (main reason behind the term *Hidden* Markov Model derivation). Finally, there should exist the start probabilities for the model initialization ($t = 1$). In order to define the values of these three parameters, a manual analysis can be performed over the observations, which is time-consuming and impracticable for large datasets. Hence, there should be an algorithm which automatically computes the emission, transition, and initial set probabilities. A widely used technique to solve this problem is the Baul-Welch algorithm (BAGGENSTOSS, 2001; FINK, 2007).

Given the formal definition of a Hidden Markov Model as $\lambda = (O, S, \pi)$, where O is the observation sequence, S is the sequence of states, and π is the initial state probabilities ($t = 1$), the practical usage of λ is to find a state sequence I so that $P(x|I, \lambda)$ is maximized, where x is the observation sequence. A very common solution is to use the Viterbi algorithm, which applies a dynamic programming approach to solve this sequence labeling problem (FINK, 2007; FORNEY G.D., 1973).

2.3.2 The Weather Example

Consider two remote friends (Paul and Mike) which only communicate by chat. They usually talk in a daily basis about what they did that day. Suppose that Mike is only interested in performing three activities: going to the beach, shopping, and cleaning the house. Mike's decision on what to do depends exclusively on the weather on that given day. Paul has no clear information about the weather where Mike lives, but he knows some general tendency. Paul is trying to guess the weather conditions based on what Mike tells him about his activities. This problem can be modeled as a Hidden Markov Model.

Assuming the weather operates as a discrete Markov chain, there are two possible states: Rainy and Sunny. As Paul has no clear definition about the weather, its conditions

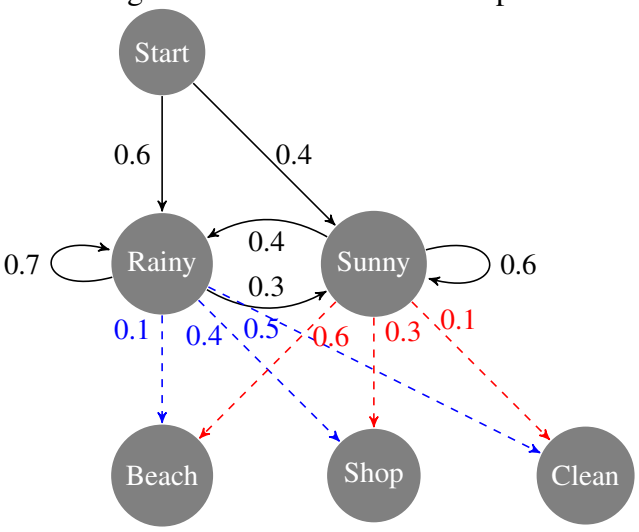
(states) are completely hidden from him. Since Mike is telling Paul about his daily activities, those are the observations. Paul knows the general weather trends, and what Mike likes to do on average. In other words, the parameters of the HMM are known. As a result, we can represent this problem using the formulation defined in the previous section. The required parameters, transition probabilities (Pr) and emission probabilities (P) are given below:

$$\lambda = (O, S, \pi)$$
$$O = (Beach, Shop, Clean)$$
$$S = (Rainy, Sunny)$$
$$\pi = (Rainy : 0.6, Sunny : 0.4)$$

$$Pr = (Rainy : \{Rainy : 0.7, Sunny : 0.3\},$$
$$Sunny : \{Rainy : 0.4, Sunny : 0.6\})$$
$$P = (Rainy : \{Beach : 0.1, Shop : 0.4, Clean : 0.5\},$$
$$Sunny : \{Beach : 0.6, Shop : 0.3, Clean : 0.1\})$$

In the above definition, π represents Paul's belief about the initial state of the system. All he knows is that it tends to be rainy on average in Mike's city. The transition probability Pr represents the climate changes. For example, there is just 30% chance that tomorrow will be sunny if today is rainy. The emission probability P represents Mike's intention to perform a certain activity on each day. If it is rainy, there is a 50% chance that he is at his apartment cleaning the house. Conversely, if it is sunny, there is a 60% chance that he is at the beach. Figure 2.3 contains the graphical model of this weather problem.

Figure 2.3: The Weather Example



2.4 Conditional Random Fields

Conditional Random Fields (CRF) are a class of probabilistic framework usually applied in pattern recognition. There are some advantages in using CRF in replacement of HMM, especially for the task of sequence labeling. The first limitation of HMM is that

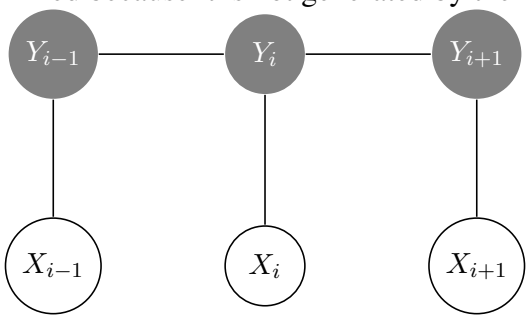
it requires the enumeration of all possible observations. As demonstrated in Figure 2.3, the simple weather example enumerates all the three possible observations (Beach, Shop, and Clean) and these observations are part of the model together with the emission probabilities. For most of real-world applications, enumerating all possible observations is impossible (WALLACH, 2004). Another important limitation of HMM is the assumption that the observation element, at any given instant of time, may only directly depend on the state or label at that time. This observation independence constraint is not valid in most cases, and it is not different in sequence labeling. When labeling sequence data, the previous observation (e.g. a word) has strong influence in the label of the next observation (JOHN LAFFERTY ANDREW MCCALLUM, 2001). CRF overcomes both aforementioned issues and studies have shown that CRF achieved a better performance when compared to HMM (JOHN LAFFERTY ANDREW MCCALLUM, 2001). Hence, some techniques evaluated the use of CRF for the task of document metadata extraction (LUONG; NGUYEN; KAN, 2010; PENG; MCCALLUM, 2004, 2006; SARAWAGI; COHEN, 2004).

2.4.1 Theoretical Foundations

In what follows, **X** is a random variable over data sequences to be labeled, and **Y** is a random variable over the label sequences. Every $Y_i \in Y$ assumes a label from a finite alphabet W. In the context of sequence labeling problem, an instance of this model would have X ranging over natural language sentences and Y ranging over part-of-speech tags, with W containing the set of all possible part-of-speech tags (JOHN LAFFERTY ANDREW MCCALLUM, 2001).

A CRF may be viewed as an undirected graph globally conditioned on X. Formally, $G = (V, E)$ where V corresponds to each of the random variable $Y_i \in Y$. Then (X, Y) is a conditional random field in case, when conditioned on X, the random variables Y_i obey the Markov property with respect to G: $p(Y_i|X, Y_j, i \neq j) = p(Y_i|X, Y_j, i \sim j)$, where $i \sim j$ means that i and j are neighbors in G. Thus, a CRF is a random field globally conditioned on the observation X (WALLACH, 2004). The structure of the graph G may be arbitrary and it represents the conditional independences in the label sequences which are being modeled. However, the most common representation encountered is a first-order chain in which nodes correspond to elements of Y (Y_i), as illustrated in Figure 2.4.

Figure 2.4: An instance of a graphical representation from a simple CRF model. X random variables are dimmed because it is not generated by the model.



The graph G may be used to factorize the joint distribution over elements $Y_i \in Y$ into a normalized product of strictly positive potential functions. Each potential function operates on a subset of the random variables represented by vertices in G. The absence of an edge between two vertices implies that the random variables (nodes) are conditionally

independent given all other random variables in the model. As a result, the potential functions must ensure that conditionally independent random variables do not appear in the same potential function. One solution is to have the potential functions operating on a set of random variables (vertices) that form a maximal clique in G. In Figure 2.4, each potential function will operate on pairs of adjacent label variables (e.g. Y_i and Y_{i+1}) (WALLACH, 2004).

JOHN LAFFERTY ANDREW MCCALLUM (2001) define the probability of a particular label sequence Y given the observation sequence X (joint distribution) to be a normalized product of potential functions in the following form:

$$p(Y|X) = exp(\sum_j \lambda_j t_j(y_{i-1}, y_i, x, i) + \sum_k \mu_k s_k(y_i, x, i)) \tag{2.1}$$

where $t_j(y_{i-1}, y_i, x, i)$ represents the transition feature function of the entire observation sequence and the labels at positions i and $i-1$; $s_k(y_i, x, i)$ represents the state feature function of the label at position i and the observation sequence; λ_j and μ_k are parameters to be estimated from training data (WALLACH, 2004) and to be explained latter in this section.

When defining feature functions, a set of real-valued features $b(x, i)$ are provided expressing some useful characteristics that help constructing the model. An example of such feature is given below:

$$b(x,i) = \begin{cases} 1 & \text{if the observation at position } i \text{ contains the world "Author"} \\ 0 & \text{otherwise} \end{cases}$$

Each feature function contains the value of one of these defined features. Hence, all feature functions are also real-valued. For example, in the context of POS tagging, the following transition feature function can be defined:

$$t_j(y_{i-1}, y_i, x, i) = \begin{cases} b(x,i) & \text{if } Y_{i-1} = NP \text{ and } Y_i = NNP \\ 0 & \text{otherwise} \end{cases}$$

The parameter estimation problem is to determine the parameters λ_j and μ_k of Equation 2.1 from training data, which is formally defined as $\theta = (\lambda_1, \lambda_2, ..., \lambda_j; \mu_2, \mu_2, ..., \mu_k)$. Assuming the training data as $(x^{(k)}, y^{(k)})$, the product of (2.1) over all training sequences, as a function of θ, is known as the likelihood, denoted by $p(y^{(k)}|x^{(k)}, \theta)$. Maximum likelihood training chooses parameter values such that the logarithm of the likelihood, also known as log-likelihood, is maximized (WALLACH, 2004). As it is not possible to manually determine the parameter values that maximize the log-likelihood, iterative techniques are considered, such as iterative scaling (DARROCH; RATCLIFF, 1972) or gradient-based methods (WALLACH, 2002). WALLACH (2004) proposes another procedure to identify the maximum-likelihood parameter values using a dynamic programming method, which is similar to the forward-backward algorithm for HMM. In order to identify the most probable label sequence Y, given an observation X, the Viterbi algorithm can be applied, exactly as used in HMM model (FORNEY G.D., 1973).

2.5 Summary

This chapter explained the base concepts required to understand the techniques applied in document metadata extraction. Section 2.1 described the different types of metadata, in special the metacontent that describes "data about data content" and do not take

the document structure into consideration. Section 2.2 explained Dynamic Programming which is a powerful programming technique to solve complex problems efficiently. LCS is a one common problem solved by DP. We will use this technique to solve some complex problems in the task of metadata extraction. Sections 2.3 and 2.4 described two probabilistic frameworks that are commonly applied in metadata extraction, which is an instance of the sequence labeling problem. Chapter 3 will summarize some algorithms and techniques to solve the problem of document metadata extraction.

3 RELATED WORK

This chapter explains the existing techniques for the task of metadata extraction from scientific documents. According to LIPINSKI et al. (2013), existing solutions are divided into tree different sub-areas: template matching, web-base lookup, and ML. The following sections discuss each of the aforementioned approaches, enumerating different algorithms and their results.

3.1 Template Matching

Solutions that use template matching try to build an apriori structure (template) for the metadata candidates based on known properties of the desired content. With the defined template, a post-processing method is applied to identify the metadata that match each template. For example, to identify the title of a given article, a simple template would be the biggest font with bold style on the first page. Then, the post-processing procedure would load an article, get the first page and look for the line that matches the title's template. The next subsections describe the two state-of-the-art algorithms that use template matching to identify document metadata.

3.1.1 Header Metadata Extraction from Semi-structured Documents Using Template Matching

HUANG et al. (2006) propose a method for header metadata extraction from documents stored in the PDF format. The document is considered as string with format and the defined templates are used to guide a finite state automaton to extract the metadata of a given article. In this article, they have defined templates for title, author(s), affiliation(s), abstract, and keywords. These elements have been formally referred as header metadata. The template definition for each header metadata is given below:

- **Title:** location is always on the upper portion of the first page, with the biggest font size. Position is always in the middle of the line, centered and with bold font.
- **Authors:** location is always immediately under the title. Font size is always smaller than title font. Font is always the same for all authors. Break symbols are " ", "," or "and". They can be listed in one or more columns.
- **Affiliations:** location is always immediately under the authors' list and before the abstract. Classical words are "university", "department", "@", etc. Font is always the same for all affiliations. Only one affiliation appears when all authors are associated with it, or affiliations mapping to authors are listed separately (one affiliation per author).

- **Abstract:** location is always immediately under the affiliations' list. Keyword is always in bold style and the content usually contains "abstract". Abstract is always before the keywords or introduction (when the keywords are missing). The abstract is optional.

- **Keywords:** location is always immediately under abstract and before introduction. It usually starts with "keywords". The keywords are optional.

In order to build a flexible solution, the templates can be redefined to match the different document layouts. The authors used XML Schema to formulate the templates. To match the template, a template matching model is created which is basically a finite state automaton where, the internal states, are the defined metadata. Additionally, there is an initial state *start* and the end state *end*. Finally, there is a state *any* representing that the data stream does not match any template.

Experiments used 400 scientific articles downloaded from digital libraries of ACM, IEEE, ELSEVIER, and LNCS. The sample is evaluated according to precision, recall, F-Measure, and accuracy. Precision is the fraction of the right metadata in the number of metadata extracted. Recall is the fraction of the correct metadata in the number of metadata which is considered correct. Accuracy is the fraction of the correct metadata in the number of all extracted metadata. Partially correct results are considered as wrong. For example, if all authors are not properly identified, this is considered partially correct yielding a negative score in the performance evaluation. In this article, the authors used mainly the accuracy measure to evaluate the system. Title demonstrated to have the most accurate result (96.3%), due to its standard structure. Affiliation(s) and Author(s) presented the lowest accuracy (71.9% and 80.2%, respectively) due to the dynamic nature of these elements. Those metadata have the most complex structural descriptions with many uncertainties. This was the main reason for the low accuracy results, according to the authors. Abstract and Keywords presented accuracy of 88.4% and 84.7%, respectively. The overall accuracy of the entire system was 84.3%, considering all metadata items.

3.1.2 Automated Template-Based Metadata Extraction Architecture

FLYNN et al. (2007) propose a two-level template matching algorithm. The first level classifies documents by layout and the second level provides a template for each layout. With this multi-level approach, templates are independent from one another. Different layouts usually end-up with complex sets of rules when using the same set of templates, leading to maintenance and scaling issues. This solution also overcome this layout heterogeneity problem, because the templates are layout-specific.

The overall process receives a PDF document as input, which can be either text-based or image-based. The document is then processed by an Optical Character Recognition (OCR) software and converted to an application-specific format (XML). In their process, they consider that documents may contain Report Document Page (RDP), which is a standard form to provide metadata. Hence, the first extraction process is to recognize any RDP forms present. Any documents without recognized forms enter the non-form extraction process. This process generates a candidate extraction solution from the available templates. After the candidates are extracted, the post-processing phase handles cleanup and normalization of the metadata. The last automated step is the validation phase, which uses an array of statistical tests, and determines the acceptance criteria to the extracted metadata. Any document that fails to meet the validation criteria is flagged for human review and correction.

The authors examined only the first and last five pages of a document, considering that the metadata can only be found in these pages. To model the document, an *Independent Document Model* (IDM) has been developed which was based on the OmniPage 14 OCR software output. The main structural components of this model are pages, regions, paragraphs, lines and words. The geometric boundaries of each of the structural elements are included as attributes. Style information, for example font style and size, is recorded at the line and word levels. Alignment and line spacing are recorded at paragraph elements.

A group of documents from which the metadata can be extracted using the same template, is called *class*. The members of a class can be selected based on structural or visual similarity. The proposed solution used several different layout schemes in order to separate the incoming document into the appropriate class for extraction. The defined template than extracts the document metadata candidates which go into the output processing to the final validation and cleanup.

Regarding the experiments, they used 9825 documents from the DTIC collection and 728 from the NASA collection. To test the ability of the system in selecting the appropriate template, they have manually classified the DTIC collection into 37 separate classes with at least 5 documents. Templates have been defined for 11 largest classes and test the ability of the extractor to correctly identify the proper class. They have achieved 87% classification accuracy. The overall accuracy for metadata extraction was 66% for DTIC collection and 64% for NASA collection. The authors justified the low accuracy due to the limited number of written templates. Assuming that all the necessary templates are in place, they expect accuracy in the 90% range.

3.2 Web-base lookup

Techniques that use web-base lookup try to identify the smallest unit of information from the document itself, usually using template matching, and then retrieve the complete information from a universal database, such as online services like Google Scholar, IEEE, and ACM.

AUMÜLLER (2009) proposes an algorithm that builds a fingerprint for a PDF article and then submits a query to an online metadata repository. The results are matched to identify the correct metadata entry. The first step is to convert the PDF to text and build a fingerprint, i.e. the query terms that are likely to locate the article in any metadata repository. As Google Scholar indexes the fulltext and not only the metadata of articles, it is possible to query in this service using any identified text fragment from the given document. The algorithm proposed takes a fragment from the beginning of the document which usually contains discriminative content, such as title, authors and abstract. Before submitting the fingerprint to the search engine, a pre-processing phase is performed with the goal of removing undesired content from the input, for example superscripts, special characters, numbers, among others. If no match is found, it might be due to some extra information placed in the beginning of the document. In this case, a second query is performed with just the document title.

As querying the search engine will usually result in multiple entries, the correct entry, if available, has to be matched to the given document. To find the correct entry, the author looks for the title string contained in the fulltext returned from the search engine. They evaluated the proposed algorithm with 91 articles from the proceedings of VLDB 2007. For this set, they have achieved 100% accuracy. On the other hand, if the metadata repository does not contain the PDF article, the algorithm cannot give a proper solution.

3.3 Machine Learning

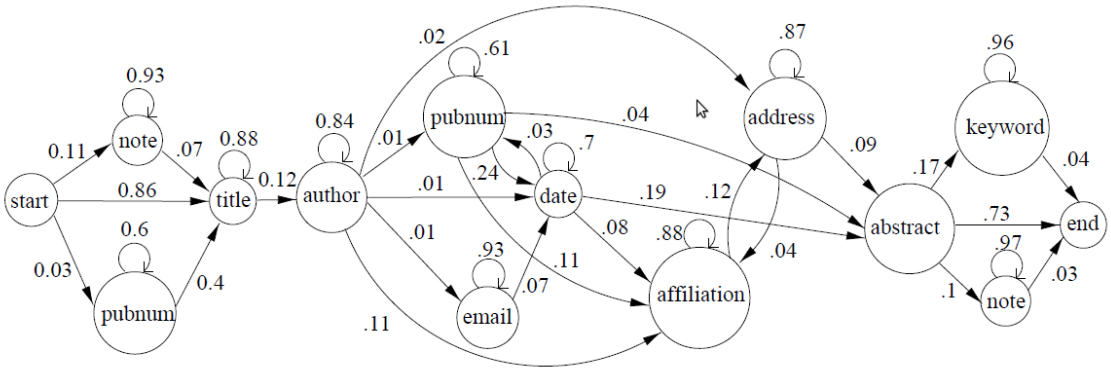
ML techniques build statistics frameworks, usually based on training samples, to avoid creating undesired assumptions about the document layout and content. The algorithms learn from the given samples to automatically identify metadata from real-instance documents. Solutions usually apply sequence labeling problems to the task of metadata extraction where the labels are the configured metadata (e.g. title, author, affiliation) and the observations are the document entries with their given features (e.g. font size, weight, etc...). Two probabilistic models can be found in the literature applied in this area: HMM and CRF. The next subsections aim to describe algorithms that use these techniques to solve the task of document metadata extraction.

3.3.1 HMM for Metadata Extraction

SEYMORE; MCCALLUM; ROSENFELD (1999) centered around the task of extracting information from the headers of computer science research articles using a single HMM model. The authors focus on the automatic creation of the model using the training samples. Unlike their work, other systems use either one state per class or hand-built models assembled by manually inspecting the training examples. The header of the research article consists of title, author names, affiliations, and addresses.

The solution is to label each word of a header as belonging to a class such as title, author, data, or keyword. This is achieved by modeling the entire header with one HMM. In the HMM model, each state is associated with a class to be extracted. Each state emits words from a class-specific distribution. It is possible to learn the class-specific distributions and the state transition probabilities from training data. The words from the header are treated as observations. To retrieve the most-likely state sequence the Viberbi algorithm has been used. An example of a HMM model annotated with class labels and transition probabilities is shown in Figure 3.1.

Figure 3.1: An example of HMM model for Metadata Extraction.



(SEYMORE; MCCALLUM; ROSENFELD, 1999)

From this constructed model, it is possible to notice that, from the start state, there is a probability of 86% that the first word belongs to a title class, and 88% chance that the next word continues to belong to that class.

In order to build a HMM, one must decide how many states the model should contain, and what transitions between states should be allowed. A simple solution is to have one state per class, and to have a fully-connected model (transitions from any state to any other state). However, this model may not be optimal in all cases. Studies have shown

that, when a specific hidden sequence is expected, multiple states per class might get better results.

An alternative to assigning one state per class is to learn the model structure from training data. Each word in the training data is assigned its own state. A transition is placed from the start state to the first state of each training instance, as well as between the last state and the end state. Based on this model structure, the authors propose two merge techniques. First, combine all states that share a transition and have the same class label. For instance, adjacent title states are merged into a single title state with self-transition loop. The probability of this transition is the expected state duration. Second, merge any two states that have the same label and share transitions from or to a common state.

Model structure can be learned automatically from training data using a technique like Bayesian model merging (STOLCKE, 1994). This techinique finds the model structure that maximizes the probability of the model M given some training data D, by iteratively merging states until a threshold is achieved. The authors used the Bayesian merging model so that learning the appropriate model structure for metadata extraction can be accomplished automatically.

Once the model has been defined, the transition and emission parameters need to be estimated from training data. Building labeled data is time-consuming, since manual effort is required. However, it is valuable to define those data, since the transitions and word occurrences in a class can be used to calculate the maximum likelihood estimation for the parameters. On the other hand, unlabeled data can be used with the Baum-Welch training to exercise the model parameters. Baum-Welch training suffers from the fact that it finds local maxima, and is thus sensitive to initial parameter settings.

Regarding the experiments, 1000 headers were manually tagged with class labels, divided into 500-header, 23.557 word tokens labeled for the training set and a 435-header, 20.308 word tokens for the test set. 65 of the headers were discarded due to formatting issues. 5000 unlabeled headers, composed of 287.770 word tokens were designated as unlabeled training data. Performance was measured by word classification accuracy, which is the percentage of header words that are emitted by a state with the label of the words' true label.

The authors tested four different models: full, self, ML and smooth. The full model is a fully-connected solution where all transitions are assigned uniform probabilities, relying only on the emission distributions to choose the best path. This model achieved a maximum accuracy of 64%. The self model is similar, except that the self-transition probability is set according to the maximum likelihood estimation from the labeled data. The accuracy of this model reaches to 89.4%. The ML model sets all transition paramaters to their maximum likelihood estimation, and achives the best results of 92.4%. The smooth model adds an additional smoothing count of one to each transition. The smooth technique did not improve the accuracy, staying with maximum accuracy of 92%.

YIN et al. (2004) use a bigram HMM for automatic metadata extraction from bibliographies with various styles. Different from the traditional HMM, which uses only the word frequency, this model also considers both words' sequential relation and position information in text fields.

In the model proposed by SEYMORE; MCCALLUM; ROSENFELD (1999), it is unlikely that two words with the same frequency in the same state have equal importance. This happens because it ignores any sequential relationship among multi-words. For example, phrases like "Technical Report" will have the same probability as "Report

Technical", even though the occurrence of "Report Technical" is unusual. The Bigram HMM overcomes this problem using a modified model for computing emission probability, while keeping the structure of the HMM unchanged. The emission probability for this new model can be calculated as follows, where q means a given state and σ means a given word:

$$P(\sigma|q) = \begin{cases} P(q \downarrow \sigma) & \sigma \text{ appears in the beginning of } q \\ P(q \uparrow \sigma) = P(\sigma|\sigma_{-1}, q) & \sigma \text{ appears in the inner of q} \end{cases}$$

where $P(q \downarrow \sigma)$ denotes state q emits word σ as the beginning word, and $P(q \uparrow \sigma)$ denotes q emits σ as the inner word. σ_{-1} means the word before σ. YIN et al. (2004) propose a modification to the Viterbi algorithm for the bigram HMM, including the beginning emission probability and the inner emission probability into account.

Due to insufficient training data, bigram HMM may not see some bigram or words, leading to a less-powerful model. The authors applied a smoothing method, known as back off-shrinkage (BIKEL et al., 1997) to overcome this problem.

In their experiments, 713 bibliography entries were extracted from 250 articles. These entries have been hand-labeled to build the training set. The authors used 4-fold cross validation. The final metric is the average of the four experimental results. Precision is the number of tokens correctly tagged using bigram HMM by the number of tokens tagged using bigram HMM. Recall is the number of tokens correctly tagged using bigram HMM by the number of tokens tagged by experts. In the same set, the authors have used their bigram model, the traditional HMM and a rule-based techinique. Bigram HMM makes an improvement in precision more than three percentages than traditional HMM, with an overall precision of 90%.

3.3.2 CRF for Metadata Extraction

LUONG; NGUYEN; KAN (2010) propose a method to detect the logical structure of a document from PDF files using CRF. Also, the authors made use of a richer representation of the document that includes features from an Optical Character Recognition (OCR) tool. The proposed algorithm not only identifies metadata such as title, authors, abstract, but also the logical structure of the document (sections, subsections, figures, tables, equations footnotes, and captions).

The metadata extraction problem has been modeled as a sequence labeling task with the input document as a sequence of lines $L = l_1, l_2, ..., l_n$. Each line l_i needs to be assigned a correct label from a set of classes $C = c_1, c_2, ..., c_m$. To classify a line l_i, features of the line itself are used. Additionally, evidence from previous classifications $l_1, l_2, ..., l_{i-1}$ are also considered. The authors assume that each document line contains text belonging to only one category.

State and transition functions are represented using binary features as illustrated below:

$$b(x, i) = \begin{cases} 1 & \text{if 1st word of line } x_i \text{ is "University"} \\ 0 & \text{otherwise} \end{cases}$$

Given $b(x, i)$, the state and transition functions can be defined as following:

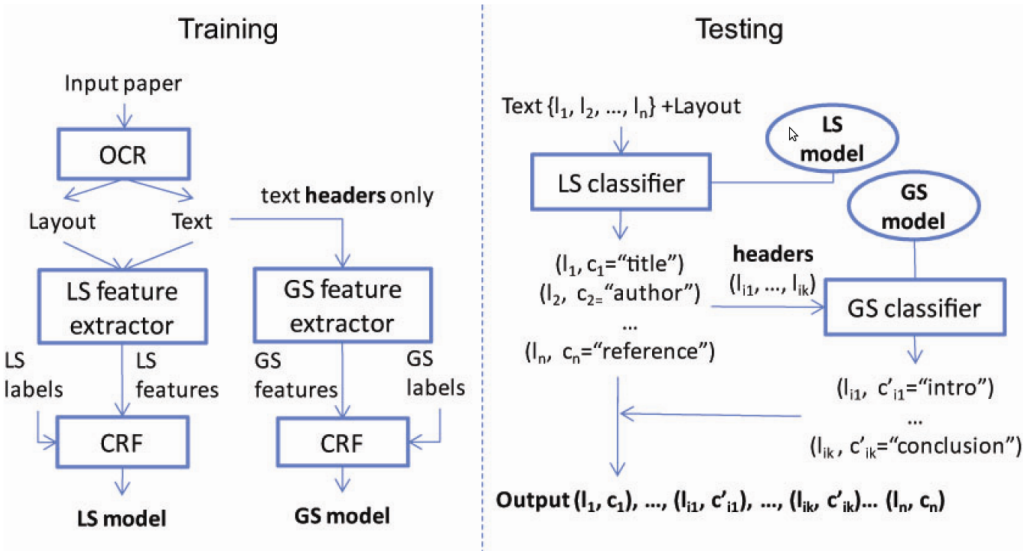
$$s(y_i, x, i) = \begin{cases} b(x, i) & \text{if label } y_i = affiliation \\ 0 & \text{otherwise} \end{cases}$$

$$t(y_{i-1}, y_i, x, i) = \begin{cases} b(x,i) & \text{if } y_{i-1} = author \text{ and } y_i = affiliation \\ 0 & \text{otherwise} \end{cases}$$

Regarding the implementation, the open-source CRF++ package [1] has been used. The input to this package is of the form "$value_1, value_2, ..., value_m category_i$" for each line $l_i (i = 1, n)$, where $category_i$ is the known class used at training time and $value_1, .., value_m$ is the set of feature values. CRF++ automatically converts the given attributes to binary features $b(x, i)$.

The system is composed of two main components: Logical Structure (LS), and a subordinating part, Generic Section (GS). The LS component receives the full-text article as input, while the GS component takes only the header lines. During training time, an OCR engine run through the given article to obtain raw text data together with XML layout information. Both the LS and GS extracted features, together with the manually labeled data, go through the CRF training process to build the corresponding models. At test time, data is represented as a set of lines $\{l_1, l_2, ..l_n\}$ together with XML layout information. The LS classifier labels each line with the corresponding categories, such as title, author, etc. Labels classified as "headers" are passed on to the GS classifier that performs section labeling. The output is the set of lines with their corresponding labels for the LS classifier and the set of header lines with their corresponding section label for the GS classifier. Figure 3.2 illustrates this process.

Figure 3.2: SectLabel Overall Architecture



LUONG; NGUYEN; KAN (2010)

For the LS component, each line of text can be assigned to one category from a set of 23 possibles values: address, affiliation, author, bodyText, categories, construct, copyright, email, equation, figure, figureCaption, footnote, keywords, liteItem, note, page, reference, sectionHeader, subsectionHeader, subsubsectionHeader, table, tableCaption, and title. For the GS component, the authors have defined a set of 13 categories to characterize scholarly document's sections: abstract, categories, general terms, keywords, introduction, background, related work, methodology, evaluation, discussion, conclusions acknowledgements and references.

[1] http://crfpp.googlecode.com/svn/trunk/doc/index.html.

Regarding the defined features, they are divided into two main areas: raw text and OCR-based. Raw text are the minimum set of features used to classify the lines of a document when no rich OCR features are provided.

The LS classifier features have two levels: token and line. The token-level has the feature for the first n tokens in each line (set to 4 experimentally). The line-level has the features that capture the aspects of the line text as a whole. The line-level features have been defined as follows:

- **Location:** relative position of each line within a document.
- **Number:** detects the occurrence of patterns specific to hierarchies ("1.1" and "1.1.1").
- **Punctuation:** checks if the line contains email addresses or web links.
- **Length:** the length of each line in terms of tokens.

The features of the GS classifier are the header position, first and second words, and the whole header. The position feature encodes the absolute and relative positions of the header in a document. The first and second word feature models the individual tokens of the header, for example "Abstract" and "Previous Work". The whole header uses concatenated header as a single feature. This works as a memoization of all headers in the training data.

Raw text features for some metadata produce acceptable results, but in header categories (e.g. title, author, and affiliation) results are usually confused by the CRF models. According to the authors, this happens due to the common properties of these metadata, for example, capitalization patterns and lengths. Additionally, inferring structure directly in PDF documents is difficult as they have many different formats to interpret and model. Hence, the authors decided to handle these documents as a sequence of page images. An OCR engine is then executed to obtain richer format information, for example, font, spacing and spatial layout of elements on the page. The authors used Nuance OmniPage 16 which provides a XML with the mapping features. This XML is normalized to become features of the CRF model.

OCR-based features have been divided into two groups: stationary and differential. Stationary features are extracted directly from the OCR output, as opposed to differential ones that model state changes between two consecutive lines. The stationary features have been defined as follows:

- **Location:** position of the text line within the page. More broadly, raw text location feature measures the position of the line with respect to the whole document.
- **Format:** font information such as font size, bold and italic.
- **Object:** models special line attributes, such as bullet, picture and table.

With the defined stationary features, there is no direct information to the CRF engine to infer if two consecutive lines are of the same format. Hence, differential features have been defined as follows:

- **Format:** to explicitly mark if the current line has the same format as the previous line. The defined properties are: font size, bold, italic, font face and alignment.

- **Paragraph:** to detect blocks of text lines belonging to the same paragraph.

A series of experiments have been conducted by the authors. The tests were performed for the LS and GS component separately. For LS module, 40 scientific articles have been used, where the majority of them in the ACM format. The authors manually assigned categories to each line of these articles using the 23 logical section categories. For GS module, 211 scientific articles were used and, also, manually assigned using the 13 generic section categories.

Let TP denote the number of correctly assigned text lines (true positives), FN for false negatives. FP for false positives and TN for true negatives. Then precision (P), recall (R) and F_1 can be calculated as follows:

$$P = \frac{TP}{TP + FP}, R = \frac{TP}{TP + FN}, F_1 = \frac{2 \times P \times R}{P + R}$$

They perform 10-fold cross-validation for both LS and GS modules. The overall performance of the system for the LS and GS classifier was 93.38% and 95.82% (F_1), respectively. SectLabel project is available as a sub-package of ParsCit, an open-source project for citation extraction (COUNCILL; GILES; KAN, 2008).

3.4 Comparative Analysis

The algorithms presented in this study report are the state-of-the-art solutions for the task of metadata extraction. To the best of our knowledge, there is no other technique except those presented in this work: template matching, web-base lookup, and ML.

The template matching algorithms presented the worst results. The main limitation of this technique is the creation of rule-sets which are tightly coupled to the document layout. Usually, the metadata templates end-up with complex rule-sets that are difficult to maintain and scale. On the other hand, template matching techniques are straight-forward to implement and do not require prior knowledge on probabilistic frameworks, as opposed to ML approaches. Hence, if a solution does not require many different layouts, and, also, does not require constant changes in the defined templates, one may consider using the template matching technique.

Web-base lookup algorithms presented excellent accuracy, but the technique is strongly dependent on a remote document repository. If you are building an application to extract your local PDF metadata, this kind of solution will not be of much help. Additionally, one of the main purposes of automatic metadata extraction research is the difficult of finding good source of information. Hence, web-base lookup algorithms depend on metadata repositories which are currently being enhanced by this area of study, generating an undesired cyclic-dependency. Conversely, if an application is being developed for a context where the documents are publicly available like ACM and IEEE, then this solution may be the best among the three existing techniques.

ML solutions presented the best results, especially the ones using Conditional Random Fields. The main advantage of this method is that there is no apriori assumption made for the document. For example, assuming that the biggest font in the first page is always the title is a strong condition inferred in the document. ML technique relies in the probabilistic models to make those analysis based on the training samples. The drawback of this methodology is that the training data usually need to be manually annotated, which is time-consuming and tedious. Also, if the documents change frequently, the model needs to be re-trained to account for the new document layouts. Hence, new

training samples should be defined and annotated. Another limitation of ML algorithms is that the definition of the exact model structure to maximize the system performance is not a trial task. The defined features are of crucial importance for the model performance. For this reason, most of the algorithms use OCR engines to retrieve what is called as rich document features. As a result, if a given OCR engine does not perform well, the system performance will be also affected.

Table 3.1 summarizes the pros and cons of existing solutions and their overall accuracy. It is important to emphasize that the performance results did not consider the same set of documents, thus not allowing a fair comparison. The reported accuracy serves only as a high-level performance measure of the given solutions.

Table 3.1: Overall comparison for existing metadata extraction techniques.

Description	**Huang et al. (Template Matching)**	**Aumueller (Web-base lookup)**	**Seymore et al. (HMM)**	**Luong et al. (CRF)**
Simplicity	X	X		
Scalability			X	X
Manual Labeling			X	X
Online service dependence		X		
OCR dependence				X

3.5 Summary

This chapter described the state-of-the-art solutions for metadata extraction from documents, which are mostly scholarly articles from scientific literature. There are three main approaches that existing algorithms may fit into: template matching, web-base lookup and ML. The reported results have shown that ML approaches usually get better results when compared to the other techniques, especially the algorithms that use CRF. The next chapter aims to explain the proposed method for this work.

4 ARTIC: A TWO-LAYER CRF METHOD FOR METADATA EXTRACTION

This chapter is divided into four sections. Section 4.1 gives an overview of the method proposed in this work. Section 4.2 explains the details of the first-level layer. Similarly, Section 4.3 explains the details of the second-level layer. Finally, Section 4.4 describes the post-processing algorithms applied to the CRF results in order to provide the metadata output in a well-defined data structure (JSON).

4.1 Overview

Artic employs a two-layer CRF model so as to allow the creation of metadata-specific features. The first layer identifies larger components (sections) that may contain metadata information. These sections are defined as classes in the CRF engine with five possible values: `Header`, `Title`, `Author Information`, `Body`, and `Footnote`. The `Header` usually holds important information about the conference/journal in which the paper has been published. The `Title` class represents the title of the paper. `Author information` contains data about the authors, such as: name, affiliation, and email. The `Body` class does not include useful data for the task of metadata extraction. We do not perform any analysis over this class. `Footnote` usually contains information about the publisher, conference, and possibly some additional information about the authors (e.g. email and affiliation). For some of these sections, a second CRF layer was created. This extra layer allows us to extract the actual metadata and define features specific for the section. The second layer of the CRF model will be executed for the `Header`, `Author Information`, and `Footnote`. The `Title` class does not require another CRF level because it contains only one semantic value, which is the title of the paper.

For each layer, we need to provide the probabilistic framework with evidences that will help identify the class that maximizes the result of the model. These evidences are called *features* (e.g. font size, format, and alignment). The features of a given section do not affect other sections, and if the feature is present in one section that does not mean that that same feature will appear in another section. To implement the CRF model, we used CRF++[1]. These features need to be structured in a way that they can be interpreted by the CRF++ engine. The training and test files should be of the form "$value_1, value_2, ..., value_m\ class_i$", where $class_i$ is assigned to one of the aforementioned classes. This framework also requires a template file which describes the semantics of these columns and, also, enables the creation of the *context windows*. These windows allow the CRF engine to look at the previous/next occurrences to infer about the current state. The template

[1] http://crfpp.googlecode.com/svn/trunk/doc/index.html

file works like a matrix, each single column in the test and/or training file receives its definition in the template, which comes in the form of "$U_i : \%x[0, j]$", where i is the feature identifier, 0 is the current line being observed (use -1,-2,+1,+2 to refer to previous/next lines), and, j is the column index.

Figure 4.1: The CRF++ train/test file.

```
1  line_0 centered ... header new TITLE
2  line_1 centered ... header same TITLE
3  line_2 left... header new AUTHOR INFORMATION
4  line_3 left... header same AUTHOR_INFORMATION
5  line_4 left... header same AUTHOR INFORMATION
6  line_5 left... header same AUTHOR_INFORMATION
7  line_6 left... new new BODY
8  line_7 justified...new new BODY
9  line_9 justified...same same BODY
10 line_10 justified... same same BODY
11 line_11 justified... same same BODY
12 line_14 justified... same same BODY
13 line_15 left true... new new BODY
14 line_16 justified... new new BODY
```

Figure 4.1 shows an example of the train/test file, and, Figure 4.2 shows an example of the corresponding template file definition. Let us assume that the current line being considered is "line_6". Then, in the template engine, 0 represents the current line. When analyzing U19:%x[-2,1], for example, -2 represents "line_4" and 1 represents the feature (alignment). In "line_4" (Figure 4.1), alignment has the value set to "left".

Figure 4.2: The CRF++ template file.

```
1  #identifier
2  U0:%x[0,0]
3  #alignment
4  U1:%x[0,1]
5  ...
6  #alignment window
7  U18:%x[-1,1]
8  U19:%x[-2,1]
9  U20:%x[1,1]
10 U21:%x[2,1]
11 ...
12 # alignment and font size
13 U30:%x[0,1]/%x[0,5]
14 ...
```

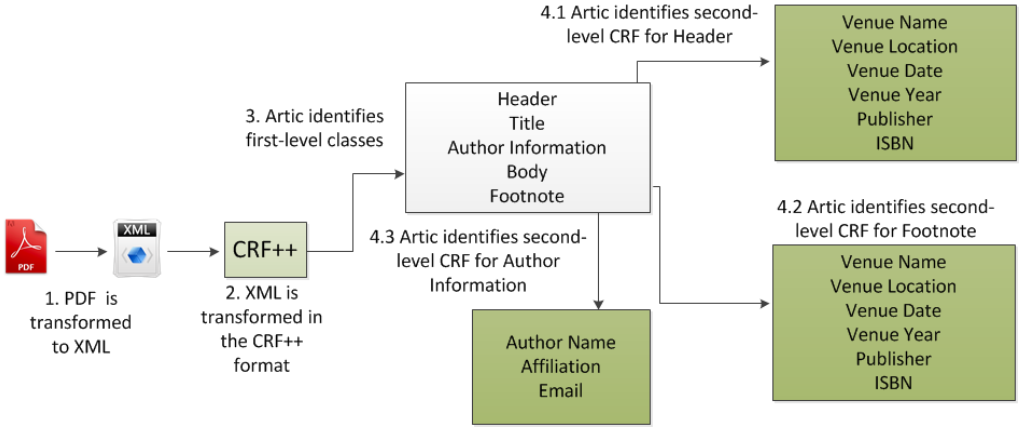
After the CRF model labels the input with the corresponding classes, post-processing algorithms are applied in order to group the data that belong to the same class. The CRF model is able to classify all author names, but it is not able to group the names that belong to the same author. For example, a common representation for the authors of a paper is given in Figure 4.3.

Figure 4.3: A sample representation of the author information of a given paper.

Hang Li, Yunbo Cao
Microsoft Research Asia
5F Sigma Center
No.49 Zhichun Road,
Haidian, Beijing, China, 100080
{hangli, yucao}@microsoft.com

Jun Xu
College of Software
Nankai University
No.94 Weijin Road,
Tianjin, China, 300071
nkxj@yahoo.com.cn

Figure 4.4: The two-layer CRF generation process proposed in this work.



In this scenario, the CRF engine is able to identify all the words that are likely to make up an author name. In Figure 4.3, these words are: Hang, Li, Yunbo, Cao, Jun, and Xu. One of the tasks of the post-processing component is to execute an algorithm that will group the words that belong to the same author. The expected result after executing this algorithm is: Hang Li, Yunbo Cao, and Jun Xu. Similar algorithms are required to group the affiliations and emails. Details of all the algorithms applied in the post-processing component are given in Section 4.4. Figure 4.4 illustrates the whole process performed in this work.

4.2 First-level CRF

When dealing with metadata extraction, the important information is usually present only in the first page. Due to this reason, this work considers only the first page for analysis. The first-level of the model classifies each line into the five classes decribed as follows:

- **Header:** usually is located at the top of the page and contains information regarding the paper itself, such as: venue, publication year, pages, among others.
- **Title:** contains the title of the paper. Papers which has the title spreading over two or more lines, each line should be assigned to this class. The location is usually at the top of the page with the largest font size.
- **Author Information:** contains information about the author, such as: first name, last name, email, affiliation, and address. This class, in most of the cases, appears after the title.

- **Body:** represents non-relevant information for the task of metadata extraction. In this work, abstract, keywords, and other structural components (e.g. section headers, body text, etc...) are not extracted.

- **Footnote:** contains information regarding copyright, conference, pages, publication year, authors' affiliation, address, etc. The location is usually at the bottom-left of the page with a small font size.

In order to distinguish among classes, features must be defined. These features will give the CRF engine evidences to help decide, for a given line $L_{i,j}$, its corresponding class $C_{i,j}$, where i represents the line and j represents the page. Some of the first-level features are: `Alignment`, `Bold`, `Underline`, `Italic`, and `Font Size`. The detailed description of each feature is given in Appendix A.

4.3 Second-level CRF

The second-level of the CRF model extracts the metadata for the `Header`, `Author Information`, and `Footnote`. `Body` and `Title` classes do not require a second layer.

The **Header CRF** model identifies the metadata for the `Header` section which is usually at the top of the first page of the paper. This model explores the words in the lines that have been identified as belonging to the `Header` class in the first-level of Artic. Each word is classified into the five different classes below:

- **Conference Name:** if the word belongs to the list of conference/journal name.

- **Conference Year:** if the word represents the conference year.

- **Conference Date:** if the word belongs to the conference date which is usually at the form "`Month DAY_START-DAY_ENDS`" (e.g. Jan 5-10).

- **Conference Location:** if the word belongs to the conference location which is usually at the form "City, Country" (e.g. Florence, Italy).

- **Publisher:** if the word represents the publisher. Currently, Artic has been tested with papers from ACM, IEEE, Springer and Elsevier.

- **Other:** if the word does not represent any useful information for the task of metadata extraction (e.g. copyright). We do not perform any further analysis over this class.

In order to distinguish one class from another, features were defined in a similar fashion to the ones implemented in the first-layer. These features will help identify for a given word $W_{i,j}$, its corresponding class $C_{i,j}$, where i is the current word in line j. Some of the Header CRF features are: `Word Content`, `Character Length`, `Month`,`Year`, and `Country`. The detailed description of each feature is given in Appendix B.1.

The **Author Information CRF** model identifies the metadata for the `Author Information` section which usually follows the `Title` section. This model explores the words of the lines that have been identified as `Author Information` class in the first-level of Artic. Each word is classified into the four classes below:

- **Author Name:** if the word represents an author name. At this point, we do not identify individual authors. The post-processing algorithms will be responsible for grouping the authors by their names.

- **Affiliation:** if the word is part of the author's affiliation. The metadata that is usually contained in this section are: university, company, department, address, and telephone. At this point we do not assign affiliation to their corresponding authors. The post-processing will be responsible for matching affiliations and authors.

- **Email:** if the word is an email. At this point, we do not assign any email to any author. The post-processing algorithms will be responsible to match the given authors with their corresponding email.

- **Other:** if the word does not represent any useful information for the task of metadata extraction (e.g. "and" word, special characters, and superscript). We do not perform any further analysis over this class.

The Author Information CRF features include `Word Content`, `Possible Affiliation`, and `Possible Email`. The detailed description of each feature is given in Appendix B.2.

The **Footnote CRF** model deals with identifying metadata for the `Footnote` section which is usually at the bottom left of the first page of the paper. This model will explore the words of the lines that have been identified as `Footnote` class in the first-level of Artic. Each word is classified into the classes below:

- **Conference Name:** if the word belongs to the list of conference/journal name.

- **Conference Year:** if the word represents a conference year.

- **Conference Date:** if the word belongs to the conference date which is usually at the form "`Month DAY_START-DAY_ENDS`" (e.g. Jan 5-10).

- **Conference Location:** if the word belongs to the list of conference locations which is usually in the form "City, Country" (e.g. Florence, Italy).

- **Publisher:** if the word represents the publisher. Currently, Artic supports ACM, IEEE, and Elsevier.

- **ISBN:** if the word is represents an ISBN number (e.g. 978-1-60558-01/08/04).

- **Email:** if the word represents an email.

- **Other:** if the word does not represent any useful information for the task of metadata extraction (e.g. copyright). We do not perform any further analysis over this class.

`Word Content`, `Possible Email`, and `ISBN` are among the Footnote CRF features. The detailed description of all features is given in Appendix B.3.

4.4 Post-processing

The last step in Artic is to get the results from the CRF model and apply algorithms that output the metadata in a well-defined format. The CRF provides the classes for the given lines and words. The task of the post-processing step is to use that information to build the output data. The output format is JSON [2].

4.4.1 Algorithms

The first algorithm is for **entity grouping**. The goal of this algorithm is to group all the words that belong to the same class, i.e., authors names and affiliation. Algorithm 1 shows the logic applied for such grouping. The first step is to create an empty map with the group index as the key and an array of words as the map value (Line 1). Every new entry in this map holds a different entity. For example, if the map has four entries while identifying authors, this means that the algorithm has identified four different authors. The core step of this method is in lines 4 and 6. The method $getGroupIndex$ takes the group map and the current word to return the group index that this word belongs to. The calculation is based on the $HORIZONTAL\_BOUNDARY$ and $VERTICAL\_BOUNDARY$ parameters. Those parameters were experimentally set to 20% and 30%, respectively. If $groupIndex$ is $null$ (line 8), it means that the current word does not have any group to be attached to. As a result, a new index has to be created in the group map (line 9 to 11). If $groupxIndex$ is found, the current word is added to the array of words of the group with $groupIndex$ key (lines 13 and 14).

Algorithm 1 Algorithm for entity grouping

Require: List of words identified as Author Name or Affiliation.

```
1:  groupMap = new Map(int, Words[ ]);
2:
3:  for all the words do
4:     groupIndex = getGroupIndex(groupMap, word,
5:          HORIZONTAL_BOUNDARY,
6:          VERTICAL_BOUNDARY);
7:
8:     if groupIndex == null then
9:        wordsOfIndex = new Array( );
10:       wordsOfIndex.add(word);
11:       groupMap(groupMap.size( ), wordsOfIndex);
12:    else
13:       wordsOfIndex = groupMap.get(groupIndex);
14:       wordsOfIndex.add(word);
15:    end if
16: end for
17: return groupMap
```

The second algorithm is the **email matching**. The purpose of this algorithm is to match the identified emails to their corresponding authors.

Algorithm 2 performs the logic required to match authors and emails. From line 2 to 21 the matching is done by calculating the edit distance between the email and

[2] http://json.org.

Algorithm 2 The Email Matching algorithm

Require: the list of authors and list of emails.

```
1:
2:  for each email in emails do
3:    for each author in authors that does not have email do
4:      distance = DynamicProgramming.distance(
5:                              email, author.name);
6:      if distance <= MAX_DISTANCE then
7:        author.email = email;
8:        break;
9:      else
10:       names[] = author.name.split("");
11:       for each name in names do
12:         distance = DynamicProgramming.distance(
13:                                 email, name);
14:         if distance <= MAX_DISTANCE then
15:           author.email = email;
16:           break;
17:         end if
18:       end for
19:     end if
20:   end for
21: end for
```

the author name using Dynamic Programming. If the edit distance is smaller than or equal to $MAX\_DISTANCE$, we found the author for the current email (lines 6 and 7). $MAX\_DISTANCE$ has been experimentally set to 20%. If this is still not satisfactory, we try to use each token in the name of the author instead of the full name (from line 10 to 18). Emails that do not match any author are ignored.

The last algorithm is the **affiliation matching**. The goal of this algorithm is to match the identified affiliations with the corresponding authors. Algorithm 3 shows the logic applied to match affiliations and authors. The logic is very similar to the author name grouping (Algorithm 1). The $getGroupIndex$ function is also used, the map is $affiliationMap$ and, for each author, we identify the best matching affiliation index. The values for *HORIZONTAL_BOUNDARY* and *VERTICAL_BOUNDARY* have been experimentally set to 17% and 70%, respectively. If the $index$ is found, we assign this affiliation to the current author. We do not use any affiliation that we could not match to an author, thus being completely ignored from the final output.

4.4.2 Artic Metadata Output

This subsection shows a sample paper and its expected output using the proposed solution in this work. Figure 4.5 contains a paper with 4 authors published at SIGCSE in 2009. This paper has the ACM format with two columns. The size of the author and email list matches, thus we don't expected the DP algorithm to run at this time (assignment will by declaration order). On the other hand, we need to group author names, affiliations and match the identified affiliations with the corresponding authors. We expect to have 3 distinct affiliations at the end of the execution of the affiliation grouping algorithm.

Algorithm 3 The Affiliation Matching Algorithm

Require: Affiliation and Authors Name map.

```
1:
2: for each author in authorsMap do
3:     index = getGroupIndex(affiliationMap, author,
4:                           HORIZONTAL_BOUNDARY,
5:                           VERTICAL_BOUNDARY);
6:     if index != null then
7:         author.affiliation = affiliationList[index];
8:     end if
9: end for
```

Figure 4.5: A sample paper used to demonstrate the JSON output.

Embedding Computer Science Concepts In K-12 Science Curricula

Chi-Cheng Lin, Mingrui Zhang
Department of Computer Science
Winona State University
Winona. MN 55987
{clin, mzhang}@winona.edu

Barbara Beck
Department of Biology
Rochester Community and Technical College
Rochester, MN 55904
barbara.beck@roch.edu

Gayle Olsen
Department of Nursing
Winona State University
Rochester, MN 55904
golsen@winona.edu

ABSTRACT

To engage a broader audience in computer science, we have developed a set of curriculum units embedded with computer science concepts for K-12 science education. We chose bioinformatics as a vehicle to deliver these units. Our curriculum development cycle began with the identification of a set of computer science concepts which are potentially relevant to life sciences. Problems in life sciences as well as bioinformatics tools to be used for solving these problems were carefully examined for the delivery of identified computer concepts. They were later presented to groups of regional K-12 science teachers in our summer workshop on bioinformatics. With their help, we adapted and polished these curriculum units to meet Minnesota state standards for K-12 science education. This paper describes our approach in developing the curriculum units.

SIGCSE'09, March 3–7, 2009, Chattanooga, Tennessee, USA.

computer science to students in secondary education [3, 8] also contributed to it.

Efforts have been made to battle this shrinking enrollment, such as the re-design of computer science curricula [12] and the outreach to K-12 education [2, 5, 6]. Both efforts are also pertinent to the goals of computer science education of engaging a broader audience in computer science concepts. Currently, the outreach activities are primarily in the format of workshops targeted to either high school computer science teachers or pre-college students. Those workshops are very valuable in recruiting students into the program. However, only a limited number of students are reached, given the fact workshop attendees are the ones who are already interested in computer science. Additional recruitment strategies for reaching a broader audience of K-12 students in North America still need to be identified.

We chose bioinformatics as a vehicle to do it, as bioinformatics is an interdisciplinary field and uses computer technologies and computational tools to model, analyze, present, visualize, and manage biological data. Its primary goal is to retrieve information from the data and use the information to facilitate problem solving in life science and medicine. If done right, we may potentially engage a broader audience in computer science

The expected JSON output is presented in Listing 4.1.

Listing 4.1– The expected JSON output for the sample paper.

```
{
  "title": "Embedding Computer Science Concepts
           In K-12 Science Curricula",
  "authors": [
    {
      "name": "Chi-Cheng Lin",
      "email": "clin@winona.edu",
      "affiliation": "Department of Computer Science
                      Winona State University
                      Winona. MN 555987"
    },
    {
      "name": "Mingrui Zhang",
      "email": "mzhang@winona.edu",
      "affiliation": "Department of Computer Science
                      Winona State University
                      Winona. MN 555987"
    },
    {
      "name": "Barbara Beck",
      "email": "barbara.beck@roch.edu",
      "affiliation": "Department of Biology
                      Rochester Community
                      and Technical College
                      Rochester, MN 55904"
    },
    {
      "name": "Gayle Olsen",
      "email": "golsen@winona.edu",
      "affiliation": "Department of Nursing
                      Winona State University
                      Rochester, MN 55904"
    }
  ],
  "venues": [
    {
      "name": "SIGCSE",
      "publisher": "ACM",
      "date": "March 3-7",
      "year": "2009",
      "location": "Chattanooga, Tennessee, USA",
      "isbn": "978-1-60558-183-5/09/03"
    }
  ]
}
```

4.5 Summary

This chapter explained the proposed solution for the task of metadata extraction. The work is divided into pre-processing, CRF models, and post-processing. The pre-processing is responsible to extract the rich text information from the paper by converting the PDF into Image and running an OCR engine that outputs the paper as an XML format. The CRF models is the core contribution with a two-layer approach to overcome the main limitations of the state-of-the-art solution. The post-processing algorithms handle the grouping and clean-up functionalities. This last part also outputs the paper in a well-defined data structure (JSON). The next chapter will show the results of this work by performing a series of experiments to validate our approach.

5 EXPERIMENTS

The purpose of the experiments presented here is to answer the following questions: (i) Can the two-layer CRF model improve the classification results compared to using a single-layer? (ii) Can the results hold for a larger dataset? And, finally, (iii) Can the post-processing algorithms properly identify the relationship between authors, emails, and affiliations?

The following sections are divided as follows. Section 5.1 explains how the experiments were structured, providing data and metrics to support our evaluations. Section 5.2 compares the proposed approach against the baseline. Section 5.3 validates the proposed solution against the JSON gold-standard.

5.1 Experimental setup

We have selected SectLabel as our baseline due to their good results reported in (LUONG; NGUYEN; KAN, 2010). Also, SectLabel has an open-source tool[1] that allowed us to have easy access to the dataset that was used to validate their proposed solution. Please refer to Chapter 3, for more details on how SectLabel works.

The dataset used in our experiments consists of of 100 scientific papers from IEEE, Elsevier, Springer and ACM. This set of papers already includes the 40 papers used by SectLabel. The remaining 60 papers were selected by eight postgraduate students from our institution. In addition to the papers, the students also provided the expected output, in JSON format, for each metadata identified by them in the papers. We refer to that as *JSON gold-standard*. In order to extract richer information from the PDF we convert the pages into images and run an OCR engine. The information provided by the OCR engine includes: coordinates, format, font size, font type, etc. The output of the OCR is a XML file. Based on the XML document we build the CRF components using the CRF++ format. OmniPage Professional Version 18[2] was used to perform all OCR-related operations. The choice for this tool was made based on their usage in the academic area and their good reported results. Also, we have manually annotated the corresponding classes for each of the CRF levels using the CRF++ format. We refer to that as *Classes gold-standard*. When selecting the papers, we advised the participants to cover as many different formats as possible. We believe this helps ensuring that the proposed method is not restricted to a specific article format.

For the validation procedure, we applied the same strategy as SectLabel. We used 10-fold cross validation for all the experiments performed in this work. K-fold validation is

[1] https://github.com/knmnyn/ParsCit/tree/master/bin/sectLabel

[2] http://www.nuance.com/for-individuals/by-product/omnipage/index.htm.

a commonly used strategy. The systems were evaluated based on the F_1 measure, which is defined as follows. Let us assume that TP denotes the number of correctly assigned classes (true positives), FP denotes the number of incorrect classification (false positives), and FN denotes the number of incorrect classifications in reverse order (false negatives). For example, assuming that we are evaluating the `Title` class. TP would be the correct hits. FP would be the wrong hits, the actual class would be `Author Information`, for example, and it was classified as `Title`. FN would be the misses, e.g. if the actual class was `Title`, and was classified as `Author Information`. Then, precision (P), recall (R) and F_1 are calculated as follows:

$$P = \frac{TP}{TP + FP}, R = \frac{TP}{TP + FN}, F_1 = \frac{2 \times P \times R}{P + R}$$

In other words, precision can be thought as "from what the method classified as `Title`, what was actually `Title`". In a similar way, recall can be thought as "from what should have been assigned as `Title`, how many were classified as such by the method".

5.2 Evaluation against the baseline

The goal of the **first experiment** is to test whether the two-layer CRF model improves the classification results compared to SectLabel single-layer strategy. For this test we used the same 40 papers which have been used by the baseline in the experiments reported in (LUONG; NGUYEN; KAN, 2010). The comparison is done using the *Classes gold-standard*. The set of metadata identified by Artic differs from the set of metadata identified by SectLabel. As a result, for this experiment, we compared just the classes that both systems provide. SectLabel results were extracted from the original paper (LUONG; NGUYEN; KAN, 2010), but we had to combine some of their classes to match ours. For example, Artic Affiliation class already includes the address as part of the metadata. So, we have combined SectLabel Address and Affiliation classes by taking the average over their F_1 results.

Table 5.1 presents the results for both methods using a total of five classes (`Title`, `Author Name`, `Email`, `Affiliation`, and `Footnote`). Artic's average F_1 is 99.84%, which represents a relative gain of 6.92% compared to SectLabel. Also, the classes `Affiliation` and `Footnote` presented the biggest improvements when applying the two-layer CRF model.

Table 5.1: Can the two-layer CRF model improve the classification results compared to using a single-layer?

Class	**SectLabel** (Single-layer)	**Artic** (Two-Layer)
Title	100.00 %	100.00 %
Author Name	97.74 %	99.41 %
Email	97.64 %	100.00 %
Affiliation	89.15 %	99.83 %
Footnote	82.34 %	100.00 %
Average (F_1)	**93.37** %	**99.84** %

Table 5.2: Can the results hold for a larger dataset?

Class	**Artic** (40 papers)	**Artic** (100 papers)
Title	100.00 %	100.00 %
Author Name	99.41 %	98.91 %
Email	100.00 %	100.00 %
Affiliation	99.83 %	99.64 %
Venue Name	85.20 %	85.94 %
Venue Year	100.00 %	100.00 %
Venue Date	100.00 %	98.89 %
Venue Publisher	100.00 %	100.00 %
Venue Location	93.86 %	96.60 %
ISBN	100.00 %	98.82 %
Average (F_1)	**97.83** %	**97.88** %

5.3 Evaluation using a larger dataset

The **second experiment** performed in this work was to evaluate the behavior of the system with a larger dataset. The test used 100 papers, as described in Section 5.1. This time, we have included all 10 classes that Artic identifies. The comparison here is still in the classification level, meaning the *Classes gold-standard*. Table 5.2 summarizes the results of this experiment. The average F_1 with 40 papers was 97.83%, and, for 100 papers, it was 97.88%. These numbers demonstrate that the results hold while for a larger number of papers. Venue name presented the lowest F_1 among all the classes that Artic identifies. We have analyzed the cases in which Artic failed and most of them occurred when the name of the venue is in full (e.g. International Conference on ML) rather than abbreviated (e.g. ICML).

The **last experiment** aims at verifying if the post-processing algorithms can properly identify the relationship between authors, emails, and affiliations. We have run Artic with the 100 paper dataset and compared the generated JSON against the gold-standard provided by the annotators (*JSON gold-standard*). One problem that occurred during this experiment was the comparison between the generated output and gold-standard JSONs. For instance, Listing 5.1 shows an example of a gold-standard JSON element for author information. Similarly, Listing 5.2 shows a hypothetical automatically generated JSON for this author element. One may observe that the author's last name has been erroneously identified as another author. Similarly, the author's first name was appended by a number 2 (i.e., an undesired superscript). Also, the University name has been divided in two. The challenge here is how to calculate precision, recall, and F_1 for this comparison. In other words, how close these two JSONs are from each other? To the best of our knowledge, there is no tool that can compare two JSONs. As a result, we developed a method that does just that. We resorted to similar comparisons made in another area, i.e. Plagiarism Detection, to develop this assessment. In (POTTHAST et al., 2010), an evaluation framework provides a model to calculate precision and recall for plagiarism detection systems. We have implemented a similar approach which is explained as follows.

Let C be the set of classes that Artic provides. Let W_c be the set of words of a class $c \in C$ from the gold-standard JSON. Similarly, let G_c be the set of words of a class $c \in C$

from the generated JSON. Precision, Recall and F_1 for a given class c is given as follows:

$$P_c = \frac{\sum_{g \in G_c} g \cap W_c}{|G_c|}$$

$$R_c = \frac{\sum_{w \in W_c} w \cap G_c}{|W_c|},$$

$$F_{1_c} = \frac{2 \times P_c \times R_c}{P_c + R_c}$$

where $\cap$ means the number of matching characters from the best match in W_c or G_c.

Listing 5.1– The gold-standard JSON element.

```
{
  "authors": [{
      "name": "Alan Souza",
      "email": "apsouza@inf.ufrgs.br",
      "affiliation": "Cal University"
  }]
}
```

For example, precision, recall and F_1 for the Author Name are calculated as follows:

$$P_{name} = \frac{4}{5}, R_{name} = \frac{4}{9}, F_{1_{name}} = \frac{2 \times 0.8 \times 0.44}{1.24} = 0.57$$

Listing 5.2– The possible generated JSON element.

```
{
  "authors": [{
      "name": "Alan2",
      "email": "apsouza@inf.ufrgs.br",
      "affiliation": "Cal"
    }, {
      "name": "Souza",
      "affiliation": "University"
    }
  ]
}
```

Table 5.3 compares the post-processing results against the classification step. The post-processing results were extracted by applying the above formula for each of the 10 classes in Artic. After executing the post-processing algorithms, the average F_1 presented a relative loss of 5.34%. The results for author name and email are still below our expectations, while affiliation had itself a relative loss of 11.30%. During our experiments, we observed different affiliations that are positioned very close too each other, thus not allowing the heuristic framework to detect that those are actually distinct entities. We believe that this was the main factor that affected the affiliation performance. Figure 5.1 shows an example of this problem.

5.4 Summary

This chapter presented the experiments performed in this work. Artic outperformed SectLabel (baseline) by 6.92% with an overall precision of 99.84% for the classification

Figure 5.1: Distinct Affiliations problem (too close)

Eric Baumer **Mark Sueyoshi**
Department of Informatics Int'l Studies / East Asian Cultures
U of California, Irvine, USA U of California, Irvine, USA

Table 5.3: Can the post-processing algorithms properly identify the relationship between authors, emails, and affiliations?

Class	**Artic** (Classification)	**Artic** (Post-Processing)
Title	100.00 %	100.00 %
Author Name	98.91 %	95.78 %
Email	100.00 %	92.57 %
Affiliation	99.64 %	88.38 %
Venue Name	85.94 %	77.40 %
Venue Year	100.00 %	97.94 %
Venue Date	98.89 %	95.27 %
Venue Publisher	100.00 %	97.22 %
Venue Location	96.60 %	83.86 %
ISBN	98.82 %	98.14 %
Average (F_1)	**97.88** %	**92.65** %

step. We also tested Artic with a larger dataset (100 papers) and results were not affected. The last experiment tested the post-processing algorithms. Results have shown a relative loss of 5.34% when compared to the classification step. Chapter 6 will conclude the work by discussing the limitations and presenting future works.

6 CONCLUSION

The first step of Artic is to extract information (evidences) from the PDF in order to build the features that will be further used by the CRF model. We tried a set of tools to extract data such as: font size, alignment, font family. To the best of our knowledge, the most effective way is to use an OCR engine that will convert the PDF into image and output a XML with all rich text features inside. Among all the OCR engines available, we chose OmniPage 18 Professional due to its good reported results in the academic area. The limitation here is that OmniPage is for Windows only, which restrains our pre-processor to run in a specific operating system. Another issue is that if the OCR fails to recognize some words, this may lead to classification errors. One benefit of using an OCR tool is that we can basically support any format that the tool provides, not only PDF. We tried to run Artic using a paper with Microsoft Word format and the results were promising. The Artic dataset includes 100 papers. While the number is more than double the size of the state-of-the-art solution, we still consider 100 papers a small set as compared to the number of different paper styles available online. Also, manually annotating these papers took us considerable time. Another possible future work is to evaluate the possibility of leveraging this set of papers and build a broader dataset minimizing human intervention.

The post-processing algorithms try to identify the relationship between authors, emails, and affiliation. At this level, we do not introduce any ML, but, in contrast, we apply some heuristics to identify the components that should be grouped together. The main limitation is when we do not find the group index for a specific element (e.g. author name is off the boundaries for existing authors in the index). In this scenario, some information may be completely ignored in the JSON output. DO et al. (2013) presents a method to match author and affiliations using Support Vector Machines (SVM). A possible future work is to evaluate the use of ML techniques also at the post-processing level.

This work presented Artic, a metadata extraction approach based on a two-layer CRF model. The first-layer focuses on identifying the line-level structural components that may contain metadata information: `Title`, `Author Information`, `Header`, and `Footnote`. The second-layer analyzes, for each structural component, the words that may belong to a specific metadata element. Artic identifies a total of 10 metadata elements: Title, Author Name, Author Email, Author Affiliation, Venue Name, Venue Year, Venue Date, Venue Publisher, Venue Location, and ISBN. Author data can be found at the `Author Information` layer. Similarly, Venue data can be found either at `Header` or `Footnote` levels. Additionally, Artic provides a post-processing component that links authors, emails, and affiliation based on pre-defined heuristics.

We carried out an evaluation composed of three experiments. Artic was compared against our baseline (SectLabel) and against a gold-standard with the expected JSON output. A total of 100 real PDF articles were used as test and training set in a 10-fold cross-validation setting. Artic presented a relative gain of 6.92% compared to SectLabel. The post-processing algorithms represented a relative loss of 5.34% compared to Artic classification step. A possible future work would consider using ML techniques also at the post-processing level.

Artic has been released as an open-source tool for anyone interested in re-using the components developed during this work. Links to all 100 papers used in our experiments together with their annotations and JSON output set can be found in the website https://github.com/alansouzati/artic-poc. We have submitted a paper for DocEng 2014 conference which will be held in Fort Collins, USA and has a Qualis of B1.

APPENDIX A DETAILED FIRST-LEVEL FEATURES

- **Identifier:** the line identifier, starts with "line_0" and goes until "line_n", where n is the last line of the page. This could be a good indicator for the classes as their location usually does not change.

- **Alignment:** the line alignment with four possible values: left, center, right, or justified. We expect this feature to be very useful for the identification of the `Title` (usually centered).

- **Bold:** boolean that represents whether the line is bold (true) or not (false). This feature could be a good evidence for identifying `Title` which usually are in bold.

- **Underline:** boolean that represents if the line contains any underlined words. This could be important to the `Footnote` that usually represents information with underline.

- **Italic:** boolean that represents if the line contains any word in italic. This feature could be useful to identify `Footnote` as this class usually contains the elements in italic.

- **Font Size:** the font size normalized into four different values: small, normal, medium, or large. The normalization process calculates the average font size for the given page, and, for each line, it assigns a value depending on the distance to the average font size. The values can be: *small* if it is more than 10% smaller than the average; *normal* if its difference in relation to the average size is less than 10%; *medium* if the size is from 10% to 45% larger than the average; and *big* if the size is over 45% larger than the average font size. All values have been set based on empirical observations of the training data. This feature could be very useful to identify `Title`, `Author Information`, and `Footnote`. `Title` usually has the biggest font size, `Author Information` usually has medium, and `Footnote` usually has small.

- **Top position:** the normalized line location with respect to the top position. The normalization process retrieves the large- st top position and creates buckets of 8 bits, which means that each bucket will have a size of $biggestTop/8$. Then, for each line, the current top position will be set depending on the bucket it belongs to. For example, if the current top location fits into the first bucket, its top position will be 0, if it fits into the second bucket, its top position will be 1, and so on. This could

be very useful to identify `Header` which most probably have their top position set to 0.

- **Left position:** the normalized line location with respect to the left position. The normalization process is almost the same as for the top position, the only difference is that it considers the left location instead of the top one. This feature could help to identify `Footnote` which usually has the left position set to 0.

- **Possible Email:** boolean that indicates whether the current line has any "@" sign. This feature is important to identify `Author Information`. If the line contains an email it is a good indication to belong to the `Author Information` class.

- **Number of Words:** the normalized number of words for the current line with four possible values: zero, few, medium, or many. The normalization process first extracts the words for the given line. Zero represents an empty line. Few if the line has between 1 and 4 words. Medium if the line between 5 and 9 words. Many if the line has more than 9 words. All values have been experimentally set based on the training data. The number of words could help to identify `Body` class, which usually has many words.

- **Paragraph Information:** represents information about the paragraph. There are three possible values: header, new, or same. The header paragraph represents all lines before Abstract or Introduction. "New" represents a new paragraph, and "same" means that the line belongs to the same paragraph as the previous line. This feature has been implemented based on (LUONG; NGUYEN; KAN, 2010). This feature could help identify the transitions between classes. When we have a different class, it is likely for it to have different paragraph information.

- **Formatting Information:** represents information about the line format and can take two different values: new or same. "New" represents that the current line has a different format than the previous one. "Same" represents that the line has exact the same format as the previous one. The format has been defined as a concatenation of: font size, bold, italic, font face, and alignment. This feature has been implemented based on (LUONG; NGUYEN; KAN, 2010). This feature could help to identify the transitions between classes. A new format could be a good indication of a new class.

After defining all the classes and features, the last step of the first-level component consists of defining the CRF++ template file. Listing A.1 shows the complete template for the first-level component. From line 1 to 24 all the features are defined. From line 26 to 52 all the context windows are defined. These windows were set manually based on the training data. Line 55 gives extra weight when both alignment and font size matches in multiple lines (e.g. Title class usually has center alignment and big font size). Similarly, Line 58 gives extra weight when both line identifier and top position matches in multiple lines.

Listing A.1– CRF++ template file for the first-layer model

```
#identifier
U0:%x[0,0]
#alignment
U1:%x[0,1]
# bold
U2:%x[0,2]
# underline
U3:%x[0,3]
# italic
U4:%x[0,4]
# fontSize
U5:%x[0,5]
# top
U6:%x[0,6]
# left
U7:%x[0,7]
# contains @
U8:%x[0,8]
# number of words
U9:%x[0,9]
# paragraph information
U10:%x[0,10]
# format information
U11:%x[0,11]

#font size window
U12:%x[-2,5]
U13:%x[-1,5]

#bold window
U14:%x[-1,2]
U15:%x[-2,2]
U16:%x[1,2]
U17:%x[2,2]

#alignment window
U18:%x[-1,1]
U19:%x[-2,1]
U20:%x[1,1]
U21:%x[2,1]

#paragraph window
U22:%x[-1,10]
U23:%x[-2,10]
U24:%x[1,10]
U25:%x[2,10]

#format window
U26:%x[-1,11]
U27:%x[-2,11]
U28:%x[1,11]
U29:%x[2,11]

# align/font size
U30:%x[0,1]/%x[0,5]

# identifier/top
U31:%x[0,0]/%x[0,6]
```

APPENDIX B DETAILED SECOND-LEVEL FEATURES

B.1 Header CRF Features

- **Word Content:** the original text of the word without spacing. This feature is very useful for identifying the `Publisher` (e.g. ACM, IEEE) and `Conference Names` (e.g. DocEng, WWW). These words tend not to vary (i.e. the same words are used in most papers), thus allowing us to use the word as a feature for the CRF engine. In the first-level of our CRF model, we did not use the value of the line because it is not useful for that level.
- **Word Identifier:** the word identifier, starts with 0 and goes till n, where n is the last word of the line. We believe that the location of the word within the line is relevant for the detection of classes. For example, `Conference Name` is usually one of the first words of the line.
- **Line Identifier:** the line identifier, starts with 0 and goes till n, where n is the last line of the page. This feature could be relevant because `Header` class usually is found in one of the first lines of the page.
- **Character length:** the normalized number of characters for the current word with four possible values: zero, few, medium, or many. The normalization process first extracts the characters for the given word. *Zero* represents an empty word. *Few* if the word between 1 and 4 characters. *Medium* if the word has between 5 and 9 characters. *Many* if the word has more than 9 characters. All values have been experimentally set based on the training data. The number of characters could help identify the `Other` class, which usually has lines full of characters.
- **Numeral:** boolean that represents if the word content is composed only by numbers. This feature is very useful when identifying days and years for the conference/journal. We remove all special characters from the word before checking for the pattern. For example, the word "$5,00" is considered a valid numeral entry in our method because we remove "$" and "," from the character sequence before applying the pattern.
- **Possible Conference:** boolean that represents if the word is a possible conference name by having the substring "conference" or "conf" (e.g. "International Conference on Database Systems"). Also, we currently maintain a list of about 1700 CS conferences around the world. If the given word is found in the conference list,

we also classify this word as a possible conference. Before applying the filter, we convert the word to lowercase.

- **Month:** boolean that identifies if the word represents a month. We keep a list with all possible months in lowercase format. Also, we add the short and full value for the month (e.g. Jun and June). Before performing any checks, we remove all special characters from the word and apply the lowercase function. For example, the word "JuNe," is considered a valid month.

- **Year:** boolean that identifies if the word holds the conference/journal year. The logic executed for this feature is presented in Algorithm 4.

Algorithm 4 Algorithm for the identification of the Year feature

Require: Word without spaces and special characters.

1: $year = false$
2: **if** $isNumberal()$ && $word.length() == 4$ **then**
3: $possibleYear = word.toInt()$
4: **if** $possibleYear > 1850$ && $possibleYear <= CURRENT\_YEAR$ **then**
5: $year = true$
6: **end if**
7: **end if**
8: **return** $year$

The "Year" feature will only be valid if the "Numeral" feature is valid too. Also, the length of the word should be 4, as described in step 2. Another important condition for any article's year is to be in some certain interval. Currenly, we are identifying papers published after 1850 and before the current year (step 4).

- **Country:** boolean that represents if the word is part of a country name. We keep a list of country names in lowercase format. Again, we remove all special characters and convert to lowercase. For example, the word "StaTes..." is a valid country feature. The word "States" is part of the country name "United States of America". This is very important to detect `Conference Location`.

- **Special Character:** boolean that represents if the word contains any special character. This feature is important when identifying the conference location and conference dates.
These classes usually include special characters like comma, dots, and colons. This feature could be useful to identify `Conference Name`, `Conference Location`, and `Conference Date`, as they usually include special characters as part of their data.

- **Website:** boolean that identifies if the word represents a URL (e.g. www.google.com). This feature is useful for `Other` class, if we find a website pattern we currently classify them as `Other`.

The CRF++ template file for the Header layer is shown in Listing B.1. From line 1 to 23 all the features are defined. From line 25 to 27 the country window is defined. As

explained earlier, the conference location is usually of the form "$City, Country$". As a result, we decided to set the country window for two words behind. Line 30 gives extra weight when both numeral and year matches in multiple lines (e.g. good evidence for Conference Year).

Listing B.1– CRF++ template file for the Header layer

```
1: #word
2: U0:%x[0,0]
3: # position withn the line
4: U1:%x[0,1]
5: # line identifier
6: U2:%x[0,2]
7: # number of letters
8: U3:%x[0,3]
9: # is numeral
10: U4:%x[0,4]
11: # is conference
12: U5:%x[0,5]
13: # is month
14: U6:%x[0,6]
15:
16: # possible year
17: U7:%x[0,8]
18: # is a country
19: U8:%x[0,9]
20: # has special character
21: U09:%x[0,10]
22: # website
23: U10:%x[0,11]
24:
25: #country window
26: U11:%x[-2,9]
27: U12:%x[-1,9]
28:
29: # numeral and year
30: U13:%x[0,4]/%x[0,8]
```

B.2 Author Information CRF Features

- **Word Content, Word Identifier, Line Identifier, Character Size, Website:** The same implementation as the Header CRF.
- **Font Size, Possible Email:** implemented following the same algorithm as the one implemented in Appendix A.
- **Possible Affiliation:** boolean that indicates whether the current word is likely to belong to the affiliation block. In order for this feature to be valid, it has to match at least one of these criteria: be a possible university, or a possible country, or a possible department, or a possible continent. To be a possible university, the word should be equal to "University" or "Faculty". The logic for the country is the same that the one executed in the Header CRF. A possible department is a word that matches at least one of these cases: department, dept, center, laboratory, division, school, group, community, or academic. We keep a list of all continents, if the given word matches any of these values it will be classified as a possible affiliation too.
- **Word Format:** represents information about the word formatting. Same implementation as the one describe in A, but for the word-level.

The CRF++ template file for the Author Information layer is shown in Listing B.2. From line 1 to 18 all the features are defined. From line 20 to 25 all windows are defined. The format window, looks at the previous format to infer about the current one. The affiliation window considers the next two words also as candidates for belonging to the same class (e.g. University of California). Line 28 gives extra weight when both format and font size matches in multiple lines.

Listing B.2– CRF++ template file for the Author Information layer

```
1: # word
2: U0:%x[0,0]
3: # position withn the line
4: U1:%x[0,1]
5: # line identifier
6: U2:%x[0,2]
7: # number of letters
8: U3:%x[0,3]
9: # possible email
10: U4:%x[0,4]
11: # possible affiliation
12: U5:%x[0,5]
13: # website
14: U6:%x[0,6]
15: # format
16: U7:%x[0,7]
17: # font size
18: U8:%x[0,8]
19:
20: #format window
21: U9:%x[-1,7]
22:
23: #affiliation window
24: U10:%x[1,5]
25: U11:%x[2,5]
26:
27: #format and font size
28: U12:%x[0,7]/%x[0,8]
```

B.3 Footnote CRF Features

- **Word Content, Word Identifier, Line Identifier, Character Size, Month, Possible Conference, Country, Year, Website, Publisher, Numeral:** the same implementation as described in Header CRF.
- **Possible Email, Possible Affiliation:** the same implementation as described in Author Information CRF.
- **Day:** boolean that indicates if the word matches the following pattern DAY-DAY (e.g. 05-10).
- **ISBN:** boolean that indicates whether the word matches the following ISBN format XXX-X-XXXXX-XX/XX/XX (e.g. 978-1-60558-01/08/04). For some papers, the ISBN comes immediately followed by the fee, like 978-1-60558-01/08/04$5.00. The regular expression applied for this feature also supports ISBN with this particular scenario.

The CRF++ template file for the Footnote layer is shown in Listing B.3. From line 1 to 32 all the features are defined. We did not identify any possible window that could be applied for the Footer section.

Listing B.3– CRF++ template file for the Footnote layer

```
1: # word
2: #U0:%x[0,0]
3: # position withn the line
4: U1:%x[0,1]
5: # line identifier
6: U2:%x[0,2]
7: # characterSize
8: U3:%x[0,3]
9: # month
10: U4:%x[0,4]
11: # conference
12: U5:%x[0,5]
13: # days
14: U6:%x[0,6]
15: # country
16: U7:%x[0,7]
17: # year
18: U8:%x[0,8]
19: # website
20: U9:%x[0,9]
21: # isbn
22: U10:%x[0,10]
23: # publisher
24: U11:%x[0,11]
25: # email
26: U12:%x[0,12]
27: # numberOnly
28: U13:%x[0,13]
29: # possibleAffiliation
30: U14:%x[0,14]
31: # issn
32: U15:%x[0,15]
```

REFERENCES

AUMÜLLER, D. Retrieving Metadata for Your Local Scholarly Papers. In: BTW, 2009. **Proceedings...** [S.l.: s.n.], 2009. p.577–583.

BAGGENSTOSS, P. A modified Baum-Welch algorithm for hidden Markov models with multiple observation spaces. **Speech and Audio Processing, IEEE Transactions on**, [S.l.], v.9, n.4, p.411 –416, may 2001.

BIKEL, D. M. et al. Nymble: a high-performance learning name-finder. In: APPLIED NATURAL LANGUAGE PROCESSING, 1997. **Proceedings...** [S.l.: s.n.], 1997. p.194–201. (ANLC '97).

BRETHERTON, F.; SINGLEY, P. Metadata: a user's view. In: SCIENTIFIC AND STATISTICAL DATABASE MANAGEMENT, 1994. **Proceedings...** [S.l.: s.n.], 1994. p.166 –174.

COUNCILL, I. G.; GILES, C. L.; KAN, M. yen. ParsCit: an open-source crf reference string parsing package. In: LREC, 2008. **Proceedings...** [S.l.: s.n.], 2008.

DARROCH, J. N.; RATCLIFF, D. Generalized iterative scaling for log-linear models. In: THE ANNALS OF MATHEMATICAL STATISTICS, 1972. **Proceedings...** [S.l.: s.n.], 1972. v.43, p.1470–1480.

DO, H. H. N. et al. Extracting and Matching Authors and Affiliations in Scholarly Documents. In: JCDL, 2013. **Proceedings...** [S.l.: s.n.], 2013. p.219–228.

FINK, G. A. **Markov Models for Pattern Recognition**: from theory to applications. Secaucus, NJ, USA: [s.n.], 2007.

FLYNN, P. et al. Automated template-based metadata extraction architecture. In: INTL CONF. ON ASIAN DIGITAL LIBRARIES, 2007. **Proceedings...** [S.l.: s.n.], 2007. p.327–336.

FORNEY G.D., J. The viterbi algorithm. **Proceedings of the IEEE**, [S.l.], v.61, n.3, p.268 – 278, march 1973.

HAN, H. et al. Automatic document metadata extraction using support vector machines. In: JCDL, 2003. **Proceedings...** [S.l.: s.n.], 2003. p.37 – 48.

HETZNER, E. A simple method for citation metadata extraction using hidden markov models. In: JCDL, 2008. **Proceedings...** [S.l.: s.n.], 2008. p.280–284.

HUANG, Z. et al. Header metadata extraction from semi-structured documents using template matching. In: INTEL CONF. ON THE MOVE TO MEANINGFUL INTERNET SYSTEMS: AWESOME, CAMS, COMINF, IS, KSINBIT, MIOS-CIAO, MONET - VOLUME PART II, 2006. **Proceedings...** [S.l.: s.n.], 2006. p.1776–1785.

JOHN LAFFERTY ANDREW MCCALLUM, F. P. Conditional Random Fields: probabilistic models for segmenting and labeling sequence data. **ScholarlyCommons**, [S.l.], 2001.

KLEINBERG, J. M.; TARDOS, É. **Algorithm design**. [S.l.]: Addison-Wesley, 2006. I-XXIII, 1-838p.

LIPINSKI, M. et al. Evaluation of Header Metadata Extraction Approaches and Tools for Scientific PDF Documents. In: JCDL, 2013. **Proceedings...** [S.l.: s.n.], 2013. p.385–386.

LUONG, M.-T.; NGUYEN, T. D.; KAN, M.-Y. Logical Structure Recovery in Scholarly Articles with Rich Document Features. **IJDLS**, [S.l.], v.1, n.4, p.1–23, 2010.

PENG, F.; MCCALLUM, A. Accurate information extraction from research papers using conditional random fields. In: HLT-NAACL04, 2004. **Proceedings...** [S.l.: s.n.], 2004. p.329–336.

PENG, F.; MCCALLUM, A. Information extraction from research papers using conditional random fields. **Inf. Process. Manage.**, [S.l.], v.42, n.4, p.963–979, July 2006.

POTTHAST, M. et al. An Evaluation Framework for Plagiarism Detection. In: COLING, 2010. **Proceedings...** [S.l.: s.n.], 2010. p.997–1005.

PRESS, N. **Understanding Metadata**. [S.l.: s.n.], 2004.

RABINER, L. A tutorial on hidden Markov models and selected applications in speech recognition. **Proceedings of the IEEE**, [S.l.], v.77, n.2, p.257 –286, feb 1989.

RABINER, L.; JUANG, B. An introduction to hidden Markov models. **ASSP Magazine, IEEE**, [S.l.], v.3, n.1, p.4 –16, jan 1986.

ROSENTHOL, L. **Developing with PDF**: dive into the portable document format. 1.ed. [S.l.]: O'REILLY, 2013.

SARAWAGI, S.; COHEN, W. W. Semi-Markov conditional random fields for information extraction. In: IN ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 17, 2004. **Proceedings...** [S.l.: s.n.], 2004. p.1185–1192.

SCHEFFER, T.; DECOMAIN, C.; WROBEL, S. Active Hidden Markov Models for Information Extraction. In: **Advances in Intelligent Data Analysis**. [S.l.]: Springer Berlin Heidelberg, 2001. p.309–318. (Lecture Notes in Computer Science, v.2189).

SCHEFFER, T. et al. **Learning Hidden Markov Models for Information Extraction Actively from Partially Labeled Text**. 2002.

SEYMORE, K.; MCCALLUM, A.; ROSENFELD, R. Learning Hidden Markov Model Structure for Information Extraction. In: AAAI WORKSHOP ON MACHINE LEARNING FOR INFORMATION EXTRACTION, 1999. **Proceedings...** [S.l.: s.n.], 1999. p.37–42.

SKULJ, D. Discrete time Markov chains with interval probabilities. **Int. J. Approx. Reasoning**, [S.l.], v.50, n.8, p.1314–1329, Sept. 2009.

STOLCKE, A. **Bayesian learning of probabilistic language models**. [S.l.: s.n.], 1994.

WALLACH, H. **Efficient Training of Conditional Random Fields**. 2002.

WALLACH, H. M. Conditional Random Fields: an introduction. **ScholarlyCommons**, [S.l.], 2004.

YIN, P. et al. Metadata Extraction from Bibliographies Using Bigram HMM. In: INTL CONF. ON ASIAN DIGITAL LIBRARIES, 2004. **Proceedings...** [S.l.: s.n.], 2004.