

Automatic Metadata Extraction: The High Energy Physics Use Case

Joseph Boyd

July 5, 2015

Contents

1	Introduction	3
1.1	Motivation	3
1.2	Aims	3
1.3	Main Results	3
1.4	Outline	3
2	Supervised Sequence Learning	4
2.1	Hidden Markov Models	4
2.2	Viterbi Algorithm	5
2.3	Forward-backward Algorithm	5
2.4	Maximum Entropy Classifiers	6
2.5	L-BFGS	6
2.6	Maximum Entropy Markov Models	6
2.7	Conditional Random Fields	7
2.8	Log-linear Models	7
2.9	Graphical Models	7
2.9.1	Hidden Markov Models	7
2.10	Conditional Random Fields	7
2.10.1	Feature Engineering	7
2.10.2	Wapiti	7
3	Automatic Metadata Extraction	9
3.1	Metadata Extraction	9
3.2	Related Work	9
3.3	GROBID	9
4	Implementation and Data	10
4.1	Extensions	10
4.2	Data Acquisition	10
5	Results and Analysis	10
5.1	Experiment Setup	10
5.2	Evaluation Method	12
5.3	Baseline	12
5.3.1	Header model - Cora dataset	12
5.3.2	Header model - Cora dataset appending HEP dataset	12
5.3.3	Header model - Cora and HEP combined datasets	12
5.3.4	Header model - HEP dataset	12
5.3.5	Header model - HEP dataset appending CORA dataset	12

5.3.6	Header model - HEP dataset appending 1/3 CORA dataset	12
5.3.7	Header model - HEP dataset appending 2/3 CORA dataset	12
5.3.8	Segmentation model - Cora dataset	12
5.3.9	Segmentation model - Cora dataset appending HEP dataset	12
5.3.10	Segmentation model - Cora and HEP combined datasets	12
5.3.11	Segmentation model - HEP dataset	12
5.3.12	Segmentation model - HEP dataset appending CORA dataset	12
5.4	Regularisation	12
5.4.1	Header model - $L2 = 0$	12
5.4.2	Header model - $L2 = 1e^{-6}$	12
5.4.3	Header model - $L2 = 1e^{-5}$	12
5.4.4	Header model - $L2 = 1e^{-4}$	12
5.4.5	Header model - $L2 = 1e^{-3}$	12
5.5	Dictionaries	12
5.5.1	Header model - HEP dataset	12
5.5.2	Header model - HEP dataset appending CORA dataset	12
5.5.3	Segmentation model - HEP dataset	12
5.5.4	Segmentation model - HEP dataset appending CORA dataset	12
5.5.5	Header Model - HEP dataset - 2^{nd} Degree Features	12
5.5.6	Header Model - HEP dataset Appending CORA - 2^{nd} Degree Features	12
5.5.7	Header Model - HEP dataset - 3^{rd} Degree Features	12
5.5.8	Header Model - HEP dataset Appending CORA - 3^{rd} Degree Features	12
5.6	Dictionaries + stop words	12
5.6.1	Header model - HEP dataset	12
5.6.2	Header model - HEP dataset appending CORA dataset	12
5.6.3	Segmentation model - HEP dataset	12
5.6.4	Segmentation model - HEP dataset appending CORA dataset	12
5.6.5	Header Model - HEP dataset - 2^{nd} Degree Features	12
5.6.6	Header Model - HEP dataset Appending CORA - 2^{nd} Degree Features	12
5.6.7	Header Model - HEP dataset - 3^{rd} Degree Features	12
5.6.8	Header Model - HEP dataset Appending CORA - 3^{rd} Degree Features	12
5.7	Token Selection	12
5.7.1	Segmentation Model - HEP dataset - 5 Tokens	12
5.7.2	Segmentation Model - HEP dataset - 10 Tokens	12
5.7.3	Segmentation Model - HEP dataset - 15 Tokens	12
5.7.4	Segmentation Model - HEP dataset - 20 Tokens	12
5.8	Levenshtein	12
5.8.1	Segmentation Model - HEP dataset - Binary Threshold (0.05)	12
5.8.2	Segmentation Model - HEP dataset - Binary Threshold (0.1)	12
5.8.3	Segmentation Model - HEP dataset - Binary Threshold (0.2)	12
5.8.4	Segmentation Model - HEP dataset - Binary Threshold (0.4)	12
5.8.5	Segmentation Model - HEP dataset - Binary Threshold (0.8)	12
5.8.6	Segmentation Model - HEP dataset - Ternary Threshold	12
5.8.7	Segmentation Model - HEP dataset - Quaternary Threshold	12
5.9	Line Shape	12
5.9.1	Segmentation Model - HEP dataset - Binary Threshold	12
5.9.2	Segmentation Model - HEP dataset - Ternary Threshold	12
5.10	Template Matching	12
5.10.1	Segmentation Model - HEP dataset	12

6 Conclusion 12

6.1	Summary	12
6.1.1	Key Results	12
6.2	Future Work	12

7 References 13

1 Introduction

1.1 Motivation

1.2 Aims

1.3 Main Results

1.4 Outline

2 Supervised Sequence Learning

In this section 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 optimisation algorithms. We begin with a presentation of Hidden Markov Models (HMM) and their inference algorithms. Following this we present multinomial classifiers, that is scalar classifiers. 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 applying CRF models.

2.1 Hidden Markov Models

Hidden Markov models (HMMs) are a staple of natural language processing (NLP) and other engineering fields. An HMM models a probability distribution over an unknown, “hidden” sequence state variables of length T , $\mathbf{y} = (y_1, y_2, \dots, y_T)$, 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”, $\mathbf{x} = (x_1, x_2, \dots, x_T)$, similarly taking values in a finite set, O . The graphical structure of an HMM (Figure 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, $\mathbf{y}_{t-k:t-1}$. In the discussion that follows, we assume the first-degree Markov condition (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 the most likely hidden sequence can be computed with the Viterbi algorithm (section 2.2).

HMMs have shown to be successful in statistical modelling problems. In Part of Speech (PoS) tagging, a classic NLP problem for disambiguating natural language, the parts of speech (nouns, verbs, and so on) of a word sequence (sequence) are modelled as hidden states, and the words themselves are the observations. The PoS sequence may be modelled and predicted using an 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 section 3.

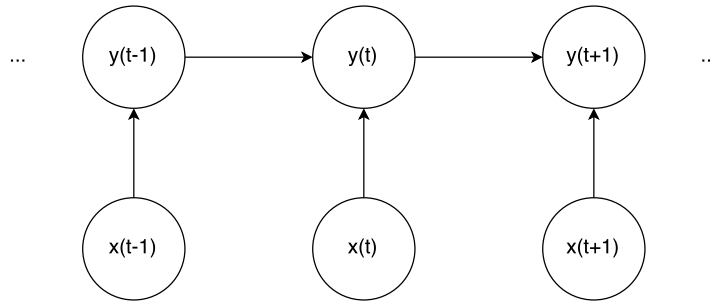


Figure 1: An Illustration of the graphical structure of a Hidden Markov Model (HMM). The arrows indicate the dependencies running from dependent to depedee.

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

$$p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x}|\mathbf{y})p(\mathbf{y}). \quad (1)$$

Applying the chain rule and the dependency assumptions we acquire,

$$\begin{aligned} p(\mathbf{x}|\mathbf{y}) &= p(x_1|\mathbf{y})p(x_2|x_1, \mathbf{y})\dots p(x_T|\mathbf{x}_{1:T-1}\mathbf{y}) \\ &= p(x_1|y_1)p(x_2|y_2)\dots p(x_T|y_T) \end{aligned} \quad (2)$$

and,

$$\begin{aligned} p(\mathbf{y}) &= p(y_1)p(y_2|y_1)\dots p(y_T, \mathbf{y}_{1:T-1}) \\ &= p(y_1)p(y_2|y_1)\dots p(y_T, y_{T-1}), \end{aligned} \quad (3)$$

Thus, we may rewrite the factorisation of the HMM as,

$$p(\mathbf{x}, \mathbf{y}) = \prod_{t=1}^T p(y_t|y_{t-1})p(x_t|y_t) \quad (4)$$

The probabilities $p(y_t|y_{t-1})$ are known as “transition” probabilities, and $p(x_t|y_t)$ 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, which are of course independent of any previous state. The model parameters must be precomputed, for example estimated through application of the Baum-Welch algorithm on an unsupervised training set. Now, given a sequence of observations, \mathbf{x} , we may predict the hidden state sequence, \mathbf{y}^* , by maximising the conditional distribution, $p(\mathbf{y}|\mathbf{x})$. Thus,

$$\mathbf{y}^* = \underset{\mathbf{y}}{\operatorname{argmax}} \left\{ \prod_{t=1}^T p(y_t|y_{t-1})p(x_t|y_t) \right\}. \quad (5)$$

Thus, 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 in the first-order Markov case by 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 which maximise the likelihood of the sequence fragment. To show this we define $v_t(s) = \max_{\mathbf{y}_{1:t-1}} p(\mathbf{y}_{1:t-1}, y_t = s|\mathbf{x})$, 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 $v_t(s) = \max_{\mathbf{y}_{1:t-1}} p(\mathbf{y}_{1:t-1}|\mathbf{x})p(y_{t-1}, y_t = s)p(x_t|y_t = s)$. Finally, $v_t(s) = \max_{\mathbf{y}_{1:t-1}} v_{t-1}(y_{t-1})p(y_{t-1}, y_t = s)p(x_t|y_t = s)$, and we may see the recursion. When the all states are 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. The algorithm must test all $|S|$ transitions from the previous state to each of the $|S|$ current states, and it does that at 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 algorithm to sequence learning is the forward-backward algorithm, so called for its computation of variables in both directions along the sequence. It is yet 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 (not the whole sequence), given the observation sequence and model parameters, namely, $p(y_t = s|\mathbf{x}, \theta)$. These conditional probabilities have many useful applications, for example in the Baum-Welch algorithm for estimating model parameters, but also for CRFs, as will be shown in section ???. We may write the forward-backward variable as,

Data: Observation sequence, \mathbf{x} , and model parameters, $\theta = (\mathbf{A}, \mathbf{B}, \mathbf{I})$

Result: Most likely sequence, \mathbf{y}^*

Initialise \mathbf{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 \mathbf{y}^*

end

end

Return \mathbf{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.

$$\gamma_t(s) = p(y_t = s | \mathbf{x}, \theta) = \frac{\alpha_t(s)\beta_t(s)}{\sum_{s' \in S} \alpha_t(s')\beta_t(s')}, \quad (6)$$

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

$$\begin{aligned} \alpha_t(s) &= \sum_{y_{t-1}} p(y_{t-1}, y_t = s, \mathbf{x}_{1:t}) \\ &= \sum_{y_{t-1}} p(y_t = s | y_{t-1}) p(x_t | y_t) p(y_{t-1}, \mathbf{x}_{1:t-1}) \\ &= \sum_{y_{t-1}} \mathbf{A}(y_{t-1}, s) \mathbf{B}(x_t, y_t) \alpha_{t-1}(y_{t-1}). \end{aligned} \quad (7)$$

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 the those from the previous time. Likewise, for the backward variables, we may write,

$$\begin{aligned} \beta_t(s) &= \sum_{y_{t+1}} p(y_t = s, y_{t+1}, \mathbf{x}_{t+1:n}) \\ &= \sum_{y_{t+1}} p(\mathbf{x}_{t+2:n} | y_{t+1}, x_{t+1}) p(x_{t+1}, y_{t+1} | y_t = s) \\ &= \sum_{y_{t+1}} \beta_{t+1}(y_{t+1}) \mathbf{A}(s, y_{t+1}) \mathbf{B}(x_{t+1}, y_{t+1}). \end{aligned} \quad (8)$$

From equations 7 and 8 comes Algorithm 2. 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 are

2.5 L-BFGS

The L-BFGS algorithm is

Data: Observation sequence, \mathbf{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 - $\mathcal{O}(T|S|^2)$

2.6 Maximum Entropy Markov Models

Maximum Entropy Markov models (MEMM) combine aspects of HMMs and maximum entropy classifiers.

2.7 Conditional Random Fields

Conditional Random Fields (CRF) are an elaboration of MEMMs.

1. HMMs
2. logistic regressions, (and optimisation thereof)
3. Maxent and how they differ to CRFs [?]
4. Inference algorithms
5. l-bfgs
6. mention sequence learning, graphical models, random fields, Hammersley-Clifford
7. define discriminative etc.
8. feature engineering
9. segue into Wapiti

2.8 Log-linear Models

2.9 Graphical Models

2.9.1 Hidden Markov Models

2.10 Conditional Random Fields

2.10.1 Feature Engineering

2.10.2 Wapiti

Lack of support for numeric features imposes constraints on our feature engineering. Any numeric-based idea must be discretised¹.

Both approaches yield the same input data for the CRF engine, and so evaluation is in fact equivalent to prediction, despite the initial difference in input formats. Figure ?? shows an excerpt from an input file to the CRF engine for training. These features are for inputs “January 1994” and “July 1996”, for training the Date model. The features range from token identity, to a variety of prefixes and punctuation features. It should be noted that OCR information is only used in higher level models, that is, the Header and Segmentation models. The input for lower-level models such as Date is plaintext, and so features are typically simple, but dictionary-based features, where information about a token is referenced in a dictionary resource within Grobid, are also used. Note the features shown are only those pertaining to the token itself. The full range of features (including those involving concatenations of the token’s neighbours etc.) are defined by a set of feature templates. The feature templates for each model are contained in a separate file. An excerpt of this is shown in Figure ?. These are given as a separate input to the CRF engine, and it is with these that the engine constructs all feature functions for the model. It is therefore vital that the feature extraction, which is generated by Grobid, is aligned with the template file, which is manually configured by the developer. As depicted in Figure ?, there is a strong coupling between these two parts of Grobid. The excerpt shown is from the Wapiti model, but the notation is the same for CRF++, which first standardised the syntax. This subset of five feature templates capture information about the capitalisation of a token and its neighbours. The notation has the structure, [identifier]:[%x][row, col], where row is the offset from the current token, and col indicates the feature index. Thus, “U50:%[0,11]”, denotes that the feature template identified as “U50” takes the 11th feature for the current token (0 offset). This feature will be equal to 1 if a token is capitalised, and 0 otherwise. “U52:%[-1,11]” indicates the same thing, but based on the capitalisation of the *previous* token. “U54:%x[-1,11]/%x[0,11]” is a binary function for detecting the capitalisation of the current *and* the following token.

Now we may see an alignment with the mathematical model. Recall a linear chain CRF is expressed in the simplest case as,

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{y})}{\sum_{\mathbf{y}'} p(\mathbf{x}, \mathbf{y}')}, \quad (9)$$

where,

$$p(\mathbf{x}, \mathbf{y}) = \exp \left\{ \sum_t \sum_{i,j \in S} \lambda_{ij} \mathbb{1}_{\{y_t=i\}} \mathbb{1}_{\{y_{t-1}=j\}} + \sum_t \sum_{i \in S} \sum_{o \in O} \mu_{io} \mathbb{1}_{\{y_t=i\}} \mathbb{1}_{\{x_t=o\}} \right\}, \quad (10)$$

Here \mathbf{x} is a sequence of observations and \mathbf{y} is a sequence of labels. S is the set of all labels, O is set of observations (the vocabulary of the tokens to be labelled). When the coefficients $\lambda_{ij} = \log p(y_t = i, y_{t-1} = j)$ and $\mu_{ij} = \log p(y_t = i, x_t = o)$, this joint distribution is equivalent to a Hidden Markov Model (HMM), with coefficients, λ_{ij} as transition probabilities and μ_{ij} emission probabilities. In this simple case, features are based solely on the token’s identity, i.e. feature functions are an indicator function. For clarity, we may write,

¹this is a footnote, and it’s crazy easy to make in latex.


```

* 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: Output from training date model

$$p(\mathbf{x}, \mathbf{y}) = \exp \left\{ \sum_{i \in S} \sum_{j \in S} \lambda_{ij} F_{ij}(\mathbf{y}) + \sum_{i \in S} \sum_{o \in O} \mu_{io} F_{io}(\mathbf{x}, \mathbf{y}) \right\}, \quad (11)$$

where $F_{ij} = \sum_t \mathbb{1}_{\{y_t=i\}} \mathbb{1}_{\{y_{t-1}=j\}}$ and $F_{io} = \sum_t \mathbb{1}_{\{y_t=i\}} \mathbb{1}_{\{x_t=o\}}$. In a CRF, however, we may replace the indicator function for observations with any sort of function, typically binary, extracting rich features from a token. Thus, $F_{io} = \sum_t \mathbb{1}_{\{y_t=i\}} f_{io}(\mathbf{x})$. The set of functions, $\{f_{io}\}$, are the functions that we define in the feature template files. Note that, unlike an HMM, the vocabulary is not pre-defined, it is “discovered” through training on samples. Therefore, the number of actual features depends on the training set itself, whereas the feature template is fixed. Since we use indicator functions, which produce a feature for every observation, we may end up with an enormous number of features. Take the Date model for example: 5815 features are produced for a single block (not counting the one representing the label), and there are seven labels. As per our formulation in (11) we therefore have $7 * 7$ “transition” features and $5815 * 7$ “emission” features, totalling 40754 features. This is corroborated by the model output in Figure 2. Wapiti automatically constructs this vast feature space from the inputs we provide. In the Date model, the labels are I-<day>, I-<month>, I-<year>, I-<other>, <day>, <month>, and <other>. The I (probably) stands for “initial”, as in training these are assigned to the first tokens of this class found in the string.

A model is typically a large file (as much as 100Mb). At the top of the file, the feature templates are declared, just as they are in the input. Because of this, that file is not required at prediction time. Following this the labels are declared. Then come two longer sections: first, the feature functions themselves as defined. Figure ?? shows the first 12 features produced from the first token in the first sample in the training set—“November”. Because this is the first token in the string, we see the first three feature macros, which relate to the identity of the token’s predecessors, remain unresolved. The fourth, however, shows the indicator for the token. This function will be true if a token is equal to “November”. The fifth function is an indicator for if the token’s successor is equal to “19”, and so on. The final (and usually largest) section of the model file defines the non-zero weights for the feature functions. The weights are represented in scientific notation and in hexadecimal representation, presumably to avoid arithmetic underflow (a common problem when dealing with the computation of HMMs and related models).

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 1: Evaluation results for reference segmentation

label	accuracy	precision	recall	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 2: Evaluation results for citations

3 Automatic Metadata Extraction

3.1 Metadata Extraction

3.2 Related Work

3.3 GROBID

[Show here Grobid vs. refextract]

4 Implementation and Data

4.1 Extensions

4.2 Data Acquisition

5 Results and Analysis

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

5.1 Experiment Setup

Months of CPU time? (parallelised), 64 experiments (before an combination experiments are run) Mind you, though we aren't explicitly interested in identifying headnotes, footnotes, page numbers etc., correctly classifying them does spare the important categories (header, references) from garbage data.

5.2 Evaluation Method

5.3 Baseline

5.3.1 Header model - Cora dataset

5.3.2 Header model - Cora dataset appending HEP dataset

5.3.3 Header model - Cora and HEP combined datasets

5.3.4 Header model - HEP dataset

5.3.5 Header model - HEP dataset appending CORA dataset

5.3.6 Header model - HEP dataset appending 1/3 CORA dataset

5.3.7 Header model - HEP dataset appending 2/3 CORA dataset

5.3.8 Segmentation model - Cora dataset

5.3.9 Segmentation model - Cora dataset appending HEP dataset

5.3.10 Segmentation model - Cora and HEP combined datasets

5.3.11 Segmentation model - HEP dataset

5.3.12 Segmentation model - HEP dataset appending CORA dataset

5.4 Regularisation

5.4.1 Header model - $L2 = 0$

5.4.2 Header model - $L2 = 1e^{-6}$

5.4.3 Header model - $L2 = 1e^{-5}$

5.4.4 Header model - $L2 = 1e^{-4}$

5.4.5 Header model - $L2 = 1e^{-3}$

5.5 Dictionaries

5.5.1 Header model - HEP dataset

5.5.2 Header model - HEP dataset appending CORA dataset

5.5.3 Segmentation model - HEP dataset

5.5.4 Segmentation model - HEP dataset appending CORA dataset

5.5.5 Header Model - HEP dataset - 2^{nd} Degree Features

5.5.6 Header Model - HEP dataset Appending CORA - 2^{nd} Degree Features

5.5.7 Header Model - HEP dataset - 3^{rd} Degree Features

5.5.8 Header Model - HEP dataset Appending CORA - 3^{rd} Degree Features

5.6 Dictionaries + stop words

7 References

8 Appendices