École Polytechnique Fédérale de Lausanne

Master Thesis

Automatic Metadata Extaction - The High Energy Physics Use Case

Author:
Joseph Boyd

Supervisor:
Dr. Gilles Louppe

A thesis submitted in fulfilment of the requirements
for the degree of Master of Computer Science

in the

Research Group Name
School of Computer and Communications Sciences

July 2015

Declaration of Authorship

I, Joseph Boyd, declare that this thesis titled, 'Automatic Metadata Extaction - The High Energy Physics Use Case' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

EPFL

Abstract

Faculty Name
School of Computer and Communications Sciences

Master of Computer Science

Automatic Metadata Extaction - The High Energy Physics Use Case

by Joseph Boyd

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor…

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Some stuff about CERN, inspire-hep etc.

Repeat some stuff from Chapter 4 here

1.2 Aims

1.3 Main Results

Repeat stuff from Chapter 5 here

1.4 Outline

In this report we talk about this and that... In Chapter 2 we provide a discussion of the relevant machine learning techniques, along with their algorithms, up to and including the state-of-the-art conditional random fields (CRF).

Chapter 2

Supervised Sequence Learning

In this chapter we present the state-of-the-art technique for metadata extraction, conditional random fields (CRF). For completeness, we include a background history of related machine learning techniques and their associated optimisation algorithms. We begin with a presentation of Hidden Markov Models (HMM) and their inference algorithms. Following this we present multinomial logistic regression. From these former topics we show how their ideas are combined to produce Maximum Entropy Markov Models (MEMM) and CRFs. Notably, we pinpoint the part of the mathematical model relevant to our work on feature engineering. Finally, we describe Wapiti, a general-purpose software engine for training and tagging with CRF models.

2.1 Hidden Markov Models

Hidden Markov models (HMMs) are a staple of natural language processing (NLP) and other engineering fields. A HMM models a probability distribution over an unknown, *hidden* sequence of state variables of length T, $\mathbf{y} = (y_1, y_2, ..., y_T)$, whose elements take on values in a finite set of states, S, and follow a Markov process. For each element in this hidden sequence, there is a corresponding observation element, forming a sequence of *observations*, $\mathbf{x} = (x_1, x_2, ..., x_T)$, similarly taking values in a finite set, O. The graphical structure of a HMM (Figure 2.1) shows the dependencies between consecutive hidden states (these are modelled with *transition* probabilities), and states and their observations (modelled with *emission* probabilities). The first dependency is referred to as the Markov condition, which postulates the dependency of each hidden state, y_t, on its k precursors in the hidden sequence, namely, $\mathbf{y}_{t-k:t-1}$[1]. In the discussion that follows, we assume the

[1] In the following we use the notation $\mathbf{x}_{a:b}$ to refer to the elements of vector $\mathbf{x}$ from index a through b inclusive.

Markov condition to be of first degree, that is, $k = 1$. Incidentally, higher-order HMMs may always be reconstructed to this simplest form (**?**). The second dependency may be referred to as *limited lexical conditioning*, referring to the dependency of an observation only on its hidden state. Properties of the model may then be deduced through statistical inference, for example, a prediction of the most likely hidden sequence can be computed with the Viterbi algorithm (Section 2.2).

HMMs have been shown to be successful in statistical modelling problems. In Part of Speech (PoS) tagging, a classic NLP problem for disambiguating natural language sentences, the parts of speech (nouns, verbs, and so on) of a word sequence (sentence) are modelled as hidden states, and the words themselves are the observations. The PoS sequence may be modelled and predicted for as a HMM. Even a simple HMM can achieve an accuracy of well over 90%. The problem of metadata extraction is clearly similar in form to PoS tagging, as we further show in Chapter 3.

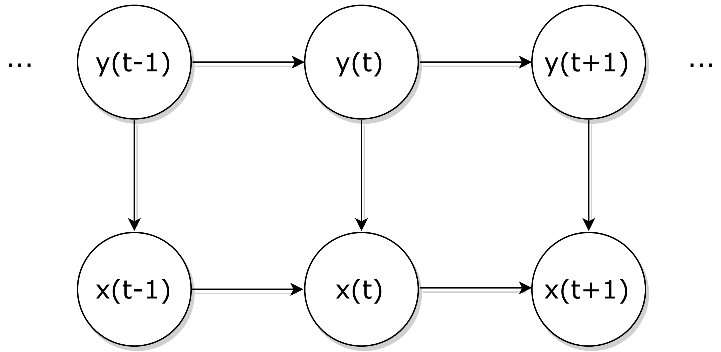

Figure 2.1: An illustration of the graphical structure of a Hidden Markov Model (HMM). The arrows indicate the dependencies running from dependee to dependent.

We may build a HMM by first forming the joint probability distribution of the hidden state sequence and the observation sequence,

$$\mathrm{p}(\mathbf{x}, \mathbf{y}) = \mathrm{p}(\mathbf{x}|\mathbf{y})\mathrm{p}(\mathbf{y}). \tag{2.1}$$

Applying the chain rule and the dependency assumptions, we acquire,

$$\begin{aligned}\mathrm{p}(\mathbf{x}|\mathbf{y}) &= \mathrm{p}(x_1|\mathbf{y})\mathrm{p}(x_2|x_1, \mathbf{y})...\mathrm{p}(x_T|\mathbf{x}_{1:T-1}\mathbf{y}) \\ &= \mathrm{p}(x_1|y_1)\mathrm{p}(x_2|y_2)...\mathrm{p}(x_T|y_T),\end{aligned} \tag{2.2}$$

and,

$$\begin{aligned} \mathrm{p}(\mathbf{y}) &= \mathrm{p}(y_1)\mathrm{p}(y_2|y_1)...\mathrm{p}(y_T, \mathbf{y}_{1:T-1}) \\ &= \mathrm{p}(y_1)\mathrm{p}(y_2|y_1)...\mathrm{p}(y_T, y_{T-1}). \end{aligned} \tag{2.3}$$

Combining 2.2 and 2.3, we may rewrite the factorisation of the HMM as,

$$\mathrm{p}(\mathbf{x}, \mathbf{y}) = \prod_{t=1}^{T} \mathrm{p}(y_t|y_{t-1})\mathrm{p}(x_t|y_t). \tag{2.4}$$

The probabilities $\mathrm{p}(y_t|y_{t-1})$ are known as *transition* probabilities, and $\mathrm{p}(x_t|y_t)$ as *emission* probabilities. These probabilities constitute the model parameters, $\theta = (\mathbf{A}, \mathbf{B}, \mathbf{I})$, where $\mathbf{A}$ is the $|S| \times |S|$ matrix of probabilities of transitioning from one state to another, $\mathbf{B}$ is the $|S| \times |O|$ matrix of probabilities of emitting an observation given an underlying hidden state, and $\mathbf{I}$ is the vector of probabilities of initial states. The model parameters must be precomputed[2]. Now, given a sequence of observations, $\mathbf{x}$, we may predict the hidden state sequence, $\mathbf{y}*$, by maximising the conditional distribution, $\mathrm{p}(\mathbf{y}|\mathbf{x})$. That is,

$$\mathbf{y}* = \operatorname*{argmax}_{\mathbf{y}} \left\{ \prod_{t=1}^{T} \mathrm{p}(y_t|y_{t-1})\mathrm{p}(x_t|y_t) \right\}. \tag{2.5}$$

The hidden state sequence prediction is chosen to be the one maximising the likelihood over all possible hidden sequences. This seemingly intractable problem may be solved in polynomial time using dynamic programming (see Section 2.2).

2.2 Viterbi Algorithm

The Viterbi algorithm is used to efficiently compute the most likely sequence, $\mathbf{y}$, given an observation sequence, $\mathbf{x}$. The algorithm can do this efficiently by working along the sequence from state to state, and choosing the transitions that maximise the likelihood of the sequence fragment. To show this we define, $v_t(s) = \max_{\mathbf{y}_{1:t-1}} \mathrm{p}(\mathbf{y}_{1:t-1}, y_t = s|\mathbf{x})$, that is, the most likely sequence from the first $t-1$ states, with the choice of state s at time t. Thus, we may write,

$$\begin{aligned} v_t(s) &= \max_{\mathbf{y}_{1:t-1}} \mathrm{p}(\mathbf{y}_{1:t-1}|\mathbf{x})\mathrm{p}(y_{t-1}, y_t = s)\mathrm{p}(x_t|y_t = s) \\ &= \max_{\mathbf{y}_{1:t-1}} v_{t-1}(y_{t-1})\mathrm{p}(y_{t-1}, y_t = s)\mathrm{p}(x_t|y_t = s), \end{aligned} \tag{2.6}$$

[2] For example, the model parameters can be estimated through application of the Baum-Welch algorithm on an unsupervised training set.

and we may see the recursion. Once all states have been computed at time t, the maximum may be chosen and the algorithm proceeds to time $t+1$. Pseudocode for the Viterbi algorithm is given in Algorithm 1 in Appendix A. The algorithm must test all $|S|$ transitions from the previous state to each of the $|S|$ current states, and it does that for each of the $|T|$ steps in the sequence. Hence, the complexity of the algorithm is a workable $\mathcal{O}(T|S|^2)$.

2.3 Forward-backward Algorithm

Another key inference algorithm to sequence learning is the forward-backward algorithm, so called for its computation of variables in both directions along the sequence. It is another example of a dynamic programming algorithm and is used to compute the so-called *forward-backward* variables, which are the conditional probabilities of the individual hidden states at each time step (that is, not the whole sequence), given the observation sequence and model parameters, namely, $\mathrm{p}(y_t = s|\mathbf{x}, \theta)$. These conditional probabilities have many useful applications, for example in the Baum-Welch algorithm for estimating model parameters, but also in the training of *conditional random fields*, as we discuss in Section 2.7. We may write the forward-backward variables as,

$$\gamma_t(s) = \mathrm{p}(y_t = s|\mathbf{x}, \theta) = \frac{\alpha_t(s)\beta_t(s)}{\sum_{s' \in S} \alpha_t(s')\beta_t(s')}, \tag{2.7}$$

where the *forward* variables, $\alpha_t(s) = \mathrm{p}(\mathbf{x}_{t+1:n}|y_t = s, \mathbf{x}_{1:t}) = \mathrm{p}(\mathbf{x}_{t+1:n}|y_t = s)$, and the *backward* variables, $\beta_t(s) = \mathrm{p}(y_t = s, \mathbf{x}_{1:t})$. To derive the forward-backward algorithm we write, by the law of total probability,

$$\begin{aligned} \alpha_t(s) &= \sum_{y_{t-1}} \mathrm{p}(y_{t-1}, y_t = s, \mathbf{x}_{1:t}) \\ &= \sum_{y_{t-1}} \mathrm{p}(y_t = s|y_{t-1})\mathrm{p}(x_t|y_t)\mathrm{p}(y_{t-1}, \mathbf{x}_{1:t-1}) \\ &= \sum_{y_{t-1}} \mathbf{A}(y_{t-1}, s)\mathbf{B}(x_t, y_t)\alpha_{t-1}(y_{t-1})). \end{aligned} \tag{2.8}$$

Thus, we may see the recursion, as well as the way the forward variables will be computed, traversing the sequence in the forward direction with each forward variable of a given time a weighted product of those from the previous time. Likewise, for the backward variables, we may write,

$$\begin{aligned}\beta_t(s) &= \sum_{y_{t+1}} \mathrm{p}(y_t = s, y_{t+1}, \mathbf{x}_{t+1:n}) \\ &= \sum_{y_{t+1}} \mathrm{p}(\mathbf{x}_{t+2:n} | y_{t+1}, x_{t+1}) \mathrm{p}(x_{t+1}, y_{t+1} | y_t = s) \\ &= \sum_{y_{t+1}} \beta_{t+1}(y_{t+1}) \mathbf{A}(s, y_{t+1}) \mathbf{B}(x_{t+1}, y_{t+1})).\end{aligned} \tag{2.9}$$

From Equations 2.8 and 2.9 comes Algorithm 2 (Appendix A). The complexity of the algorithm comes from noting that at each of the T steps in the sequence (in either direction), we compute $|S|$ variables, involving a summation of $|S|$ products. Hence, like the Viterbi algorithm, the complexity of the forward-backward algorithm is $\mathcal{O}(T|S|^2)$.

2.4 Maximum Entropy Classifiers

Maximum entropy classifiers, also known as multinomial logistic regression, are a family of classification techniques. A prediction is a discrete (categorical), scalar *class*, rather than a class sequence as it is for HMMs. To build a model, we require a *training set* consisting of a $N \times D$ matrix, $\mathbf{X}$, of N training samples of dimension D[3], as well as the N corresponding classifications in the form of a vector, $\mathbf{y}$. A convex cost function known as a maximum log-likelihood function is constructed and subsequently optimised over the choice of model parameters, denoted $\boldsymbol{\beta}$. Thus, building a model is equivalent to solving a convex optimisation problem. A classification (prediction), $y*$, for an unseen data sample, $\mathbf{x}$, is made by employing these optimal model parameters in a linear function. The result is then passed through a non-linear *logistic* function, denoted σ, to obtain a probability. Formally,

$$y* = \sigma(\boldsymbol{\beta}^T \mathbf{x}). \tag{2.10}$$

The simplest form of maximum entropy classifier is binary logistic regression, where the number of classes to predict from is two, denoted C_1 and C_2. In this case, $\mathrm{p}(y_n = C_1 | \mathbf{x}_n; \boldsymbol{\beta}) = \sigma(\boldsymbol{\beta}^T \mathbf{x}_n)$, and $\mathrm{p}(y_n = C_2 | \mathbf{x}_n; \boldsymbol{\beta}) = 1 - \sigma(\boldsymbol{\beta}^T \mathbf{x}_n)$, where C_1 and C_2 are encoded as 0 and 1 respectively. Notice the probabilities sum to 1. Now, the log-likelihood can be expressed as,

[3] The dimensions of a model is synonymous with the model fields or features.

$$\log \mathrm{p}(\mathbf{y}|\mathbf{X},\boldsymbol{\beta}) = \log \prod_{n=1}^{N} \mathrm{p}(y_n,\mathbf{x}_n) = \log \left(\prod_{n:y_n=C_1}^{N} \sigma(\boldsymbol{\beta}^T\mathbf{x}_n) \prod_{n:y_n=C_2}^{N} 1-\sigma(\boldsymbol{\beta}^T\mathbf{x}_n) \right)$$
$$= \log \prod_{n=1}^{N} \sigma(\boldsymbol{\beta}^T\mathbf{x}_n)^{y_i}(1-\sigma(\boldsymbol{\beta}^T\mathbf{x}_n))^{1-y_i} \tag{2.11}$$

where $y_i \in \{0,1\}$. We may then generalise to,

$$\log \mathrm{p}(\mathbf{y}|\mathbf{X},\boldsymbol{\beta}) = \log \prod_{n=1}^{N}\prod_{c=1}^{C} \mu_{nc}^{y_{nc}}, \tag{2.12}$$

where $y_{nc} = \mathbb{1}_{\{y_n=c\}}$ and y_n is a bit vector indicating the class of the nth sample. In this general, multinomial case, the probabilities are written, $\mu_{nc} = \frac{\exp(\boldsymbol{\beta}_c^T\mathbf{x}_n)}{\sum_{c'=1}^{C}\exp(\boldsymbol{\beta}_c^T\mathbf{x}_n)}$, which are normalised to ensure they sum to 1, and $\boldsymbol{\beta}_c$ is part of a set of C parameter vectors notated as $D \times C$ matrix, $\mathbf{B}$. From this we obtain a cost function,

$$\mathcal{L}(\mathbf{B}) = \log \mathrm{p}(\mathbf{y}|\mathbf{X},\mathbf{B}) = \sum_{n=1}^{N} \left(\sum_{c=1}^{C} y_{nc}\boldsymbol{\beta}_c^T\mathbf{x}^{(n)} \right) - \log \left(\sum_{c'=1}^{C} \exp(\boldsymbol{\beta}_{c'}^T\mathbf{x}^{(n)}) \right). \tag{2.13}$$

Now we require an optimisation algorithm to solve for $\mathbf{B}$.

2.5 L-BFGS

Convex optimisation problems may be solved numerically using variants of the method of greatest descent. These methods find the optimal model parameters by iteratively approaching a global minimum by taking steps opposite the gradient along the cost function hypersurface. The classic first-order gradient descent algorithm defines its iteration step to be,

$$\boldsymbol{\beta}^{k+1} = \boldsymbol{\beta}^{k} - \alpha\nabla\mathcal{L}(\boldsymbol{\beta}^{k}), \tag{2.14}$$

where $\boldsymbol{\beta}$ is the vector of model parameters, α is the step size, and $\mathcal{L}$ is the cost function. Newton's method (also known as Iterated Reweighted Least Squares (IRLS)) takes a step in the direction minimising a second-order approximation of the cost function,

$$\boldsymbol{\beta}^{k+1} = \boldsymbol{\beta}^{k} - \alpha_k \mathbf{H_k}^{-1} \nabla \mathcal{L}(\boldsymbol{\beta}^{k}), \tag{2.15}$$

where **H** is the $(D \times D)$ Hessian matrix of partial second derivatives. For smaller problems, these algorithms are adequate, however for models with millions of features, such as those that may be encountered in metadata extraction, smarter approaches are required. The *Broyden–Fletcher–Goldfarb–Shanno* (BFGS) algorithm saves on the expensive computation of the Hessian by building up an approximation iteratively. The *limited memory* BFGS (L-BFGS) algorithm makes further savings on the Hessian's *storage*, and has come to be the standard learning algorithm for such problems. The L-BFGS algorithm is the tool of choice for many problems (Murphy [2012]) and is the algorithm we use in our analysis.

2.6 Regularisation

To avoid overfitting, we add a penalty to the cost function. This imposes a cost proportional to the size of the parameters for each dimension. Large parameters are therefore discouraged and this helps prevent the creation of complex models during training that do not fit test data well. This is equivalent to adding constraints to the optimisation problem. The two most common regularisation types are known as l_1 ad l_2 regularisation. The former imposes a *Laplace* prior distribution on the parameters, and the latter a *Gaussian*[4]. According to a probabilistic interpretation, this makes large parameters less likely and moderates their choices, and this is expressed in the cost function as the penalty. In our work, l_2 is the only type of regularisation compatible with L-BFGS (Section 2.5).

2.7 Conditional Random Fields

Conditional Random Fields (CRFs) are a machine learning technique for making structured predictions. They are an improvement to the similar, Maximum Entropy Markov models (MEMM) (McCallum et al. [2000]), which combine aspects of maximum entropy classifiers and Hidden Markov models (Lafferty et al. [2001]). They are a member of a class of structured sequence models called *random fields*, which are part of a broader family known as *graphical models*, including within it *Bayesian networks*.

Classification over relational data can benefit greatly from rich features, that is, describing observed attributes of an observation beyond merely its identity (as with HMMs).

[4] A prior distribution in the Bayesian sense is the initial distribution of a variable taken independently.

Take for example the context of text processing, where we might consider describing a string token (observation) by non-lexical features such as by its capitalisation or punctuation. Furthermore, we may wish to model context-aware features that contrast a string token with its surroundings. However, the complexity of the interdependencies of such features will likely make their explicit modelling infeasible. With CRFs, we circumvent this problem by instead modelling the conditional distribution, $\mathrm{p}(\mathbf{y}|\mathbf{x})$, of the underlying graph structure, giving us free choice over features and, in so doing, *implicitly* defining a distribution over $\mathbf{x}$ without having to model this distribution directly (Sutton and McCallum [2006]). Such a conditional model is called a *discriminative* model, in contrast to a *generative* model, whereby the joint probability distribution is modelled explicitly. If we wish to model the interdependencies in a generative model, we must either extend the model which may both be difficult and entail intractable solution algorithms, or we must simplify the model and thereby compromise model performance. Notice that modelling the conditional distribution is sufficient for classification, where the observation sequence is known. This freedom for rich feature engineering is what makes CRFs the current state-of-the-art in metadata extraction, where arbitrarily defined features often make for good indicators. One may be tempted to use a logistic regression and classify each part of a sequence separately, but this would fail to take into account the contextual relations between the entities. For example, in the metadata extraction of a bibiographic reference, it is more likely for a publication title to follow an author list, and for a journal name to follow a publication title. This is what we mean by structured sequence learning, where the data to predict exhibits interdependencies and are correlated.

When the graph structure of a CRF model is the same as for a HMM (Figure 2.1), we have what is called a *linear-chain* CRF. HMMs and linear-chain CRFs thereby form what is called a generative-discriminative pair. In the general case, where the graph structure is more complex, we have what is called *skip-chain* CRFs. In this case the problem becomes far more complex, and we will not discuss these models here. A HMM may alternatively be expressed by the joint probability,

$$\mathrm{p}(\mathbf{x}, \mathbf{y}) = \exp\left\{ \sum_{i,j \in S} \lambda_{ij} F_{i,j}(y_t, y_{t-1}, x_t) + \sum_{i \in S} \sum_{o \in O} \mu_{io} F_{i,o}(y_t, y_{t-1}, x_t) \right\}, \tag{2.16}$$

where the parameters λ_{ij} are the transition probabilities and μ_{ij} are the emission probabilities. $F_{i,j}(y_t, y_{t-1}, x_t) = \sum_t^T \mathbb{1}_{\{y_t = i\}} \mathbb{1}_{\{y_{t-1} = j\}}$ is a *feature function* used to activate the transition probabilities, and $F_{i,o}(y_t, y_{t-1}, x_t) = \sum_t^T \mathbb{1}_{\{y_t = i\}} \mathbb{1}_{\{x_i = o\}}$ for the emissions. The indicator functions activate the probabilities in accordance with the identity of the states and observations. Regardless, this formulation is equivalent to Equation 2.4. With some notational abuse we can define the more compact expression,

$$\mathrm{p}(\mathbf{x},\mathbf{y}) = \exp\left\{\sum_k \lambda_k F_k(y_t, y_{t-1}, x_t)\right\}, \tag{2.17}$$

where F_k is a general feature function and λ_k a general feature weight. Now we may define the discriminative counterpart to this joint distribution, the linear-chain CRF,

$$\mathrm{p}(\mathbf{y}|\mathbf{x}) = \frac{\mathrm{p}(\mathbf{x},\mathbf{y})}{\sum_{y'} \mathrm{p}(\mathbf{x},\mathbf{y}')} = \frac{1}{Z(\mathbf{x})} \exp\left\{\sum_k \lambda_k F_k(y_t, y_{t-1}, x_t)\right\}, \tag{2.18}$$

where $Z(\mathbf{x}) = \sum_{y'} \exp\left\{\sum_k \lambda_{ij} F_k(y'_t, y'_{t-1}, x_t)\right\}$ is known as the partition function, ensuring probabilities sum to 1. Whereas HMMs model only the *occurence* of a word, with conditional random fields we may choose F_k to define arbitrarily complex features, describing rich information about a word, its attributes, and its context. Finally, we may define a cost function in the following way,

$$l(\theta) = \sum_{n=1}^{N}\sum_{t=1}^{T}\sum_{k=1}^{K} \lambda_k F_k(y_t, y_{t-1}, x_t) - \sum_{n=1}^{N} \log Z(\mathbf{x}^{(n)}) - \sum_{k=1}^{K} \frac{\lambda_k^2}{2\sigma^2}, \tag{2.19}$$

where $\theta = \{\lambda_k\}_{k=1}^{K}$. This is called penalised maximum log likelihood. The penalty term, $\sum_{k=1}^{K} \frac{\lambda_k^2}{2\sigma^2}$, imposes an l_2 regularisation on the solution parameters, θ. However, according to (McCallum et al. [2000]), varying the tuning parameter, σ^2 (the variance), by orders of magnitude has little effect on the outcome, a claim we corroborate in Chapter 5. This cost function represents a strictly convex function, solvable using numerical methods such as L-BFGS (see Section 2.5). Forward-backward processing (Section 2.3) is performed at each iteration to compute the partition function, as well as the conditional probabilities resulting from deriving the partial derivatives required for gradient descent. Finally, the Viterbi algorithm (Section 2.2) is used to make a prediction with the trained model, that is, the best state sequence is found given the optimal parameter set found in training. A detailed exposition of this is given in (McCallum et al. [2000]).

2.8 Feature Functions

In the simple HMM case (Equation 2.16), there is a single feature function for each (i, j) and (i, o) pair. Furthermore, they are merely indicator functions that facilitate the activation of the transition and emission probabilities. In CRFs, however, we may define arbitrarily many and varied feature functions of the form $F_k(\mathbf{x}, y) = \sum_t^T f_k(\mathbf{x}, y)$, where f_k is a (typically boolean) function describing one of several features about a token.

Notice that while such a function is centered on a given token, that is, a specific element of $\mathbf{x}$, the function has access to the full vector $\mathbf{x}$, enabling the creation of context-aware features, combining information about a token with its neighbours. Further notice that the summation over the sequence is what enables instances (token sequences) of varying lengths to remain compatible with the model.

The form of the functions themselves, $f(\cdot)$, are known in Wapiti (Section 2.9) as *templates*. It is in choosing these explicitly that we perform feature engineering. The literal feature functions, $\{f_k\}_{k=1}^{K}$, are formed by resolving the templates over the *vocabularly* of features encountered in the extraction process prior to model training. In this way, we may see how model complexity depends on the diversity of the training set, and consequently, for smaller training sets, a model will have fewer feature functions.

2.9 Wapiti

There are several open source software packages for the general purpose training and application of conditional random fields and related models. Wapiti (Lavergne et al. [2010]), written in C, is the tool of choice for this project, given its compatibility with metadata extraction tool GROBID (Section 3.3), its speed advantage over alternatives, and the recency of its development. It is developed by Thomas Lavergne at LIMSI, a computer science laboratory in Orsay affiliated with Paris-Sud University. It is capable, given sufficient memory, of training models with thousands of classes and billions of features. It implements several optimisation algorithms including L-BFGS (Section 2.5) and stochastic gradient descent (SGD) and training is fully parallelisable. Wapiti has few drawbacks, but one is surely its lack of support for numeric features, as this curtails the scope for our feature engineering; any numeric-based idea must be discretised. Wapiti's main functions are training models and tagging. Training requires two inputs:

1. a feature template file, and;
2. a file of extracted features.

The output of training is a model file. Tagging requires three inputs:

1. a feature template file;
2. a file of extracted features, and;
3. a trained model.

```
# Capitalization
U50:%x[0,11]
U51:%x[1,11]
U52:%x[-1,11]
U53:%x[0,11]/%x[1,11]
U54:%x[-1,11]/%x[0,11]
```

FIGURE 2.2: Excerpt of capitalisation features templates or *macros*.

The output of tagging is the file of extracted features appended with the classifications of each token. In the following we present samples of each of these files as they may look for a simple *date* model for classifying dates into their *day*, *month*, and *year* components.

2.9.1 Feature Templates

Feature templates are the main access point for modelling with Wapiti. These files use a special syntax introduced by an older CRF engine, *CRF++* (also supported by GROBID), allowing the operator to specify the form of the feature functions to be implemented in the model (see Section 2.8). The features are listed in a manner such as seen in Figure 2.2. The five features shown in Figure 2.2 follow a similar format. The prefixes 'U50', 'U51' etc. are the unique identifiers of the macros. The '%x' figures are wildcards for literal tokens. These macros are ultimatley expanded to feature functions when they are combined with the extracted features shown in Figure 2.3. The indices given in square brackets indicate the row and column offset of the features considered. For example, '[0, 11]' in macro 'U50' indiciates a row offset of 0, that is, pertaining to the current token, and a column offset of 11, pertaining to the $11th$ feature extracted in the feature extraction file (Section 2.9.2). Finally, macros 'U53' and 'U54', combine features from past and future tokens with the current one to make bigram features.

2.9.2 Extracted Features

The extracted features file give the raw features for individual tokens. Note that feature templates may combine the raw features to make other, more complex features. Each line corresponds to a single token within each instance, and instances are grouped and separated by a line space. Figure 2.3 shows the features for a single instance of a date sequence, the string '4 August 1989'. The features for each token range from the original token (corresponding to a simple token indicator feature function such as in Equation 2.16), to token prefixes [5], to information about capitalisation and punctation, and so

[5] Prefixes are best seen for the 'August token ('A', 'Au', etc.); for the token '4', prefixes are identical to the original token.

```
4 4 4 4 4 4 LINESTART NOCAPS ALLDIGIT 1 0 0 NOPUNCT I-<day>
August august A Au Aug Augu LINEIN INITCAP NODIGIT 0 0 1 NOPUNCT I-<month>
1989 1989 1 19 198 1989 LINEEND NOCAPS ALLDIGIT 0 1 0 NOPUNCT I-<year>
```

Figure 2.3: Features for a single date instance of three tokens: '4 August 1989'.

```
10:u50:NOCAPS,
11:u51:INITCAP,
11:u52:INITCAP,
18:u53:NOCAPS/INITCAP,
18:u54:INITCAP/NOCAPS,
```

Figure 2.4: Expanded feature functions deriving from capitalisation macros.

on. Finally, we see the classifications of those tokens as 'I-<day>', 'I-<month>', and 'I-<day>'.

2.9.3 Models

In Wapiti, a model consists of a large text file adhering to the following structure:

1. A list of the macros used (taken from the feature template file);
2. A list of classes modelled;
3. A list of expanded feature functions, and;
4. A list of corresponding (non-zero) weights, that is, the model parameters, represented in hexadecimal notation[6].

Of most interest are the expanded feature functions[7], such as shown in Figure 2.4. For example, feature function 'u50' is a binary indicator for the capitalisation of the token. If the corresponding feature for this token is 'NOCAPS', the result will be '1', otherwise, '0'. These functions are derivations of the macros defined in Figure 2.2.

2.9.4 Training

Training a model with Wapiti once all the input files have been prepared. The output given in Figure 2.5 shows the first six iterations of L-BFGS optimisation for training a *date* model. In this case the number of instances ('nb train'), $N = 493$. The figure 'nb blocks' refers to the number of feature functions per class that have come from combining extracted features (Section 2.9.2) and feature templates (Section 2.9.1). The total number of feature functions ('nb features') is therefore this number multiplied by the

[6] This presumably to avoid numeric underflow.

[7] Note the initial values of each line are simply the line lengths as a convenience to input processing.

```
* Initialize the model
* Summary
    nb train:    493
    nb labels:   7
    nb blocks:   5816
    nb features: 40754
* Train the model with l-bfgs
  [   1] obj=1688,58    act=16482    err=25,80%/50,91% time=0,08s/0,08s
  [   2] obj=1221,30    act=15580    err=19,11%/35,50% time=0,05s/0,12s
  [   3] obj=922,15     act=13869    err=17,20%/33,67% time=0,04s/0,17s
  [   4] obj=638,04     act=10845    err= 6,53%/15,21% time=0,04s/0,20s
  [   5] obj=478,72     act=10582    err= 5,68%/13,59% time=0,04s/0,24s
  [   6] obj=416,15     act=9926     err= 3,77%/ 9,53% time=0,04s/0,28s
```

Figure 2.5: Output from training date model

number of classes, plus the number of transition functions, hence, $5816 \times 7 + 7 \times 6 = 40754$ features in total. Training a model to be sufficiently accurate generally takes hundreds or even thousands of iterations of L-BFGS.

Chapter 3

Automatic Metadata Extraction

In this chapter we define the problem of automatic metadata extraction and discuss the methods by which the problem may be solved. Moreover, we introduce GROBID, the metadata extraction tool around which our work is based, and describe the cascade of CRF models it uses to solve the problem. Notably, we provide a comparison between GROBID and the existing solution for metadata extraction at CERN, 'REFEXTRACT'.

3.1 Metadata Extraction

Automatic metadata extraction (AME) has been referred to as 'one of the hardest problems in document engineering'. In our work we are concerned with extraction for scientific articles that are usually (though not necessarily) in the form of a PDF document, as these predominate in the INSPIRE-HEP digital library. Nevertheless, the same techniques will be effective for books, theses, or and even have novel applications[1]. At CERN, the problem has been partially solved, albeit in a rudimentary way 3.3.2, and also entails a lot of manual curation to complete the work.
Definition 3.1. Here is a definition.

Metadata refers to various information explicitly or implicitly contained in

Extraction can refer to two distinct concepts:

1. The parsing of a PDF document and extraction of plaintext and images

2. The *classification* of document text into predefined categories.

[1] Such as for segmenting cooking recipes, as reported in The New York Times (http://open.blogs.nytimes.com/2015/04/09/extracting-structured-data-from-recipes-using-conditional-random-fields/?_r=0).

It is on the second definition that we are concerned with in this work, despite the fact that GROBID as a tool addresses both of these problems.

To appreciate the difficulty of automating such a task, consider Figure X . Though the same sorts of information are there. The contents of a document header are unpredictable, which makes the problem hard, but are not entirely random, a condition that would render the problem impossible.

3.2 Solutions Methods

Mention here the different solution types. Also mention the paper comparison used to select GROBID.

3.3 GROBID

GROBID (https://github.com/kermitt2/grobid) is a Java-based tool for automatic metadata extraction

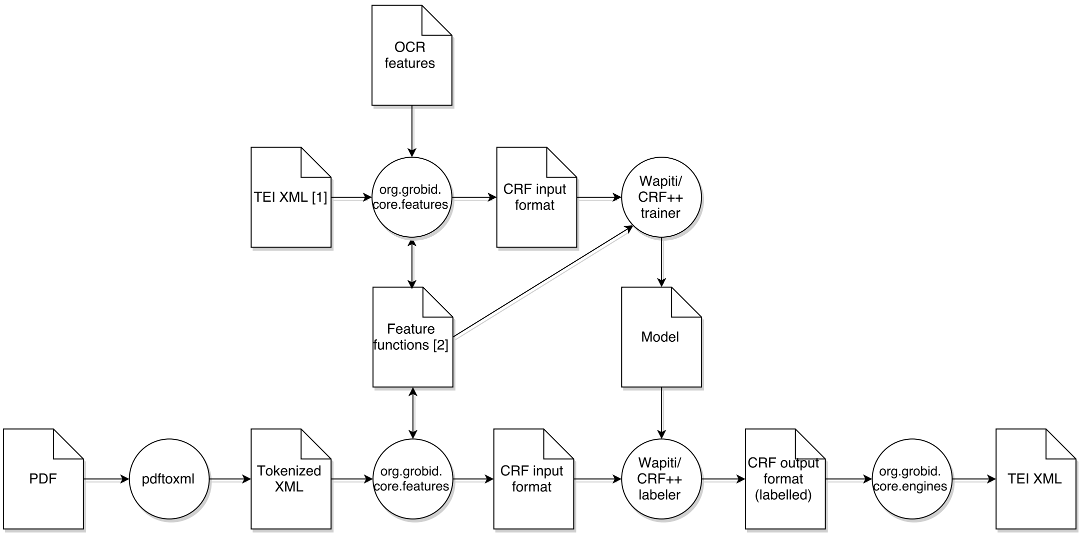

Figure 3.1: An illustration of the interactions between Grobid and Wapiti for the two main functions of training and tagging.

3.3.1 Text Encoding Initiative

One of GROBID's features is to transform Wapiti outputs into a standard output conforming to the Text Encoding Initiative, therefore we briefly describe it here. The Text Encoding Initiative (TEI) is a standard encoding format for describing documents of

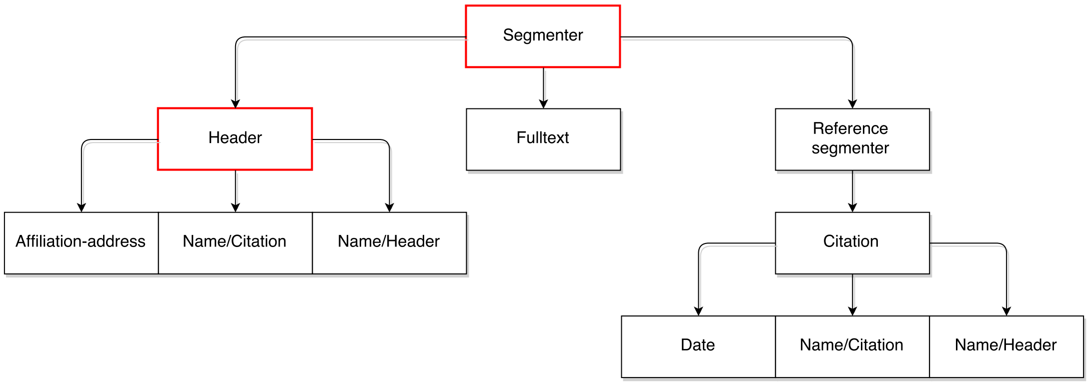

FIGURE 3.2: The models of Grobid are organised into a cascade, where each part of a document is classified in increasingly greater detail.

any kind. In Figure Y we show a sample of a TEI document used to represent a date. The structured XML format gives information about the class of each of the tokens in a sequence.

3.3.2 Comparison with REFEXTRACT

label	accuracy	precision	recall	f1
<label>	99.96	100	99.2	99.6
<reference>	99.96	99.96	100	99.98
(micro average)	99.96	99.96	99.96	99.96
(macro average)	99.96	99.98	99.6	99.79

TABLE 3.1: Evaluation results for reference segmentation

	GROBID				REFEXTRACT			
label	accuracy	precision	recall	f1	acc.	prec.	rec.	f1
<author>	99.85	99.68	99.75	99.72	98.33	100	92.22	95.95
<title>	99.59	98.87	99.25	99.06	94.89	100	71.75	83.55
<journal>	98.84	88.87	93.98	91.35	97.12	100	46.78	63.74
<volume>	99.95	99.07	98.15	98.6	98.36	0	0	0
<issue>	99.93	100	94.63	97.24	98.87	0	0	0
<pages>	99.75	93.51	99.45	96.39	97.26	0	0	0
<date>	98.39	57.39	98.31	72.47	98.88	100	37.55	54.6
<pubnum>	98.71	100	12.96	22.95	98.77	0	0	0
<note>	99.4	43.75	35	38.89	99.55	0	0	0
<publisher>	99.81	63.46	94.29	75.86	99.73	0	0	0
<location>	99.81	86.32	91.11	88.65	99.32	0	0	0
<institution>	99.78	25	25	25	99.88	0	0	0
<booktitle>	98.7	55.56	41.67	47.62	98.82	0	0	0
<web>	99.64	51.85	100	68.29	99.68	0	0	0
<editor>	99.93	100	46.67	63.64	99.89	0	0	0
<tech>	99.95	83.33	50	62.5	99.92	0	0	0
(micro average)	99.5	93.63	94.77	94.19	98.7	100	63.47	77.65
(macro average)	99.5	77.92	73.76	71.76	98.7	25	15.52	18.62

Table 3.2: Evaluation results for citations

Chapter 4

Implementation and Data

4.1 Extensions

4.2 Data Acquisition

4.3 Feature Engineering

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if min(i, j)} = 0 \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j)+1 \\ \mathrm{lev}_{a,b}(i,j-1)+1 \\ \mathrm{lev}_{a,b}(i-1,j-1)+1_{a_i \neq b_j} \end{cases} & \text{otherwise} \end{cases}$$

Radar plots here

Chapter 5

Results and Analysis

5.1 Experiment Setup

Here talk about k-fold, number of iterations, computing resources etc.

5.2 Evaluation Method

Include somewhere a table of all experiments run

5.3 Baseline

5.4 Regularisation

5.5 Dictionaries

5.6 Dictionaries + Stop Words

5.7 Token Selection

5.8 Levenshtein

5.9 Line Shape

5.10 Character Classes

Chapter 6

Conclusion

6.1 Summary

6.1.1 Key Results

6.2 Future Work

Appendix A

Algorithms

```
Data: Observation sequence, x, and model parameters, $\theta = (\mathbf{A}, \mathbf{B}, \mathbf{I})$
Result: Most likely sequence, y*
Initialise y* as a zero-length sequence for $s \in S$ do
    $v_1(s) = \mathbf{I}(s) \times \mathbf{B}(x_1, s)$
end
for $t = 2$ to $T$ do
    for $s \in S$ do
        $v_t(s) = \max_{s'}(\mathbf{A}(s', s) \times v_{t-1}(s')) \times \mathbf{B}(x_t, s)$
        Append s to y*
    end
end
Return y*
```

Algorithm 1: The Viterbi algorithm ($\mathcal{O}(T|S|^2)$) for computing the most likely hidden sequence for a given observation sequence of an HMM.

```
Data: Observation sequence, x, and model parameters, $\theta = (\mathbf{A}, \mathbf{B}, \mathbf{I})$
Result: Set of forward variables, $\{\alpha_t(s)\}_{s \in S, t \in T}$, and backward variables,
        $\{\beta_t(s)\}_{s \in S, t \in T}$
for $s \in S$ do
    $\alpha_1(s) = B(x_1, s) \times \mathbf{I}(s)$
    for $t$ = 2 to $T$ do
        $\alpha_t(s) = \sum_{s'} \mathbf{A}(s, s') \times \mathbf{B}(x_t, s) \times \alpha_{t-1}(s')$
    end
end
for $s \in S$ do
    $\beta_T(s) = 1$
    for $t$ = T-1 to 1 do
        $\beta_t(s) = \sum_{s'} \beta_{t+1}(s') \times \mathbf{A}(s, s') \times \mathbf{B}(x_t, s)$
    end
end
Return the sets of backward and forward variables
```

Algorithm 2: The forward-backward algorithm - $\mathcal{O}(T|S|^2)$

Bibliography

Kevin P Murphy. *Machine learning: a probabilistic perspective.* MIT press, 2012.

Andrew McCallum, Dayne Freitag, and Fernando CN Pereira. Maximum entropy markov models for information extraction and segmentation. In *ICML*, volume 17, pages 591–598, 2000.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. *Introduction to statistical relational learning*, pages 93–128, 2006.

Thomas Lavergne, Olivier Cappé, and François Yvon. Practical very large scale CRFs. In *Proceedings the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 504–513. Association for Computational Linguistics, July 2010. URL http://www.aclweb.org/anthology/P10-1052.