Automatic Metadata Extraction: The High Energy Physics Use Case

Joseph Boyd

July 7, 2015

Contents

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, $\mathbf{y} = (y_1, y_2, ..., y_T)$, 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 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.

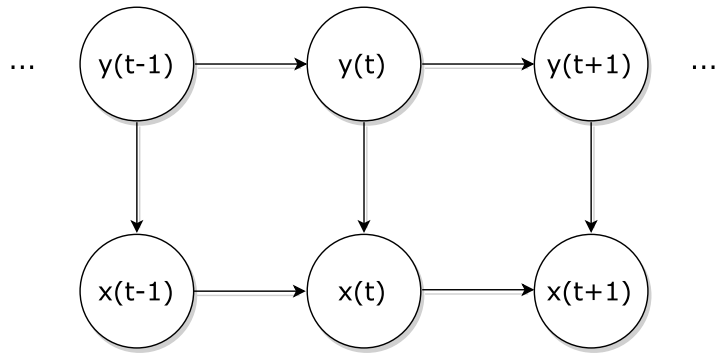

Figure 1: An Illustration of the graphical structure of a Hidden Markov Model (HMM). The arrows indicate the dependencies running from dependent to dependee.

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

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

Applying the chain rule and the dependency assumptions we acquire,

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

and,

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

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

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

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, $\mathrm{p}(\mathbf{y}|\mathbf{x})$. Thus,

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

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, $v_t(s) = \max_{\mathbf{y}_{1:t-1}} \mathrm{p}(\mathbf{y}_{1:t-1}, y_t = s|\mathbf{x})$, 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,

$$
\begin{aligned}
v_t(s) &= \max_{\mathbf{y}_{1:t-1}} \mathrm{p}(\mathbf{y}_{1:t-1}|\mathbf{x})\mathrm{p}(y_{t-1}, y_t = s)\mathrm{p}(x_t|y_t = s) \\
&= \max_{\mathbf{y}_{1:t-1}} v_{t-1}(y_{t-1})\mathrm{p}(y_{t-1}, y_t = s)\mathrm{p}(x_t|y_t = s),
\end{aligned} \tag{6}
$$

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 **??**. 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, $\mathrm{p}(y_t = s|\mathbf{x}, \theta)$. These conditional probabilities have many useful applications, for example in the Baum-Welch algorithm for estimating model parameters, but also for CRFs, as will be shown in section 2.7. We may write the forward-backward variable as,

$$
\gamma_t(s) = \mathrm{p}(y_t = s|\mathbf{x}, \theta) = \frac{\alpha_t(s)\beta_t(s)}{\sum_{s' \in S} \alpha_t(s')\beta_t(s')}, \tag{7}
$$

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

$$\begin{aligned}\alpha_t(s) &= \sum_{y_{t-1}} \mathrm{p}(y_{t-1}, y_t = s, \mathbf{x}_{1:t}) \\ &= \sum_{y_{t-1}} \mathrm{p}(y_t = s|y_{t-1})\mathrm{p}(x_t|y_t)\mathrm{p}(y_{t-1}, \mathbf{x}_{1:t-1}) \\ &= \sum_{y_{t-1}} \mathbf{A}(y_{t-1}, s)\mathbf{B}(x_t, y_t)\alpha_{t-1}(y_{t-1})).\end{aligned} \tag{8}$$

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

$$\begin{aligned}\beta_t(s) &= \sum_{y_{t+1}} \mathrm{p}(y_t = s, y_{t+1}, \mathbf{x}_{t+1:n}) \\ &= \sum_{y_{t+1}} \mathrm{p}(\mathbf{x}_{t+2:n}|y_{t+1}, x_{t+1})\mathrm{p}(x_{t+1}, y_{t+1}|y_t = s) \\ &= \sum_{y_{t+1}} \beta_{t+1}(y_{t+1})\mathbf{A}(s, y_{t+1})\mathbf{B}(x_{t+1}, y_{t+1})).\end{aligned} \tag{9}$$

From equations 8 and 9 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, 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, 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 β. 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 applying these

optimal model parameters linearly. The result is then passed through a non-linear *logistic* function, denoted σ to obtain a probability. Formally,

$$y* = \sigma(\beta^T \mathbf{x}). \tag{10}$$

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, $\mathrm{p}(y_n = C_1|\mathbf{x}_n; \beta) = \sigma(\beta^T \mathbf{x})$, and $\mathrm{p}(y_n = C_1|\mathbf{x}_n; \beta) = 1 - \sigma(\beta^T \mathbf{x})$, 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,

$$\begin{aligned} \log \mathrm{p}(\mathbf{y}|\mathbf{X}, \boldsymbol{\beta}) = \log \prod_{n=1}^{N} \mathrm{p}(y_n, \mathbf{x}_n) &= \log \left(\prod_{n:y_n=C_1}^{N} \sigma(\beta^T \mathbf{x}) \prod_{n:y_n=C_2}^{N} 1 - \sigma(\beta^T \mathbf{x}) \right) \\ &= \log \prod_{n=1}^{N} \sigma(\beta^T \mathbf{x})^{y_i} (1 - \sigma(\beta^T \mathbf{x}))^{1-y_i} \end{aligned} \tag{11}$$

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

$$\log \prod_{n=1}^{N} \prod_{c=1}^{C} \mu_{i,c}^{y_{ic}}, \tag{12}$$

where $y_{ic} = \mathbb{1}_{y_i=c}$ and y_i is a bit vector indicating the class of the ith sample. In this general, multinomial case, the probabilities are written, $\mu_{i,c} = \frac{\exp(\beta_c^T \mathbf{x}_i)}{\sum_{c'=1}^{C} \exp(\beta_c^T \mathbf{x}_i)}$, which are normalised to ensure they sum to 1, and β_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,

$$\mathcal{L}(\mathbf{B}) = \log \mathrm{p}(\mathbf{y}|\mathbf{X}, \boldsymbol{\beta}) = \sum_{n=1}^{N} \left(\sum_{c=1}^{C} y_{ic} \beta_c^T \mathbf{x}_n \right) - \log \left(\sum_{c'=1}^{C} \exp(\beta_c^T \mathbf{x}_n) \right). \tag{13}$$

We then require an optimisation algorithm to solve for $\mathbf{B}$.

2.5 L-BFGS

The classic first-order gradient descent algorithm defines the iteration step to be,

$$\boldsymbol{\beta}^{k+1} = \boldsymbol{\beta}^k - \alpha \nabla \mathcal{L}(\boldsymbol{\beta}^k), \tag{14}$$

where ... 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,

$$\boldsymbol{\beta}^{k+1} = \boldsymbol{\beta}^k - \alpha_k \mathbf{H_k}^{-1} \mathbf{g}_k, \tag{15}$$

where $\mathbf{H}$ is the $(D \times D)$ Hessian matrix of partial derivatives. Whereas for smaller problems, these algorithms are adequate, for models of large dimension, such as those encountered in metadata extraction, which may run up into the millions, smarter approaches are required. The *limited memory Broyden–Fletcher–Goldfarb–Shanno* (L-BFGS) algorithm makes savings on both the computation and storage of the Hessian matrix central to the second-order Newton's method., and has come to be the standard learning algorithm for ?? . The original BFGS already being a quasi-Newton method, saving on the *computation* of the . The L-BFGS algorithm is the tool of choice for many problems[3] and is the algorithm we use in our analysis.

2.6 l_2 Regularisation

2.7 Conditional Random Fields

Conditional Random Fields (CRFs) are . They are an improvement to the similar, Maximum Entropy Markov models (MEMM)[2], which combine aspects of maximum entropy classifiers and Hidden Markov models.[1]

Classification over relational data can benefit greatly from rich features, that is, describing observed attributes about an observation beyond simply its identity as we modelled for HMMs. However, the resulting model complexity of defining rich,

context-aware features will be typically too great to be workable. In conditional random fields (CRF), we circumvent this problem by instead modelling the conditional distribution, $p(\mathbf{y}, \mathbf{x})$ of the graph structure giving us a free choice over features, which we may choose and implicitly define a distribution over $\mathbf{x}$ without having to model this distribution at all.[4] Furthermore, modelling the conditional distribution is sufficient for classification wherein the observation sequence is given. This freedom for rich feature engineering is what make CRFs the current state of the art in metadata extraction, where arbitrarily defined features may make for good indicators.

With CRFs, as with HMMs, our objective is to make a sequence prediction for a sequence of structured data.

1. mention sequence learning, graphical models, random fields, Hammersley-Clifford
2. define discriminative etc.

2.8 Beyond Conditional Random Fields

2.9 Feature Engineering

2.10 Wapiti

Lack of support for numeric features imposes constraints our feature engineering. Any numeric-based idea must be discretised[1].

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

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

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,

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{y})}{\sum_{y'} p(\mathbf{x}, \mathbf{y}')}, \tag{16}$$

where,

$$p(\mathbf{x}, \mathbf{y}) = \exp\left\{ \sum_t \sum_{i,j \in S} \lambda_{ij} \mathbb{1}_{\{y_t = i\}} \mathbb{1}_{\{y_{t-1} = j\}} + \sum_t \sum_{i \in S} \sum_{o \in O} \mu_{io} \mathbb{1}_{\{y_t = i\}} \mathbb{1}_{\{x_t = o\}} \right\}, \tag{17}$$

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,

$$p(\mathbf{x}, \mathbf{y}) = \exp\left\{ \sum_{i \in S} \sum_{j \in S} \lambda_{ij} F_{ij}(\mathbf{y}) + \sum_{i \in S} \sum_{o \in O} \mu_{io} F_{io}(\mathbf{x}, \mathbf{y}) \right\}, \tag{18}$$

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 (1) 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

```
* 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

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 with the computation of HMMs and related models).

3 Automatic Metadata Extraction

3.1 Metadata Extraction

3.2 Related Work

3.3 GROBID

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

[Show here Grobid vs. refextract]

4 Implementation and Data

4.1 Extensions

4.2 Data Acquisition

5 Results and Analysis

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.

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

5.2 Evaluation Method

5.3 Baseline

5.4 Regularisation

5.5 Dictionaries

5.6 Dictionaries + stop words

5.7 Token Selection

5.8 Levenshtein

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

$$\text{similarity}_{a,b} = 1 - \frac{\text{lev}_{a,b}(|a|,|b|)}{\max(|a|,|b|)}$$

5.9 Line Shape

5.10 Character Classes

6 Conclusion

6.1 Summary

6.1.1 Key Results

6.2 Future Work

* expand training sets * model collaborations in the citation model

References

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

[2] 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.

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

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

7 Appendices

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

Result: Most likely sequence, **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.

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 - $\mathcal{O}(T|S|^2)$