Automatic Metadata Extraction: The High Energy Physics Use Case

Joseph Boyd

July 5, 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. Notably, we pinpoint the piece of the mathematical model relevant to our work on feature engineering. Finally, we describe Wapiti, a general-purpose "engine" for training and applying CRF models.

2.1 Hidden Markov Models

Hidden Markov models (HMMs) are a staple of natural language processing (NLP). An HMM models a probability distribution over an unknown, "hidden" sequence of length T, $\mathbf{y} = (y_1, y_2, ..., y_T)$, whose elements take on values in a finite set of states, S, and its corresponding known sequence of "observations", $\mathbf{x} = (x_1, x_2, ..., x_T)$, similarly taking values in a finite set, O. 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 80). 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 (these are modelled with "emission probabilities"). The first dependency is referred to as the Markov condition which postulates the dependence of each hidden state on its k precursors in the hidden sequence. 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 the dependency of an observation only on its hidden state.

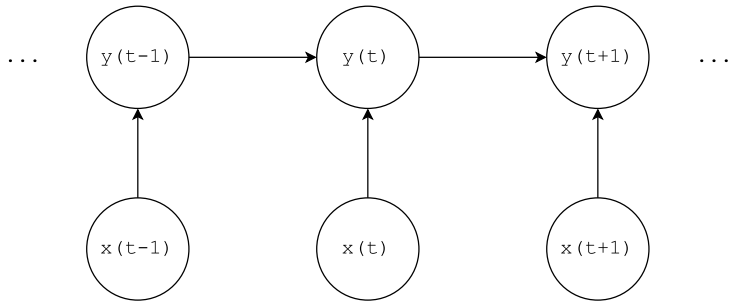

Figure 1: Illustration of the graphical structure of a Hidden Markov Model (HMM). The arrows indicate the dependencies running from dependent to dependee.

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 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,

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

The probabilities $p(y_t|y_{t-1})$ are "transition" probabilities, and $p(x_t|y_t)$ "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 $\mathbf{I}$ if 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,

$$\mathbf{y}* = \underset{\mathbf{y}}{\operatorname{argmax}} \left\{ \prod_{t=1}^{T} p(y_t|y_{t-1})p(x_t|y_t) \right\}. \tag{4}$$

HMMs have shown to be successful in statistical modelling problems . In Part of Speech (PoS) tagging, the hidden parts of speech (nouns, verbs, and so on) of a word sequence (sentence). In this case, .

are predicted by maximising the likelihood over all possible hidden sequences. This seemingly intractable problem may be solved in quadratic time in the first-order Markov case by using dynamic programming.

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 best sequence from the first $t-1$ states, with the choice of s at time t. Thus, we may write $v_t(s) = \max_{\mathbf{y}_{1:t-1}} \mathrm{p}(\mathbf{y}_{1:t-1}|\mathbf{x})\mathrm{p}(y_{t-1}, y_t = s)\mathrm{p}(x_t|y_t = s)$. Finally, $v_t(s) = \max_{\mathbf{y}_{1:t-1}} v_{t-1}(y_{t-1})\mathrm{p}(y_{t-1}, y_t = s)\mathrm{p}(x_t|y_t = s)$, and we may see the recursion. Pseudocode is given in Algorithm 1. Clearly 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 modest $\mathcal{O}(T|S|^2)$.

Data: Observation sequence, $\mathbf{x}$, and model parameters, $\theta = (\mathbf{A}, \mathbf{B}, \mathbf{I})$
Result: Most likely sequence, **y***
for $s \in S$ **do**
  $v_1(s) = \mathbf{I}_s \times \mathbf{B}_s(x_1)$
end
for $t = 2$ *to* T **do**
  for $s \in S$ **do**
    $v_t(s) = \mathbf{B}_s(x_t) \times \max_{s'}(\mathbf{A}_{s'}(s) \times v_{t-1}(s'))$
  end
end
The maximising sequence may then be read backward

Algorithm 1: The Viterbi algorithm - $\mathcal{O}(T|S|^2)$

2.3 Forward Backward Algorithm

Another important algorithm to sequence learning is the forward backward algorithm, so called for its. 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 (not the whole sequence) at each time step, given the observation sequence and model parameters, namely, $\mathrm{p}(y_t = s|\mathbf{x}, \theta)$.

Data: Observation sequence, $\mathbf{x}$, and model parameters, $\theta = (\mathbf{A}, \mathbf{B}, \mathbf{I})$
Result: Most likely sequence, **y***
for $s \in S$ **do**
| $v_1(s) = \mathbf{I}_s \times \mathbf{B}_s(x_1)$
end
for $t = 2$ *to* T **do**
| **for** $s \in S$ **do**
| | $v_t(s) = \mathbf{B}_s(x_t) \times \max_{s'}(\mathbf{A}_{s'}(s) \times v_{t-1}(s'))$
| **end**
end
The maximising sequence may then be read backward

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

2.4 L-BFGS

2.5 Maximum Entropy Classifiers

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 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 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,

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{y})}{\sum_{y'} p(\mathbf{x}, \mathbf{y}')}, \tag{5}$$

where,

$$p(\mathbf{x}, \mathbf{y}) = \exp\left\{ \sum_t \sum_{i,j \in S} \lambda_{ij} \mathbb{1}_{\{y_t = i\}} \mathbb{1}_{\{y_{t-1} = j\}} + \sum_t \sum_{i \in S} \sum_{o \in O} \mu_{io} \mathbb{1}_{\{y_t = i\}} \mathbb{1}_{\{x_t = o\}} \right\}, \tag{6}$$

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,

[1] this 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

$$p(\mathbf{x}, \mathbf{y}) = \exp\left\{ \sum_{i \in S} \sum_{j \in S} \lambda_{ij} F_{ij}(\mathbf{y}) + \sum_{i \in S} \sum_{o \in O} \mu_{io} F_{io}(\mathbf{x}, \mathbf{y}) \right\}, \tag{7}$$

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 (7) 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 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

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

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

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