École Polytechnique Fédérale de Lausanne

Master Thesis

Automatic Metadata Extraction - The High Energy Physics Use Case

Author:
Joseph Boyd

Supervisors:
Dr. Martin Rajman
Dr. Gilles Louppe

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

July 2015

Declaration of Authorship

I, Joseph Boyd, declare that this thesis titled, 'Automatic Metadata Extraction - 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 master's 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 Extraction -
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

I would firstly like to thank my two supervisors, Dr. Gilles Louppe at CERN and Dr. Martin Rajman at EPFL, for their advice and guidance. I would also like to thank my colleagues and friends at CERN for their inspiration and support.

Contents

Declaration of Authorship i

Abstract ii

Acknowledgements iii

Contents iv

List of Figures vii

List of Tables ix

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Aims . . . 2
- 1.3 Main Results . . . 2
- 1.4 Outline . . . 3

2 Supervised Sequence Learning 4
- 2.1 Hidden Markov Models . . . 4
- 2.2 Viterbi Algorithm . . . 6
- 2.3 Forward-backward Algorithm . . . 7
- 2.4 Maximum Entropy Classifiers . . . 8
- 2.5 L-BFGS . . . 9
- 2.6 Regularisation . . . 10
- 2.7 Conditional Random Fields . . . 10
- 2.8 Feature Functions . . . 13
- 2.9 Wapiti . . . 13
  - 2.9.1 Feature Templates . . . 14
  - 2.9.2 Extracted Features . . . 14
  - 2.9.3 Models . . . 15
  - 2.9.4 Training . . . 15

3 Automatic Metadata Extraction 17
- 3.1 Metadata Extraction . . . 17
- 3.2 Solution Methods . . . 18
- 3.3 GROBID . . . 19
  - 3.3.1 Text Encoding Initiative . . . 20
  - 3.3.2 Training . . . 22

3.3.3 Evaluation . . . 22
3.3.4 Prediction . . . 23
3.3.5 Other Functionality . . . 23

4 Data, Methods, and Implementation 25
4.1 Assessments . . . 25
4.2 Data Acquisition . . . 26
4.2.1 CORA dataset . . . 28
4.2.2 HEP dataset . . . 28
4.3 Methods . . . 29
4.3.1 Baseline . . . 29
4.3.2 Block Size . . . 29
4.3.3 Character Classes . . . 30
4.3.4 Dictionaries . . . 32
4.3.5 Levenshtein Distance . . . 33
4.3.6 Regularisation . . . 34
4.3.7 Token Extensions . . . 34
4.4 Implementation . . . 34
4.4.1 Extensions to GROBID . . . 34
4.4.2 Experiment Pipeline . . . 35

5 Results and Analysis 37
5.1 Evaluation Method . . . 37
5.1.1 Evaluation Metrics . . . 37
5.1.2 Evaluation in GROBID . . . 39
5.2 Experiment Setup . . . 39
5.3 Comparison with *refextract* . . . 41
5.4 Results . . . 43
5.4.1 Baseline . . . 43
5.4.2 Block Size . . . 45
5.4.3 Character Classes . . . 46
5.4.4 Dictionaries . . . 47
5.4.5 Levenshtein Distance . . . 48
5.4.6 Regularisation . . . 49
5.4.7 Token Extensions . . . 50
5.5 Key Results . . . 51
5.5.1 Header Model . . . 51
5.5.2 Segmentation Model . . . 52

6 Conclusions 55
6.1 Summary . . . 55
6.2 Future Work . . . 56

A Algorithms 58

B Figures 59

C Statistical Tests 62
C.1 Stop Word Frequency . . . 62

Bibliography **64**

List of Figures

2.1 An illustration of the graphical structure of a Hidden Markov Model (HMM). The arrows indicate the dependencies running from dependee to dependent. . . . 5
2.2 Excerpt of capitalisation features templates or *macros*. . . . 14
2.3 Features for a single date instance of three tokens: '4 August 1989'. . . . 15
2.4 Expanded feature functions deriving from capitalisation macros. . . . 15
2.5 Output from training date model . . . 16

3.1 An illustration of the way a header section might be segmented and classified. The classes modelled are colour-coded (title in yellow, authors in green, and so on). Article excerpt from (McCallum et al. [2000a]). . . . 19
3.2 Sample tagged citation for GROBID training input. . . . 22
3.3 An illustration of the interactions between GROBID and Wapiti for the two main functions of training and tagging. The dashed arrows indicate training operations; the solid arrows, tagging. . . . 24
3.4 The models of GROBID are organised into a cascade, where each part of a document is classified in increasingly greater detail. . . . 24

4.1 Figure (A) shows a collaboration field in a header section. Figure (B) shows discontinuous front matter that sits on the first page, but apart from the main header section and within the introductory section. Figures (C) and (D) give the authors list and affiliations for a large HEP collaboration; the author list begins on page 8 and continues to page 33. Figure (B) from (Maguire et al. [2012]), other excerpts from (Aaij et al. [2015]). . . . 27
4.2 Character class breakdown of sample lines from different sections of a CERN LHCb collaboration paper. The paper in question is the current world record holder for number of authors, and lists over 5000 authors and their affiliations. The radar plots give a different impression for each of the samples. . . . 31
4.3 Misclassification of <header> to <body> proportion and count (given in parentheses) for five papers in an evaluation fold, an output of the confusion matrix utility. . . . 35
4.4 Example of a successfully classified collaboration. The choice of XML tags is ours and was selected to be consistent with the TEI standard. . . . 35
4.5 An illustration of the experimentation pipeline. . . . 36

5.1 The different cross-validation configurations used in our experiments. Figures (A) and (B) show cross-validation on HEP and CORA sets independently. Figures (C) and (D) show cross-validation on the HEP and CORA datasets respectively, appending the other at training time. . . . 40

5.2 Confusion matrix for *segmentation* model with baseline features, trained on the pure HEP dataset. Counts are given, as well as a heatmap to indicate the most frequent classifications. 45
5.3 Distribution of model parameters with l_2 regularisation. 49
5.4 Comparison of baseline, dictionary, and stop word features for overall *header* model performance. 51
5.5 Comparison of baseline, Levenshtein distance and character class features for overall *segmentation* model performance. 52
5.6 Comparison of baseline, Levenshtein distance and character class features for the header extraction. 53
5.7 Comparison of baseline, Levenshtein distance and character class features for the reference extraction. 53
5.8 Confusion matrix for segmentation model with character class features, trained on the pure HEP dataset. Counts are given, as well as a heatmap to indicate the most frequent classifications. 54

B.1 The header section of a scientific paper. Excerpt from Peng and McCallum [2004] 59
B.2 The header section of a HEP paper. Excerpt from Zubair [2015] 59
B.3 Comparison of different character class feature discretisation strategies. . . 60
B.4 Comparison of different levenshtein distance feature thresholding strategies. 60
B.5 Comparison of different data configurations subsampling the CORA dataset. 61
B.6 Confusion matrix for *header* model, baseline features, trained on HEP data.. 61

C.1 Box plots of stop word frequency according to header section. 62
C.2 ANOVA showed the average stop word frequency of header sections varies significantly. 63
C.3 Pairwise t-tests showed significance for each comparison. 63

List of Tables

3.1 A summary of the models coordinated by GROBID. We have here excluded the Patent, Entities, and E-book models as these are experimental models not currently used by GROBID. 21

4.1 Number of training instances for each model from each dataset. 28
4.2 Character classes used as features, along with the regular expressions used to count them. 32

5.1 A summary of our experiments, organised by category, models trained for, and data configurations used. 41
5.2 Token-level evaluation results for reference segmentation. 42
5.3 Token-level evaluation results of citation extraction for GROBID and *refextract*. 43
5.4 Mean and standard deviation for baseline data configurations. 44
5.5 Mean and standard deviation for subsampling CORA dataset. 44
5.6 Mean and standard deviation for block size variations. 46
5.7 Mean and standard deviation for character class discretisation strategies. . 47
5.8 Mean and standard deviation for dictionary features. 48
5.9 Mean and standard deviation for dictionary features combined with stop word features. 48
5.10 Mean and standard deviation for Levenshtein distance thresholding variations. 49
5.11 Mean and standard deviation for tuning the variance of the l_2 variance parameter. 50
5.12 Mean and standard deviation for token extension variations. 50

Chapter 1

Introduction

1.1 Motivation

CERN (Centre Européen de la Recherche Nucléaire) is the foremost European particle physics research laboratory, located on the border of France and Switzerland to the west of Geneva. The organisation, employing as many as 15,000 staff, fellows, and students at a time, is the global leader in high energy physics research, centered around massive particle collision experiments in the Franco-Swiss subterrain. INSPIRE-HEP (inspire-hep.net) is an online open access digital library for high energy physics (HEP) papers built by an international collaboration of particle physics laboratories. INSPIRE-HEP derives from the older SPIRES project (Gentil-Beccot et al. [2009]), and mounted upon the Invenio digital library software package. The digital library comprises of over one million HEP articles, theses and books, each painstakingly curated by professional librarians. As in most domains, such manual work may be enhanced with the assistance of automated processes, such as software systems based on state-of-the-art computer science techniques. Of particular interest is the annotation of scientific articles, that is, extracting a document's metadata from its *content* such that it may be represented as an identifiable entity of the digital library. We refer to this as the problem of *automatic metadata extraction* (AME). Though such technology is mainstream, at present INSPIRE-HEP has only limited solutions to this problem, relying heavily on tedious curation to complete the work. AME (Chapter 3) has been described as one of the hardest problems in document engineering (Souza et al. [2014]), and a range of solution methods exist. This thesis presents a state-of-the-art open-source software for metadata extraction, GROBID, a system based on machine learning approaches, and shows how HEP papers have particular characteristics that require specialised extensions and feature engineering to improve the accuracy of extraction. Of particular note is the way a HEP paper may contain collaborations of thousands of authors (refer to Chapter 4). This

alters the document layout drastically, creating structures unseen by baseline models. The requirements are therefore twofold:

1. to train models on a custom HEP training set, and;
2. to engineer specialised features conducive to better generalisation on HEP papers.

1.2 Aims

Above all, we hypothesise the HEP use case: a qualitative difference in both the content and layout of a HEP article compared with other scientific articles, moreover implying the value of specialised model training and feature engineering. These differences are in fact directly observable in any representative HEP corpus, as we see in Chapter 4. The availability of a baseline dataset, in addition to our own custom HEP dataset, allow us to further experiment with hybrid datasets. In short, our aims are:

1. to demonstrate the qualitative difference between HEP and general papers;
2. to propose improvements to model features, with suitable justifications;
3. to run experiments to confirm or reject our improvements, and;
4. to draw conclusions about what characterises good feature engineering.

1.3 Main Results

The results of our 66 cross-validated experiments are presented in Chapter 5. These results confirm our initial hypothesis about HEP papers, as well as many of our intuitions. Concretely, we found improvements for each of the two GROBID models that we addressed, namely the *header* model for processing article front matter, as well as the *segmentation* model, the most important model in the cascade. In each case we found two feature variations to make substantial improvements over the baseline evaluations, making error reductions as great as $20\% - 25\%$ for the most important classes.

1.4 Outline

In Chapter 2 we provide a discussion of the relevant machine learning techniques, along with their solution algorithms, up to and including the state-of-the-art conditional random fields (CRF). In Chapter 3 we present the problem of automatic metadata extraction, and the leading open-source software for the task, GROBID. Then, in Chapter 4,

we showcase our propositions for improvements and extensions to the baseline models. In Chapter 5 we present the results of our work and finally in Chapter 6 we conceive of ways our research may be continued and extended.

Chapter 2

Supervised Sequence Learning

This chapter presents the state-of-the-art technique for metadata extraction, conditional random fields (CRF). For completeness, it includes a background history of related machine learning techniques and their associated optimisation algorithms. It begins with a presentation of hidden Markov models (HMM) and their inference algorithms. Following this, multinomial logistic regression is presented. Combining ideas from these former topics produces Maximum Entropy Markov Models (MEMM) and CRFs. Notably, the discussion pinpoints the part of the mathematical model relevant to feature engineering. The chapter concludes with a description of Wapiti, a general-purpose software engine for training and tagging with CRF models.

2.1 Hidden Markov Models

Hidden Markov models (HMMs) (Rabiner [1989]) are a staple of natural language processing (NLP) and other engineering fields. A HMM models a probability distribution over an unknown, *hidden* sequence of states 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 Markov condition to be of first degree, that is, $k = 1$. Incidentally,

[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.

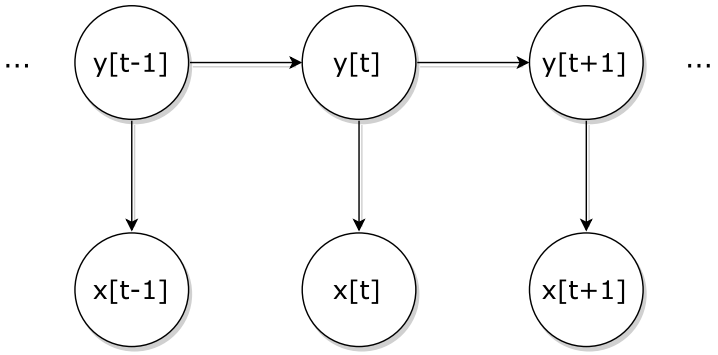


FIGURE 2.1: An illustration of the graphical structure of a Hidden Markov Model (HMM). The arrows indicate the dependencies running from dependee to dependent.

higher-order HMMs may always be reconstructed to this simplest form. For example, for $k = 2$ it suffices to define $S' = S \times S$ and $O' = O \times O$, that is, compound transition and emission probabilities. Thus, we have a first-order HMM with variables $Y'_t = (Y_{t-1}, Y_t)$. The second dependency assumption 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 (Charniak et al. [1993]). 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.

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 two 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,

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

where $\mathrm{p}(y_1|y_0) = \mathrm{p}(y_1)$. 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,

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

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 (Viterbi [1967]) 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,

[2]For example, the model parameters can be estimated through application of the Baum-Welch algorithm (Baum and Petrie [1966]) on an unsupervised training set.

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

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 (Binder et al. [1997]), 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), which combines the results to compute the forward-backward variables, $\gamma_t(s)$. 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 consists of 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 $\sigma = \frac{1}{1+e^{-x}}$, 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

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

encoded as 0 and 1 respectively. Notice the probabilities sum to 1. Now, the log-likelihood can be expressed as,

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

where $y_i \in \{0, 1\}$ and i.i.d. 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 log-likelihood 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[4]. 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,

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

[4]Or, equivalently, greatest *ascent* for maximising a *concave* function.

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,

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

where $\mathbf{H}$ 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. The complexity of each iteration is linear in the dimensionality of the model as well as the number of training samples.

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. 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*[5]. Large parameters are therefore discouraged and this helps prevent the creation of complex models during training that do not fit test data well. 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 gradient-based L-BFGS (Section 2.5), as an l_1 penalty is not differentiable.

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. [2000a]), which combine aspects of maximum entropy

[5] A prior distribution in the Bayesian sense is the initial distribution of a variable taken independently. In l_2, the penalty is $\frac{1}{2}\lambda\beta^T\beta = \log e^{\frac{\lambda}{2}\beta^T\beta} = \log \mathcal{N}(\beta|0, 1/\lambda) = \log \mathrm{p}(\beta)$.

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). 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 bibliographic 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 *occurrence* 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. [2000a]), varying the tuning parameter, σ^2 (the variance), even 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. Hence, the complexity of training is $\mathcal{O}(INT|S|^2)$, where I is the number of iterations required and N is the number of training samples. 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. [2000a]).

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 *vocabulary* 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 larger training sets, a model will have more 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:

```
# 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*.

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

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 ultimately 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' indicates 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

```
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.

token (corresponding to a simple token indicator feature function such as in Equation 2.16), to token prefixes[6], to information about capitalisation and punctuation, and so 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[7].

Of most interest are the expanded feature functions[8], 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 begins 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

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

[7] This presumably to avoid numeric underflow.

[8] 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

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 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

This chapter presents the problem of automatic metadata extraction, giving illustrations of the problem, its complexities, and a discussion of the methods by which the problem may be solved. Moreover, this chapter introduces GROBID, the metadata extraction tool around which our work is based, and describes the cascade of CRF models it uses to classify a full document.

3.1 Metadata Extraction

In our work we are concerned with automatic metadata extraction (AME) 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 may even have novel applications[1]. At CERN, the problem has been partially solved, albeit in a rudimentary way, and entails a lot of manual curation to complete the work. See Section 5.3 for a comparison between this existing solution and GROBID, the leading tool for metadata extraction.

Metadata refers to various information explicitly or implicitly contained in a scientific article. Perhaps the most important metadata for an article is that contained in the header, that is, the text at the front of a document, typically containing the title of the article as well as the names, affiliations and often the contact details of the authors, concluding finally with the article abstract. As a general rule, this is tantamount to the text of the document falling before the first section of the body (usually called 'Introduction'), though as we find in Chapter 4, sometimes significant amounts of front

[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).

matter is held in unexpected places. Other important types of metadata may be the references of the article, typically classified into fields such as publication title, authors, date of publication, and so on. Another potential metadata type is that of the document structure, its chapters and sections. All of these types are modelled by GROBID.

Extraction could refer to either of two distinct concepts. First, it may be the parsing of a PDF document and extraction of plaintext and images. This in itself is a complex problem, and may involve machine learning techniques for OCR analysis, depending on the rendering of the document. Or, it may be the *classification* of document content into predefined categories. It is on the second idea that we are focused within this work. Indeed, GROBID addresses both of these points, but the first is merely a precondition for the analysis with which it is primarily concerned, and it houses a third-party PDF-to-XML conversion tool, *pdftoxml* (Déjean and Meunier [2006]), developed at Xerox Research Centre Europe (XRCE), to handle this.

To appreciate the difficulty of automating such a task, contrast Figures B.1 and B.2 in Appendix B, contrasting the header sections of two articles from our dataset. Though the same sorts of information are present in both headers–title, author names, affiliations, and document abstract–the arrangement and presentation of these fields are different, for example the sizing and placement of the document title, the juxtapositioning of authors (which are variable in number) and author details, and the labelling of the abstract block. Furthermore, the second header is more complete, in that it contains information not present in the other, for example copyright and publication details. The contents of a document header do not follow a predictable ordering, making the problem hard, but are not entirely random, a condition that would render the problem impossible to solve. There is structure to a document, but it is likely infeasible to model deterministically. Therefore, we must look to probabilistic approaches, and accept that these will be error-prone. Also, if we are to process an entire document, it is unlikely we can create a one-size-fits-all model, rather, the problem must be decomposed.

3.2 Solution Methods

A 2013 study of metadata extraction techniques (Lipinski et al. [2013]) identified three fundamental methods for AME:

1. stylistic analysis;
2. knowledge base, and;
3. machine learning approaches.

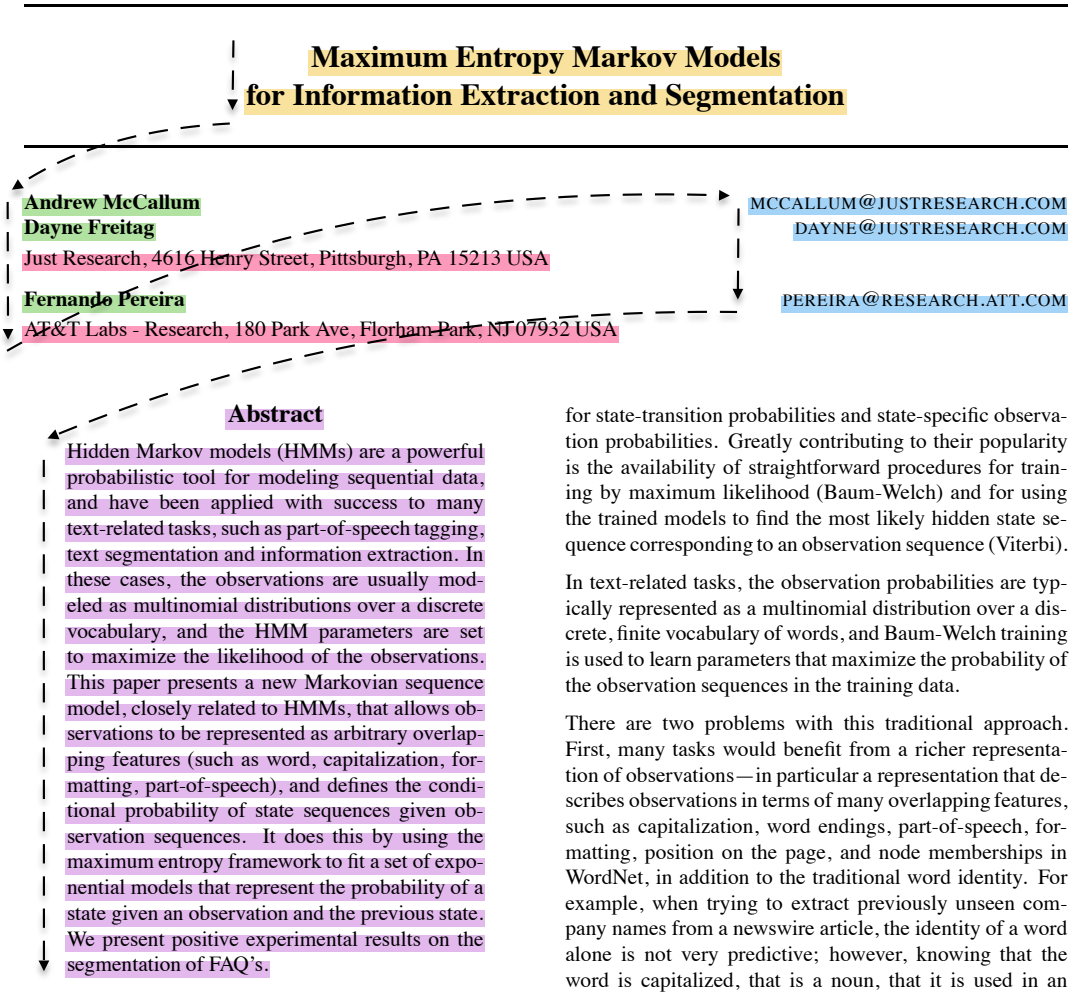


FIGURE 3.1: An illustration of the way a header section might be segmented and classified. The classes modelled are colour-coded (title in yellow, authors in green, and so on). Article excerpt from (McCallum et al. [2000a]).

Stylistic analysis refers to heuristic approaches to analysing physical characteristics of text font and layout. *Knowledge base* methods rely on online repositories to cross-reference extracted information. *Machine learning* refers here either to conditional random fields, or to other approaches, hidden Markov models or support vector machines. The study includes a comparison of the leading AME tools based on an *ad hoc* scoring system over a header test set. GROBID performed best by a considerable margin.

3.3 GROBID

GROBID (GeneRatiOn of BIbliographic Data) (Lopez [2009]) is an open-source (Apache license) Java-based tool for automatic metadata extraction of scientific articles. It has been in development by Patrice Lopez at the French Institute for Research in Computer

Science and Automation (INRIA) since 2008. GROBID manages the training, evaluation and application of a hierarchy of Wapiti-trained CRF models, each addressing a part of the information extraction of scientific articles. Figure 3.4 shows the *cascade* of models used to progressively refine classifications of article content. Through GROBID, higher-tier models such as the *header* and *reference* models may be applied individually to PDFs, while the other, more specific models, such as the *date* model, operate only on plaintext inputs. Moreover, some models, such as *segmentation*, are not intended to be used independently, but rather contribute to the cascade, supplying lower levels with their inputs. For example, reference extraction begins with the *segmentation* model, which classifies each line of a document, resulting in homogeneous blocks of lines, for example, header, paragraph, figure, and references. This information is then distributed to the other models, for example the *reference-segmenter* model, which further breaks down the reference list into individual references. The *citation* model then classifies the parts of each reference into classes, for example, *date*, *affiliation*, and *author*. Finally, the atomic subcomponents of these are classified by their respective models. Note that the *citation* branch of the cascade has the option of further cross-checking extracted references with the third-party CrossRef web service[2]. Thus, the overall accuracy of the system is dependent on the combined accuracy of models in the cascade. Conversely, any errors are propagated down the hierarchy. Though they have much in common, the models vary in the classifications they assign, the features they exploit, and, due to the varying size of the vocabulary (compare say, the number of possible month names to the number of possible author names), the size (dimensionality) of the models. Table 3.1 summarises each of GROBID's models. Calling GROBID function `processFullText` runs all available models on a batch of PDF documents, classifying each entirely.

3.3.1 Text Encoding Initiative

One of GROBID's functions is to transform Wapiti outputs into an output conforming to the Text Encoding Initiative (TEI) standard, therefore we briefly describe it here. TEI is a text encoding standard maintained by the TEI consortium. It specifies an XML representation of a document's contents from the front matter, to the document body, to the document rear. It gives semantic meaning to document components, and can facilitate highly detailed representations. It is therefore apt as a format for annotating a document's metadata. GROBID uses TEI to format its outputs, as well as its training data. In Figure 3.2, we show a sample of a TEI document used to represent a date. The

[2]CrossRef is an online DOI agency with a REST API.

Model	Description	Labels
Header	Classifies front matter	<title>, <author>, <affiliation>, <reference>, <submission>, <abstract>, <address>, <keyword>, <degree>, <pubnum>, <email>, <date>, <copyright>, <intro>, <web>, <note>, <phone>, <dedication>, <entitle>, <grant>, <date-submission>
Affiliation address	Classifies the components of author affiliations and affiliation addresses. Subordinate to the *header* model.	<institution>, <other>, <settlement>, <department>, <postcode>, <country>, <marker>, <region>, <addrLine>, <laboratory>, <postbox>, <other>, <null>
Name/ Header	Classifies an author's full name (as identified by the *header* model) into first and last name etc.	<forename>, <surname>, <marker>, <middlename>, <other>, <suffix>, <title>
Name/ Citation	Classifies an author's full name (as identified by the *citation* model) into first and last name etc.	<surname>, <forename>, <other>, <middlename>
Citation	Classifies a reference into its subcomponents.	<journal>, <volume>, <other>, <issue>, <pages>, <date>, <author>, <title>, <booktitle>, <location>, <pubnum>, <note>, <publisher>, <editor>, <institution>, <tech>, <web>, <issue>
Date	Classifies the components of a date string identified by higher-tier models.	<other>, <day>, <month>, <year>
Segmentation	The highest-level model in the architecture–primarily supplies the three models beneath it (header, fulltext and reference-segmenter) with inputs.	<headnote>, <header>, <body>, <page>, <references>, <footnote>, <cover>, <acknowledgement>, <annex>
Reference-Segmenter	Segments a full reference list into individual citations.	<label>, <reference>, <other>
Fulltext	Classifies elements of article body.	<section>, <paragraph>, <citation_marker>, <other>, <table_marker>, <figure_marker>, <figure_head>, <trash>, <figDesc>, <equation>, <item>

TABLE 3.1: A summary of the models coordinated by GROBID. We have here excluded the Patent, Entities, and E-book models as these are experimental models not currently used by GROBID.

```
<bibl>
    <author>V. Gundelach and D. Eisenburger</author>, &quot;
    <title level="a">Principle of a direction sensitive borehole
    antenna with advanced technology and data examples</title>, &
    quot; in
    <title level="m">Proceedings of the 4th International Workshop
    on Advanced Ground Penetrating Radar (IWAGPR &apos;07)</title>,
     pp.
    <biblScope type="pp">28-31</biblScope>,
    <date>June 2007</date>.
</bibl>
```

FIGURE 3.2: Sample tagged citation for GROBID training input.

XML format can be used to give semantic information about the information enclosed. Thus, a natural mapping exists between the model classes and the XML schema.

3.3.2 Training

In GROBID, models are trained in isolation. The models produced by Wapiti (Section 2.9) are stored in the GROBID project directory. Each model has its own training data, and feature function templates. For certain models, such as the *header* and *segmentation* models, training data consists of pairs of associated files including:

1. a TEI file, and;
2. a raw feature file containing extracted features.

Other, simpler models, require only the former. Together, the files form an abstraction over the CRF engine inputs. The reason for including the feature files is because the original PDF files are assumed not to be available at training time, and therefore any features derived from text styling or positioning must be extracted in advance. The TEI files simply provide the classifications of the plaintext required to build a ground truth. The feature extraction itself is done by a module of GROBID. Since feature files and feature template files, which are configured manually by the developer, must agree, there is a strong coupling between this module and the templates.

3.3.3 Evaluation

Both training and evaluation are performed on sets of TEI documents (see Section 3.3.1 and Figure 3.2). This somewhat paradoxically, when prediction is based on PDF files. However, with closer inspection, an equivalence can be seen between:

1. applying the *pdftoxml* tool, tokenising the output and transforming to CRF input data, and;

2. extracting tokens from TEI documents (and possibly feature files also–see Section 3.3.2), and transforming to CRF input data.

Both approaches yield the same input data for the CRF engine, and so the evaluation inputs are, in effect, the same as for prediction, despite the initial differences. Training may be done with a split defined by the developer, which GROBID uses to set aside a proportion of the training data for evaluation. The evaluation of a model produced by training follows identical procedures, preparing the same input data. The output, however, is not a model but the tagged data. GROBID compares these predictions with the ground truth and reports *accuracy*, *precision*, *recall*, and *F1* scores as performance indicators, at the token, field, and instance levels. A token usually refers to a single contiguous string of characters (without spaces), but in the case of the *segmentation* and *fulltext*, a token is a line. A field is a block of contiguous tokens sharing a class, and an instance is the whole data sample. The accuracy of an instance is therefore judged by the correctness of all tagging for the whole sample (which may be a whole document), a difficult thing to achieve without any mistakes.

3.3.4 Prediction

Figure 3.3 shows the flow of information from input to output, as well as the relationship between training and prediction. When it comes to labelling (prediction), the starting point is a PDF document. With a third-party tool, *pdftoxml*, this is transformed into an XML file containing rich text information (font, style, orientation) for every string token in the document. These tokens are arranged into blocks and features are extracted as they were for training and evaluation. The model created in the training phase is first loaded, and then GROBID calls Wapiti to label the inputs. Unlike for training, the feature template file is not required, as this has already been absorbed into the model file. After processing, Wapiti returns the same file with classifications inserted. GROBID then further processes this information to transform it into the final TEI output format.

3.3.5 Other Functionality

GROBID provides a means of producing training sets semi-automatically. This consists of applying the existing models on a PDF batch to produce the XML inputs. Each field must be checked manually against a ground truth and errors corrected before it is used as a training set. We use this feature to generate HEP training data (see Chapter 4).

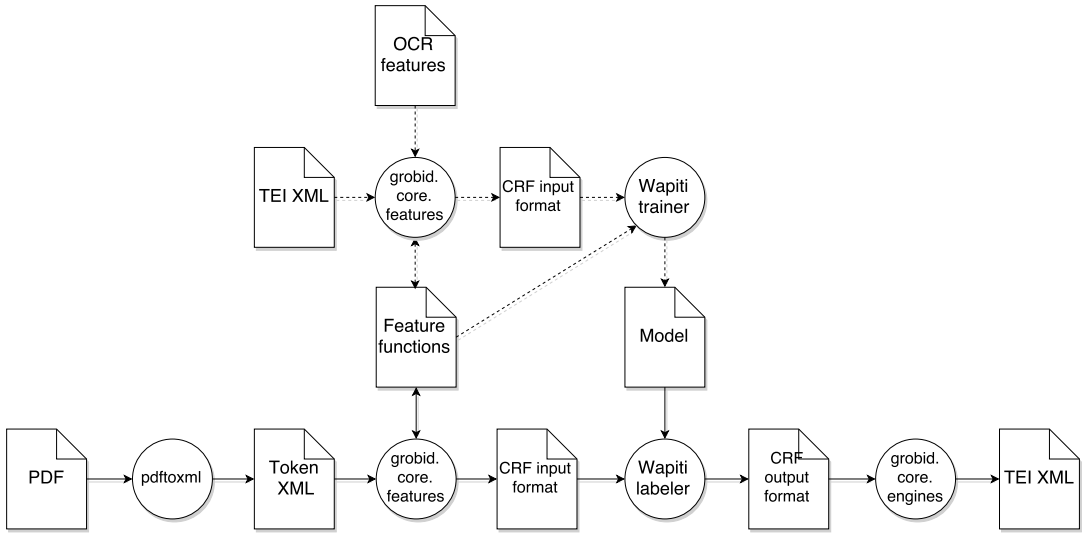


FIGURE 3.3: An illustration of the interactions between GROBID and Wapiti for the two main functions of training and tagging. The dashed arrows indicate training operations; the solid arrows, tagging.

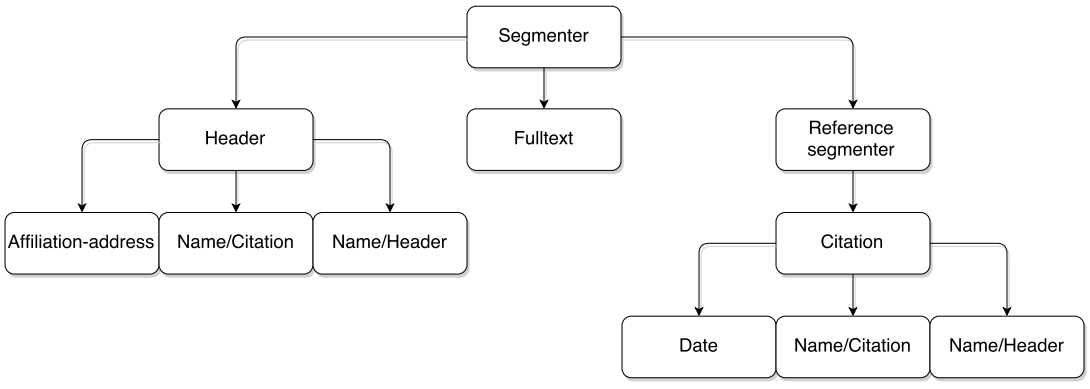


FIGURE 3.4: The models of GROBID are organised into a cascade, where each part of a document is classified in increasingly greater detail.

Chapter 4

Data, Methods, and Implementation

This chapter presents the different strands of our work. The work is divided into three components: first, the procurement of HEP training data on which to perform experiments; the implementations of extensions to GROBID to facilitate the feature engineering and evaluations, as well as the pipeline assembled for automating the experimentation; and finally, the different categories of feature engineering, and our reasons for choosing them, supplying, where relevant, statistical analyses to support our intuitions.

4.1 Assessments

As articulated in Chapter 3, GROBID manages a hierarchy of models that propagates classified information from top to bottom in a cascade. Our objectives are therefore to enhance some models within the cascade for HEP papers. It does not, on the other hand, make sense to attempt to improve all models. After all, we may assume a HEP *date* is no different from dates printed in other scientific papers. The same goes for author names, and with little exception[1], reference lists and their contents. Undoubtedly, the models with the most promising scope for improvement are the *header* and *segmentation* models. It is these models that address the parts of an article most distinct in scientific papers. Specifically, physics journals have recurring styles and layouts for headers sections that are distinct from others publishers. In addition, the vocabulary of a HEP paper header will be distinct from that of papers from other branches of science. These should be both trained for and engineered for, for example through the use of dictionary-based

[1]It is in fact true that HEP collaborations feature in isolated references in HEP papers (see Section 6.2).

features (see Section 4.3.4). Exceptionally, the *header* model is not, by default, part of the cascade, rather, the header section is extracted separately using heuristics based on locating the start of the body (usually identified by the heading, 'Introduction'). The reason for this approach is that, on average, it is more accurate than relying on the *segmentation* model for finding front matter, given the limited amount of *segmentation* training data. However, this precludes the modelling of *discontinuous* front matter, which may occur at the base of a first page, at the very end of an article, or just about anywhere else (see Figure 4.1). Reconnecting the *header* model with with the *segmentation* model is therefore an implementation objective. Thus, the *header* model may be improved for a number of reasons, including:

1. physics publishers present a format unique to HEP papers;
2. scientific collaborations as seen in HEP papers are not modelled as a *header* class by vanilla GROBID, and;
3. discontinuous header data (see Figure 4.1), which may contain substantial front matter is by default neither trained nor modelled for.

The *segmentation* model may also be improved for a number of reasons:

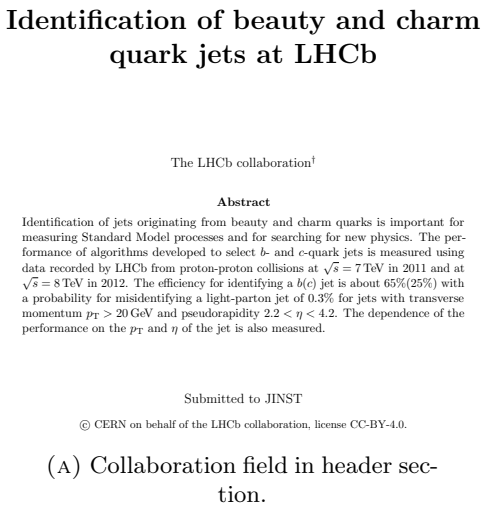
1. discontinuous header data (see Figure 4.1), which may contain substantial front matter is neither trained nor modelled for;
2. HEP collaborations entail long author lists and affiliation lists, often disjoint from the main header section, which are neither trained nor modelled for, and;
3. the dataset is small (we more than double it in Section 4.2).

Note that the *segmentation* model is the parent model of the entire cascade, and therefore any improvement to it will benefit all other models at prediction time. Aside from being the root of the cascade, the *segmentation* model is special in that it models a full line at the token level, rather than a single character string such as for the *header* model. We are mindful of this distinction as we go about our feature engineering. Our focus is therefore on the two models, *header* and *segmentation*. Hence, we require two separate training sets of HEP papers. Both models require that for each paper we produce a TEI representation and a feature file of extracted features, as introduced in Section 3.3.

4.2 Data Acquisition

The starting point for generating training data is to apply the existing, shipped models of GROBID on our new dataset of PDF papers. This creates a pair of TEI and feature

file for each document, in a first attempt at a ground truth, and a structure on which to work. Though this is greatly preferable to starting from scratch, the researcher must then manually correct the inevitable myriad errors to achieve a gold standard of training data. In the case of *segmentation*, this involves the validation of a full document, which may contain many recurring misclassifications, rendering the task stupendously time-consuming and a barrier to actual research. In the case of the *header* model, this is often simpler, as it involves validating a single section. However, wherever discontinuous front matter exists, changes must be made to both the TEI *and* feature files. This first involves a modification to GROBID such that it extracts features for all tokens rather than just those contained in the header. Then, the unseen front matter must be manually formatted and appended to the TEI file, and the corresponding features copied to the feature file. This is error-prone, as there must be a one-to-one correspondence between



Identification of beauty and charm quark jets at LHCb

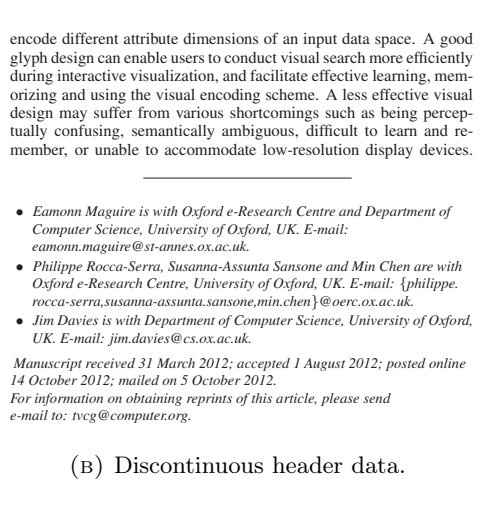
The LHCb collaboration†

Abstract

Identification of jets originating from beauty and charm quarks is important for measuring Standard Model processes and for searching for new physics. The performance of algorithms developed to select b- and c-quark jets is measured using data recorded by LHCb from proton-proton collisions at $\sqrt{s} = 7$ TeV in 2011 and at $\sqrt{s} = 8$ TeV in 2012. The efficiency for identifying a $b(c)$ jet is about 65%(25%) with a probability for misidentifying a light-parton jet of 0.3% for jets with transverse momentum $p_{\rm T} > 20$ GeV and pseudorapidity $2.2 < \eta < 4.2$. The dependence of the performance on the $p_{\rm T}$ and η of the jet is also measured.

Submitted to JINST

(A) Collaboration field in header section.

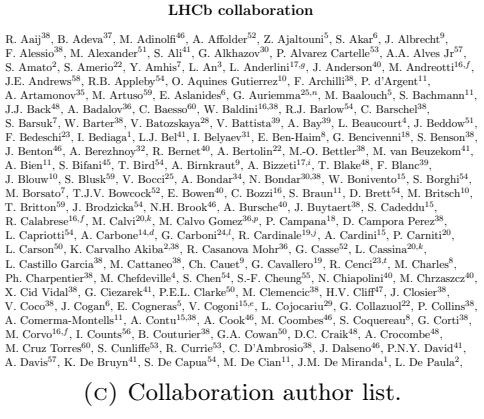


encode different attribute dimensions of an input data space. A good glyph design can enable users to conduct visual search more efficiently during interactive visualization, and facilitate effective learning, memorizing and using the visual encoding scheme. A less effective visual design may suffer from various shortcomings such as being perceptually confusing, semantically ambiguous, difficult to learn and remember, or unable to accommodate low-resolution display devices.

- *Eamonn Maguire is with Oxford e-Research Centre and Department of Computer Science, University of Oxford, UK. E-mail: eamonn.maguire@st-annes.ox.ac.uk.*
- *Philippe Rocca-Serra, Susanna-Assunta Sansone and Min Chen are with Oxford e-Research Centre, University of Oxford, UK. E-mail: {philippe.rocca-serra,susanna-assunta.sansone,min.chen}@oerc.ox.ac.uk.*
- *Jim Davies is with Department of Computer Science, University of Oxford, UK. E-mail: jim.davies@cs.ox.ac.uk.*

Manuscript received 31 March 2012; accepted 1 August 2012; posted online 14 October 2012; mailed on 5 October 2012.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

(B) Discontinuous header data.



LHCb collaboration

R. Aaij[38], B. Adeva[37], M. Adinolfi[46], A. Affolder[52], Z. Ajaltouni[5], S. Akar[6], J. Albrecht[9], F. Alessio[38], M. Alexander[51], S. Ali[41], G. Alkhazov[30], P. Alvarez Cartelle[53], A.A. Alves Jr[57], S. Amato[2], S. Amerio[22], Y. Amhis[7], L. An[3], L. Anderlini[17,g], J. Anderson[40], M. Andreotti[16,f], J.E. Andrews[58], R.B. Appleby[54], O. Aquines Gutierrez[10], F. Archilli[38], P. d'Argent[11], A. Artamonov[35], M. Artuso[59], E. Aslanides[6], G. Auriemma[25,n], M. Baalouch[5], S. Bachmann[11], J.J. Back[48], A. Badalov[36], C. Baesso[60], W. Baldini[16,38], R.J. Barlow[54], C. Barschel[38], S. Barsuk[7], W. Barter[38], V. Batozskaya[28], V. Battista[39], A. Bay[39], L. Beaucourt[4], J. Beddow[51], F. Bedeschi[23], I. Bediaga[1], L.J. Bel[41], I. Belyaev[31], E. Ben-Haim[8], G. Bencivenni[18], S. Benson[38], J. Benton[46], A. Berezhnoy[32], R. Bernet[40], A. Bertolin[22], M.-O. Bettler[38], M. van Beuzekom[41], A. Bien[11], S. Bifani[45], T. Bird[54], A. Birnkraut[9], A. Bizzeti[17,i], T. Blake[48], F. Blanc[39], J. Blouw[10], S. Blusk[59], V. Bocci[25], A. Bondar[34], N. Bondar[30,38], W. Bonivento[15], S. Borghi[54], M. Borsato[7], T.J.V. Bowcock[52], E. Bowen[40], C. Bozzi[16], S. Braun[11], D. Brett[54], M. Britsch[10], T. Britton[59], J. Brodzicka[54], N.H. Brook[46], A. Bursche[40], J. Buytaert[38], S. Cadeddu[15], R. Calabrese[16,f], M. Calvi[20,k], M. Calvo Gomez[36,p], P. Campana[18], D. Campora Perez[38], L. Capriotti[54], A. Carbone[14,d], G. Carboni[24,l], R. Cardinale[19,j], A. Cardini[15], P. Carniti[20], L. Carson[50], K. Carvalho Akiba[2,38], R. Casanova Mohr[36], G. Casse[52], L. Cassina[20,k], L. Castillo Garcia[38], M. Cattaneo[38], Ch. Cauet[9], G. Cavallero[19], R. Cenci[23,t], M. Charles[8], Ph. Charpentier[38], M. Chefdeville[4], S. Chen[54], S.-F. Cheung[55], N. Chiapolini[40], M. Chrzaszcz[40], X. Cid Vidal[38], G. Ciezarek[41], P.E.L. Clarke[50], M. Clemencic[38], H.V. Cliff[47], J. Closier[38], V. Coco[38], J. Cogan[6], E. Cogneras[5], V. Cogoni[15,e], L. Cojocariu[29], G. Collazuol[22], P. Collins[38], A. Comerma-Montells[11], A. Contu[15,38], A. Cook[46], M. Coombes[46], S. Coquereau[8], G. Corti[38], M. Corvo[16,f], I. Counts[56], B. Couturier[38], G.A. Cowan[50], D.C. Craik[48], A. Crocombe[48], M. Cruz Torres[60], S. Cunliffe[53], R. Currie[53], C. D'Ambrosio[38], J. Dalseno[46], P.N.Y. David[41], A. Davis[57], K. De Bruyn[41], S. De Capua[54], M. De Cian[11], J.M. De Miranda[1], L. De Paula[2],

(C) Collaboration author list.

[18] *Laboratori Nazionali dell'INFN di Frascati, Frascati, Italy*
[19] *Sezione INFN di Genova, Genova, Italy*
[20] *Sezione INFN di Milano Bicocca, Milano, Italy*
[21] *Sezione INFN di Milano, Milano, Italy*
[22] *Sezione INFN di Padova, Padova, Italy*
[23] *Sezione INFN di Pisa, Pisa, Italy*
[24] *Sezione INFN di Roma Tor Vergata, Roma, Italy*
[25] *Sezione INFN di Roma La Sapienza, Roma, Italy*
[26] *Henryk Niewodniczanski Institute of Nuclear Physics Polish Academy of Sciences, Kraków, Poland*
[27] *AGH - University of Science and Technology, Faculty of Physics and Applied Computer Science, Kraków, Poland*
[28] *National Center for Nuclear Research (NCBJ), Warsaw, Poland*
[29] *Horia Hulubei National Institute of Physics and Nuclear Engineering, Bucharest-Magurele, Romania*
[30] *Petersburg Nuclear Physics Institute (PNPI), Gatchina, Russia*
[31] *Institute of Theoretical and Experimental Physics (ITEP), Moscow, Russia*
[32] *Institute of Nuclear Physics, Moscow State University (SINP MSU), Moscow, Russia*
[33] *Institute for Nuclear Research of the Russian Academy of Sciences (INR RAN), Moscow, Russia*
[34] *Budker Institute of Nuclear Physics (SB RAS) and Novosibirsk State University, Novosibirsk, Russia*
[35] *Institute for High Energy Physics (IHEP), Protvino, Russia*
[36] *Universitat de Barcelona, Barcelona, Spain*
[37] *Universidad de Santiago de Compostela, Santiago de Compostela, Spain*
[38] *European Organization for Nuclear Research (CERN), Geneva, Switzerland*
[39] *Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland*
[40] *Physik-Institut, Universität Zürich, Zürich, Switzerland*
[41] *Nikhef National Institute for Subatomic Physics, Amsterdam, The Netherlands*
[42] *Nikhef National Institute for Subatomic Physics and VU University Amsterdam, Amsterdam, The Netherlands*
[43] *NSC Kharkiv Institute of Physics and Technology (NSC KIPT), Kharkiv, Ukraine*
[44] *Institute for Nuclear Research of the National Academy of Sciences (KINR), Kyiv, Ukraine*

(D) Collaboration affiliation list.

FIGURE 4.1: Figure (A) shows a collaboration field in a header section. Figure (B) shows discontinuous front matter that sits on the first page, but apart from the main header section and within the introductory section. Figures (C) and (D) give the authors list and affiliations for a large HEP collaboration; the author list begins on page 8 and continues to page 33. Figure (B) from (Maguire et al. [2012]), other excerpts from (Aaij et al. [2015]).

a token in the TEI file and its features in the feature file. Hence, any copyist mistake will invoke errors at training time. In Sections 4.2.1, 4.2.2, we detail the mixture of training data that we have assembled. Table 4.1 gives a breakdown of the data for each model.

Model	HEP	CORA
Header	157 papers	**2506 papers**
Segmentation	**169 papers**	125 papers

TABLE 4.1: Number of training instances for each model from each dataset.

4.2.1 CORA dataset

The CORA dataset (McCallum et al. [2000b]) is a substantial dataset of annotated documents. It is popular in metadata extraction studies, and has come to be a sort of standard due to the difficulty of creating custom data (Peng and McCallum [2004]). We use it in combination with our own HEP dataset. For the *segmentation* model, the baseline dataset is not part of CORA, but rather part of the GROBID project, but for brevity we refer to this also as *CORA*.

4.2.2 HEP dataset

At the recommendation of an INSPIRE-HEP digital library curator, we selected a set of articles deemed to be a representative sample of the database. It contains the following varieties of papers:

1. conference papers (with DOI);
2. conference papers (without DOI);
3. miscellaneous papers (including non-English language);
4. collaboration papers, and;
5. general papers.

This totalled 191 papers[2], however we additionally removed documents found to be unsuitable for training, such as book-length article compendiums. To start creating HEP *header* model training data, we execute GROBID command, `createTrainingHeader`. Modifications were made to GROBID to produce header features for an entire document, rather than just the first two pages as per the default. This was essential in order to obtain features for any discontinuous front matter in other parts of a document. Wherever such

[2]Originally this numbered in excess of 200, but certain papers could not be parsed by *pdf2xml*.

matter was found, it was necessary first to manually append this material to the TEI file, following the TEI XML standard, then to copy the discontinuous segments of the extracted features into the master copy feature file. Whenever a copyist mistake was made, this entailed runtime errors, and lengthy corrections. For *segmentation* training data, we run command `createTrainingSegmentation`. The starting point for building a *segmentation* training set is invariably worse than for *header*, due to the lower accuracy of this model. Typically each instance contains many recurrent misclassifications which sometimes may be corrected efficiently using regular expressions.

4.3 Methods

Here we list the ideas for feature functions that we have implemented and cross-validate for in Chapter 5. The features were extracted either with changes made to GROBID directly, or by modifying baseline features with an external script from our pipeline (Section 4.4.2). In the following we use a notation consistent with that introduced in Chapter 2, that is, x represents a token, $\mathbf{x}$ an instance, and y and $\mathbf{y}$ the corresponding label and label vectors.

4.3.1 Baseline

To set a benchmark by which to compare our results, we ran a series of experiments including the default features for GROBID, training on different combinations of CORA and HEP data (see Section 5.2). In addition, we examined the effects of ramping up the size of the CORA set appended during training. Baseline features depend on the model, but generally include token identities, prefixes, suffixes, indicators of capitalisation, punctuation, and numeric characters, font information, dictionary membership, and physical positioning. Our new features were combined with the baseline features unless otherwise noted.

4.3.2 Block Size

A block is a collection of lines physically grouped together. The block to which each token belongs is specified in the *pdftoxml* outputs. One characteristic that is distinct about article layout are the varied dimensions of blocks. This is most striking in the header section, where different blocks manifest in distinct sizes. For example, an article title is usually wider than any other element, an abstract is the tallest element, and so on. It seems reasonable that the dimensions of a block to which a token belongs may

provide information on the class of the token. We therefore devise features based on the pixel lengths and widths of header text blocks, information that comes from *pdftoxml*. We try four variations on this idea: block width, block height, block width and height together, and block area, each normalised by the largest block dimensions. For example, for area, the feature function is,

$$\hat{f}_{size}(x_t) = \frac{\sum_{b \in B} \mathbb{1}_{\{x_t \in b\}} \cdot \text{height}(b) \cdot \text{width}(b)}{\max_{b' \in B} \text{height}(b') \cdot \max_{b'' \in B} \text{width}(b'')}, \tag{4.1}$$

where x_t is a token (hence a word for the *header* model) and B is the set of blocks in the header instance. We then define,

$$f_{size}(x_t) = \lfloor C \cdot \hat{f}_{size}(x_t) \rfloor, \tag{4.2}$$

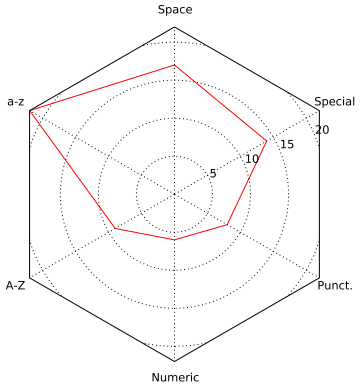
where C is a discretisation factor. In our experiments $C = 10$, giving a categorical variable of 10 values.

4.3.3 Character Classes

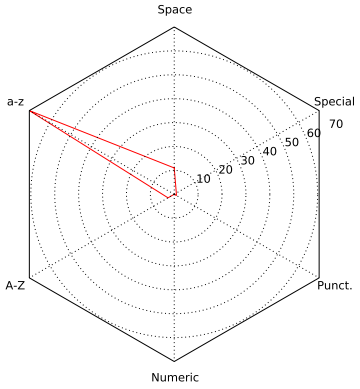
A visual scan of any scientific paper allows one to see that lines from different sections are most easily distinguished by their composition of characters. It therefore stands to reason that we can build informative features for the *segmentation* model on this basis. Indeed, it may be that a line is more effectively characterised at the *character* level than the *word* level. For an illustration of this effect, see Figure 4.2. Note that the baseline feature function set does include some basic capitalisation and punctuation indicators, but we advocate our approach for several reasons:

1. it is more complete in that it models more character classes;
2. it does this systematically in a feature framework that is easily modified or extended, and;
3. it performs better (see Chapter 5).

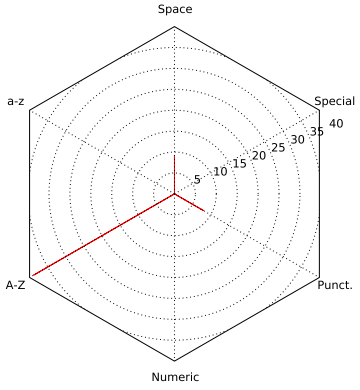
In Table 4.2, we give a list of the character classes used to model features. The regular expressions (regexes) were used to count the number of characters in a token (line) belonging to each class. This was then normalised over the line length. Because such a result is numeric, we have necessarily to discretise it. We tried four different discretisation strategies: binary (according to some *ad hoc* threshold), decimal (round down),



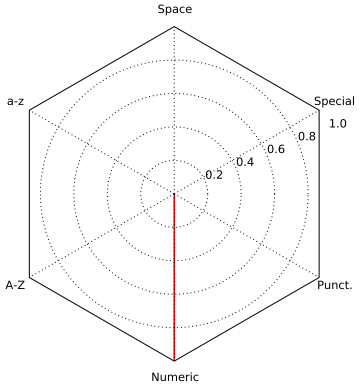
(A) Body (formula)



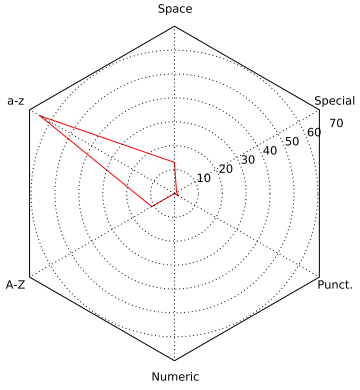
(B) Body (normal)



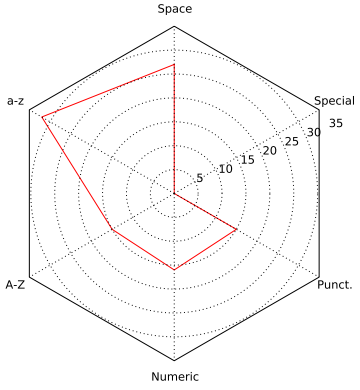
(C) Headnote



(D) Page number



(E) Affiliation list



(F) Author list

FIGURE 4.2: Character class breakdown of sample lines from different sections of a CERN LHCb collaboration paper. The paper in question is the current world record holder for number of authors, and lists over 5000 authors and their affiliations. The radar plots give a different impression for each of the samples.

decimal (round to nearest), and 20-point discretisation[3]. For the decimal case (rounding down),

$$\hat{f}_{\text{class}_i}(x_t) = \frac{1}{|x_t|} \sum_{n=1}^{|x_t|} \mathbb{1}_{\{x_{ti} \in \text{class}_i\}}, \tag{4.3}$$

for each character class, class_i, where x_t is a token (hence a line for the *segmentation* model), and x_{ti} is the ith character in the line. For the decimal (round down) case, we then define,

$$f_{\text{class}_i}(x_t) = \lfloor C \cdot \hat{f}_{\text{class}_i}(x_t) \rfloor, \tag{4.4}$$

where C is the discretisation factor and $C = 10$. The other discretisation strategies may be defined similarly.

Class	Regex
Spacing	r‘[\s]’
Lower case	r‘[a-z]’
Upper case	r‘[A-Z]’
Numeric	r‘[\d]’
Punctuation	r‘[.,?:;]’
Special character	r‘[^\sa-zA-Z d.,?:;]’

TABLE 4.2: Character classes used as features, along with the regular expressions used to count them.

4.3.4 Dictionaries

As mentioned previously, the vocabulary of a HEP paper *header* is distinct from other branches of science, containing jargon and terminology particular to high energy physics. Aside from token indicators, we can support this characteristic with features based on dictionary membership. To achieve this, we exported the full set of author names, affiliations, journal names, article titles, and collaborations from the INSPIRE-HEP database. These dictionaries were then tokenised to give a set of *words* rather than phrases. Each dictionary required extensive cleaning prior to being fit for purpose. In addition, we used the Natural Language Toolkit (`nltk`) for Python to create a dictionary of stop words, the expectation being that different classes contain stop words in varying amounts. For example, prosaic text, such as an abstract, will contain stop words (the, a, an, it, etc.) in greater number than summary information such as a keyword term,

[3] In this final discretisation case we categorised results by each $5th$ percentage point, capping at 50%, such that we model 10 categories in total.

or an author details block. We confirmed this hypothesis with statistical ANOVA and pairwise t-tests in *R*, showing significant differences of stop word frequency according to header section (see Appendix C.1). The dictionary feature functions may therefore be written formally as,

$$f_{\text{dict}_i}(x_t) = \mathbb{1}_{\{x_t \in \text{dict}_i\}}, \tag{4.5}$$

for each dictionary, dict_i. These features did not require modifications to GROBID directly, and were instead created by editing baseline feature files with pipeline script, `feature_modifier.py`.

4.3.5 Levenshtein Distance

The Levenshtein or *edit* distance can be used to quantify the edit distance between two strings, a and b, by counting the number of changes, insertions, or deletions required for transforming a into b. Beginning with $\text{lev}_{a,b}(|a|, |b|)$, the recursive step is defined to be,

$$\text{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if min(i, j)} = 0 \\ \min \begin{cases} \text{lev}_{a,b}(i-1,j)+1 \\ \text{lev}_{a,b}(i,j-1)+1 \\ \text{lev}_{a,b}(i-1,j-1)+1_{a_i \neq b_j} \end{cases} & \text{otherwise} \end{cases} \tag{4.6}$$

With some exploratory data analysis we may see that the average edit distance varies between different sections, in particular at the transition points between these sections. It therefore stands to reason that the Levenshtein distance may be used as a feature of line tokens in the *segmentation* model. Therefore, we define a similarity measure based on the Levenshtein distance, first normalising the distance between a line and its precursor, by dividing by the length of the longer of the two, before subtracting this result from 1, to give a measure of similarity,

$$\text{similarity}(a,b) = 1 - \frac{\text{lev}_{a,b}(|a|,|b|)}{\max(|a|,|b|)}. \tag{4.7}$$

Due to the constraints on numeric features (see Section 2.9), we must discretise the result. Thus, for a given line, x_t, we define the feature function,

$$f_{lev}(x_t) = \begin{cases} 0 & \text{if } 0 \leq \text{similarity}(x_t, x_{t-1}) \leq T_1 \\ 1 & \text{if } T_1 \leq \text{similarity}(x_t, x_{t-1}) \leq T_2 \\ \vdots & \quad \vdots \\ \text{N-1} & \text{if } T_{N-1} \leq \text{similarity}(x_t, x_{t-1}) \leq 1 \end{cases} \tag{4.8}$$

where $T_1, T_2, ..., T_{N-1}$ are thresholds selected to create the N categories. We try several thresholding strategies in our experimentation (see Section 5.2).

4.3.6 Regularisation

We additionally cross-validated the tuning parameter for the model, the variance for l_2 regularisation, σ^2, for values on a logarithmic scale: $\sigma^2 = 0$, $\sigma^2 = 10^{-6}$, $\sigma^2 = 10^{-5}$[4], $\sigma^2 = 10^{-4}$, and $\sigma^2 = 10^{-3}$.

4.3.7 Token Extensions

In the baseline *segmentation* model, only the first two words from each line are modelled as features. The model might therefore benefit from modelling more of the line. However, unlike character class features (Section 4.3.3), modelling the words does not reduce the dimensionality of the token. The signal from these token extensions may therefore be too diffuse to make useful indicators. We nevertheless try four variations on this idea, extending the feature set to model the first 5, 10, 15, and 20 words of each line.

4.4 Implementation

4.4.1 Extensions to GROBID

The effects of our extensions are seen in Sections 4.4.2, 4.3 and finally in Chapter 5, and wherever appropriate we indicate them. Extensions were made as a branch of GROBID in the following ways:

1. reconnecting the *segmentation* and *header* models;
2. modelling HEP specific header field, *collaboration*;
3. producing new features;

[4]Wapiti default value.

```
(<header>, <body>)
        1343657.training.segmentation.tei.xml - 0.7292 (35)
        1342206.training.segmentation.tei.xml - 0.3571 (5)
        1345915.training.segmentation.tei.xml - 0.3333 (7)
        1344707.training.segmentation.tei.xml - 0.1765 (3)
        1347299.training.segmentation.tei.xml - 0.1667 (8)
```

FIGURE 4.3: Misclassification of <header> to <body> proportion and count (given in parentheses) for five papers in an evaluation fold, an output of the confusion matrix utility.

```
<author>
    <orgName>ALICE Collaboration</orgName>
</author>
```

FIGURE 4.4: Example of a successfully classified collaboration. The choice of XML tags is ours and was selected to be consistent with the TEI standard.

4. logging results automatically, and;

5. extending the evaluation utilities for our analysis and reporting aims.

The most significant extension made was with the class, `ConfusionMatrix.java`, used within GROBID's `EvaluationUtilities.java` framework to create confusion matrix outputs, ultimately visualised by the pipeline (see Chapter 5). These allowed us to see which misclassifications were made most frequently. In addition, `ConfusionMatrix.java` tracks the documents on which the the model committed the most errors for each misclassification. A sample output is given in Figure 4.3. Modelling collaborations as a class of the *header* model involved extending GROBID's training and tagging modules alike. An XML structure conforming to the TEI standard (Section 3.3.1) was selected. An example of a successful prediction of a collaboration is given in Figure 4.4.

4.4.2 Experiment Pipeline

To automate the experimentation process, we developed a pipeline of scripts in Python, the language chosen for its ubiquity in INSPIRE-HEP. The repository is open source and hosted on GitHub[5]. The pipeline begins with using GROBID (including all necessary extensions) to generate raw feature files for all TEI files in the ground truth dataset. These are then manually assembled into a stripped-down copy of GROBID (containing all necessary Java executables and additional data) and placed in directory, `batches/`, with other scenarios. The assemblage of training data depends on the data scenario desired. If we wish to cross-validate over all data, we simply ensure that both the `training/` and `evaluation/` directories within the GROBID project contain all the feature files,

[5] https://github.com/jcboyd/pykelet/src

and that all TEI files are in `training/`. The cross validation (CV) process will move a fold from the training directory to the evaluation directory, and return it when the iteration is complete. If, however, we wish to *append* data for training, yet exclude it from CV and evaluation, we place the features files only under `training/`. The process will then cross-validate only on those files present in `evaluation/`. GROBID requires both TEI and feature files to be present or else they are ignored. This trick allows us to run such complex experiments without modifying GROBID or its directory structure. Each iteration of CV produces a log file of results exported from GROBID's evaluation utilities. Another script then aggregates the file contents (token- and field-level performance metrics and confusion matrices) and visualises them automatically. The pipeline process is depicted in Figure 4.5. The pipeline consists of a Python wrapper for GROBID, `grobid.py`, written using `pyjnius`, a Python library for the manipulation of Java executables through the Java Native Interface (JNI)[6]. For an iteration of CV, (`k_fold_cross_validation.py`), our wrapper may be used in the following way:

```
grobid_trainer = GrobidTrainer(classpath=classpath_trainer,
                               grobid_home=grobid_home)
grobid_trainer.train(model)
grobid_trainer.evaluate(model)
```

Listing 4.1: Excerpt from our Python wrapper for GROBID

The process uses Python scientific computing library, `numpy`, to randomly shuffle training data (according to a fixed seed), and then Python machine learning library `scikit-learn` to create CV folds. The folds are withheld during training and are evaluated upon in the conventional way. The output of CV is a set of log files containing results. These results are processed by further scripts to produce visualisations of confusion matrices and performance metric comparisons.

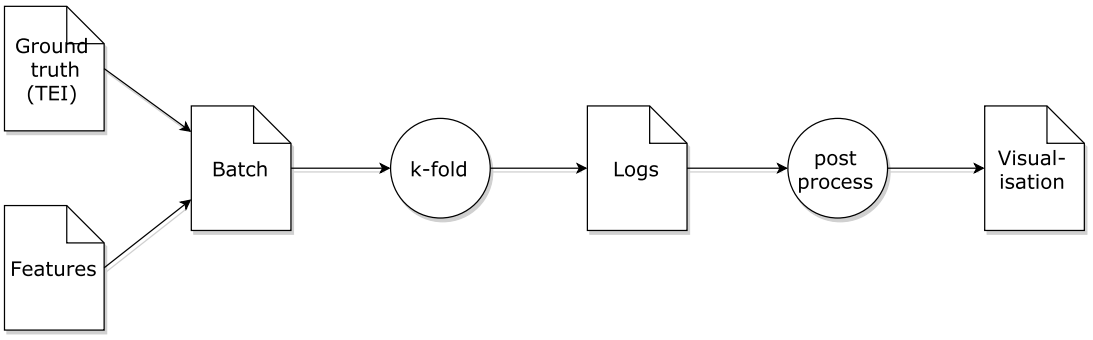


Figure 4.5: An illustration of the experimentation pipeline.

[6]Certain limitations on this led us to write a simpler Python `subprocess`-based wrapper also.

Chapter 5

Results and Analysis

This chapter presents the results of our implementation work, which comprises of 66 cross-validated feature engineering experiments including a baseline evaluation. It notably begins with a comparison between GROBID and refextract, the existing partial solution for metadata extraction within INSPIRE-HEP. Following this, the evaluation method and approach to running experiments are detailed. Finally, the experimental results are presented and we provide our analysis and interpretations. The results are first shown by category, in accordance with the methods described in Chapter 4, then inter-category results are shown to highlight the most significant improvements.

5.1 Evaluation Method

Cross-validation (see Section 5.2) is use to give an estimation of model generalisation error. When it comes to reporting results, we focus on the cross-validated performance metric, micro average (see Section 5.1.1). This is more informative than a *macro* average due to the skew in class representation. For example, the <abstract> class contains on average far more tokens than any other class in the *header* model; likewise, <body> for *segmentation*. The micro average effectively gives a weighted average that says more about general token accuracy. Where necessary, we further look at model performance in individual fields. For the *segmentation* model, the key fields are <header> and <references>. For the *header* model, they are <title>, <authors>, and <abstract>.

5.1.1 Evaluation Metrics

The findings of this chapter refer to various standard measures of classification performance. We define these presently. Accuracy is defined to be,

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN}, \tag{5.1}$$

that is, the proportion of correct classifications to total classifications, where TP is the number of *true positives*, the number of times a class is correctly predicted to have occurred; TN is the number of *true negatives*, the number of times a given class is correctly predicted not to have occurred; FN is the number of *false negatives*, the number of times a class is incorrectly predicted to have occurred; and FN is the number of *false negatives*, the number of times a class is incorrectly predicted not to have occurred. Accuracy can be a misleading statistic when we have uneven representations of classes in the dataset. In the event that we have a sufficiently high bias, we can achieve excellent accuracy simply by always predicting the dominant class. For this reason, we consider other statistics too. *Precision* is the number of times a class is *correctly* predicted proportional to the overall number of predictions for that class, that is,

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{5.2}$$

This, however, does not inform us as to whether we have missed any occurrences of the class, which would be shown in the number of false negatives, FN. We could therefore have a very high precision with limited accuracy. *Recall* is the number of times a class is *correctly* predicted proportional to the number of occurrences of that class (equivalently, the accuracy with respect to the class), that is,

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{5.3}$$

However, a simple strategy of always predicting one class will give perfect recall for that class, because then misclassifications are only captured by FP. The F_1 statistic is a common measure used to assess classifiers that combines precision and recall, and is defined as,

$$F_1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}, \tag{5.4}$$

that is, the harmonic mean of precision and recall (the "1" in F_1 indicates the two are evenly weighted). The F_1 statistic is a neat way of summarising both metrics at once. Furthermore, a large imbalance in precision and recall results in a lower F_1 score. It is necessary to be good in both precision and recall to have a good F_1 score; the harmonic mean of any data is always upper-bounded by its arithmetic mean. Thus, the F_1 score

addresses their shortcomings simultaneously. To summarise each of these statistics over a set of classes, we may adopt two approaches: macro and micro averages. A macro average is the aggregation of statistics *a posteriori*. For example, for accuracy,

$$\text{Accuracy}_{macro} = \frac{1}{N}\sum_{n=1}^{N}\text{Accuracy}_n, \tag{5.5}$$

where Accuracy_n is the accuracy for the *nth* of N classes. By contrast, a micro average is an aggregation of statistics that is in effect weighted by the proportion of each class. For example, again for accuracy,

$$\text{Accuracy}_{micro} = \frac{\sum_{n=1}^{N} TP_n + TN_n}{\sum_{n=1}^{N} TP_n + FP_n + FN_n + TN_n}, \tag{5.6}$$

5.1.2 Evaluation in GROBID

GROBID calculates the aforementioned statistics for each of the classes in each model. In our results (Section 5.4), we concentrate on the F_1 score micro average and scores for key classes (depending on the model we consider) at the *token* level. We supplement the GROBID evaluation output with our own confusion matrices, an example of which is shown in Figure 5.2. Whereas the metrics allow us to compare one model to another, a confusion matrix can be used to see exactly which misclassifications are being made, which can in turn inform our feature engineering.

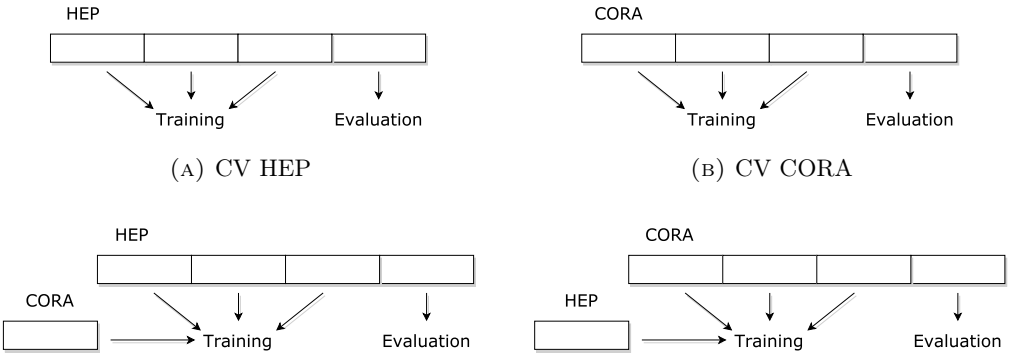
5.2 Experiment Setup

Our computing resources for experimentation consisted of two powerful virtual machines on the CERN LXPLUS cluster, each possessing 16 CPUs and 32 GB of RAM. The experiments were configured and uploaded to these machines in batches, and were processed by our experimentation pipeline (see Section 4.4.2). The high dimensionality of the models (up to tens of millions of features) led to long runtimes. To control the runtime of training, we enforced a maximum number of 500 iterations for Wapiti's L-BFGS algorithm. This number was chosen from observing the diminishing improvements of models trained to this extent[1]. With Wapiti parallelised to 8 cores, we were able to run two processes on each virtual machine when required. Even with this parallelised setup, our experiment batches took several days to process each time, and the sum total of our experiments amounts to perhaps months of CPU time. Recall that model training

[1]There is also the argument that training to convergence may cause overfitting.

time is dependent on both the number of training samples and the model dimensionality, which is itself dependent on the number of samples (see Section 2.5).

In conjunction with our feature engineering variations, we tried different configurations of data. Figure 5.1 illustrates our four approaches to cross-validation with different combinations of HEP and CORA training data. Where we *append* data, we include it in training, but exclude it from evaluation. Thus, HEP app. CORA denotes the training of a model on HEP and CORA combined, but cross-validation and evaluation only on HEP. Of most interest, naturally, were those configurations evaluating purely on HEP papers, that is, those we denote *HEP* and HEP app. CORA, and these were the only configurations run beyond the baseline. In Section 5.4, we refer to these configurations as we present the results. All experiments were run with 5−fold cross validation[2]. The dataset was randomly shuffled prior to cross-validation, but with a fixed seed, such that models trained on the same data configuration could be compared equitably.



(A) CV HEP (B) CV CORA

(C) CV HEP append CORA (D) CV CORA append HEP

FIGURE 5.1: The different cross-validation configurations used in our experiments. Figures (A) and (B) show cross-validation on HEP and CORA sets independently. Figures (C) and (D) show cross-validation on the HEP and CORA datasets respectively, appending the other at training time.

The variety of 66 experiments that we cross-validated are presented in Table 5.1, organised into 8 categories. We also distinguish by model, running some experiments for both models, and some for one alone. Generally speaking, we chose feature engineering ideas with a particular model in mind, that is, either *header* or *segmentation*. Finally, we distinguish by data configuration. For dictionary-based features, which may be derived from a baseline feature file alone, we may try all data configurations. However, as we do not have access to the original PDF papers for the CORA dataset, we can only extract our other feature ideas on the HEP dataset.

[2]Note that by a happy coincidence this allows us to read individual CV iteration results from our boxplots as they correspond directly with Q_1, Q_2, Q_3, and the two outliers (see Section 5.4)

Feature Category	Variations	Models	Data
Baseline	-	Segmentation, header	CORA, CORA app. HEP, CORA + HEP, HEP, HEP app. CORA
		Header	HEP app. 1/3 CORA, HEP app. 2/3 CORA
Dictionaries	1^{st} order, 2^{nd} order, 3^{rd} order	Segmentation, header	HEP, HEP app. CORA
Dicts. + Stops	1^{st} order, 2^{nd} order, 3^{rd} order, stops only	Segmentation, header	HEP, HEP app. CORA
Regularisation	$\sigma^2 = 0$, $\sigma^2 = 10^{-6}$, $\sigma^2 = 10^{-5}$, $\sigma^2 = 10^{-4}$, $\sigma^2 = 10^{-3}$,	Header	HEP
Token Extension	First 5 words, first 10, first 15, first 20	Segmentation	HEP
Block Size	Height, width, height & width, area	Header	HEP
Levenshtein Distance	$T_1 = 0.05$, $T_1 = 0.1$, $T_1 = 0.2$, $T_1 = 0.4$, $T_1 = 0.8$, ($T_1 = 0.1, T_2 = 0.4$), All	Segmentation	HEP
Character Classes	Binary, decimal (round down), decimal (round), decimal (20 point)	Segmentation	HEP

TABLE 5.1: A summary of our experiments, organised by category, models trained for, and data configurations used.

5.3 Comparison with *refextract*

As a first result for GROBID, we compare its reference list classification performance with that of *refextract*, the existing solution for automatic reference extraction at CERN. *refextract* is an example of a *stylistic analysis* tool (see Section 3.2), as it employs regular expressions in a heuristic framework for metadata extraction. As previously mentioned, *refextract* is incomplete and greatly lacking in both breadth and depth of detail. It is capable only of retrieving an article's references[3], and the classification itself is quite simplistic. Since the modelling of reference classes differs between the two, a comparison is difficult to make. Our results will however be at least indicative and we are able to make reasonable comparisons across the most important classes. The dataset for the comparison consists of 60 articles retrieved from the SCOAP[3] online repository[4].

[3] A comparatively easy task; GROBID's citation model usually performs at a significantly higher accuracy than, say, its *header* model.

[4] SCOAP[3] (Sponsoring Consortium for Open Access Publishing in Particle Physics) is an open access digital library hosted at CERN, backed by an international partnership of research institutions.

Unlike *refextract*, GROBID requires two separate models to classify the citations of a given article: the *reference-segmenter* and *citation* models[5]. The *reference-segmenter* model is the simplest model in GROBID's arsenal, and is responsible for segmenting a reference list block into individual references. It does this by modelling two classes, <label>, the delimiting tokens between individual references, and <reference>. Therefore, the accuracy of the citation model is ultimately subject to the accuracy of the reference block inputs supplied it by the *reference-segmenter* above. The results for training and evaluating the *reference-segmenter* on 60 SCOAP^3 papers with an 80–20 split are given in Table 5.2. The results show the *reference-segmenter* to be extremely accurate. In fact, only 5 token misclassifications out of 622 were made for the <label> class, and from a grand total of 12,981.

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 5.2: Token-level evaluation results for reference segmentation.

The most significant difference between the tools is the set of classes modelled. *refextract* attempts only to classify to the minimum detail required for identifying the originating document within INSPIRE-HEP. Therefore, there are no equivalents to GROBID's classes, <volume>, <pages>, and so on. Rather, these parts of references are absorbed into other, higher-level classes, and are indicated by a dash (-) in the results table, Table 5.3. Comparisons can be made, however, on fields, <title>, <author>, <journal>, and <date>. There we see the superiority of GROBID over *refextract*. Note that here the *citation* model was not trained on the evaluation set, and in particular this may explain its dismal performance in precision for the <date> class, and recall for <pubnum>. The dataset instances contained a recurring publication number that was almost uniformly misclassified by GROBID as a date. Notice that this is an example of a domain specificity of HEP papers. Had we trained on these papers, we could expect an improvement. That *refextract* has reported perfect recall is the result of a flaw in our simplistic evaluation, where missing expected classifications were simply assigned to a <null> class, and so *false positives* (FP) were not counted; only *false negatives* (FN). Therefore, the true performance of *refextract* is upper-bounded by the figures in Table 5.3. The table contents are one of the outputs of GROBID's evaluation utilities. For an explanation of the performance metrics, see Section 5.1.1.

[5]Strictly speaking, there is another model, (full) *segmentation*, above the *reference-segmenter*, and so *citation* accuracy depends on this also. But because one focus of our work is to improve this model, we permit this omission.

system	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	-	-	-	-
<issue>	99.93	100	94.63	97.24	-	-	-	-
<pages>	99.75	93.51	99.45	96.39	-	-	-	-
<date>	98.39	57.39	98.31	72.47	98.88	100	37.55	54.6
<pubnum>	98.71	100	12.96	22.95	-	-	-	-
<note>	99.4	43.75	35	38.89	-	-	-	-
<publisher>	99.81	63.46	94.29	75.86	-	-	-	-
<location>	99.81	86.32	91.11	88.65	-	-	-	-
<institution>	99.78	25	25	25	-	-	-	-
<booktitle>	98.7	55.56	41.67	47.62	-	-	-	-
<web>	99.64	51.85	100	68.29	-	-	-	-
<editor>	99.93	100	46.67	63.64	-	-	-	-
<tech>	99.95	83.33	50	62.5	-	-	-	-
(micro average)	99.5	93.63	94.77	94.19	-	-	-	-
(macro average)	99.5	77.92	73.76	71.76	-	-	-	-

TABLE 5.3: Token-level evaluation results for citations for GROBID and *refextract*.

5.4 Results

As specified earlier, we use the micro average of F_1 scores of all model classes to compare model performance, only looking to specific fields when more nuanced comparisons are required. We hereafter state F_1 micro averages unless otherwise specified. Note that Section 5.5 presents the key results and comparisons from this section.

5.4.1 Baseline

Given the datasets available (see Section 4.2), we are able to experiment with combinations of HEP and CORA papers, as well as with subsampling the CORA dataset to find the ideal mixture. After all, in spite of the common wisdom that increasing the amount of training data will improve generalisation and model performance, it is not clear what effects combining different ground truths will have. One may imagine that generalising over a hybrid dataset might learn a model that is misleading when it comes to evaluate on a pure HEP dataset, especially when the CORA *header* set dwarfs our HEP one. The baseline results are given in Table 5.4 and these show the micro average of F_1 scores (see Section 5.1.1) for each of the data configurations. Of most importance are the scenarios involving the HEP training set taken alone, and the HEP training set appending the CORA set, as these evaluate on HEP papers only, and are the configurations used in the

Model	Variation	Data	Mean	Std.
Header	-	HEP	90.19	2.63
		HEP app. CORA	89.54	3.76
		CORA	82.22	2.03
		CORA app. HEP	81.8	2.76
Segmentation	-	HEP	92.98	0.79
		HEP app. CORA	93.58	1.65
		CORA	94.02	2.96
		CORA app. HEP	94.39	2.72

TABLE 5.4: Mean and standard deviation for baseline data configurations.

Model	Variation	Data	Mean	Std.
Header	-	HEP	90.19	2.63
		HEP app. 1/3 CORA	89.15	5.75
		HEP app. 2/3 CORA	**92.1**	**2.43**
		HEP app. CORA	89.54	3.76

TABLE 5.5: Mean and standard deviation for subsampling CORA dataset.

other experiments. Surprisingly, for the *header* model, there is a degradation of performance when the CORA set is appended. We pursue this curiosity in Table 5.5, where we see mixed results for *subsampling* CORA data, appending an extra, randomly chosen third of the 2506 training instances each time. The pure HEP set with no appending of CORA data ($\mu = 90.19, \sigma = 2.63$) yields a model second only to the case where we append 2/3 of the CORA dataset ($\mu = 92.1, \sigma = 2.43$). The case of appending 1/3 CORA experienced a major outlier in the third iteration of cross-validation, where it scored an F_1 micro average of as little as 78.15, performing badly across all classes. This result notwithstanding, it seems there is some value in using a hybrid dataset, yet increasing the proportion of CORA papers ultimately degrades performance. This strongly supports our hypothesis that there a qualitative difference between HEP papers and general papers, but that training a successful model demands an expansion of the HEP dataset. In further sections, we see the same effect when we train on the HEP dataset and HEP appending CORA. For a visualisation of the subsampling results, see Figure B.5 in Appendix B. The successful predictions of our newly introduced collaboration class for the header model (see Section 4.4.1) should be noted also.

In contrast, the *segmentation* model generally benefited from combining datasets, with HEP appending CORA ($\mu = 93.58, \sigma = 1.65$) and CORA appending HEP ($\mu = 94.39, \sigma = 2.72$) respectively improving over HEP alone ($\mu = 92.98, \sigma = 0.79$) and CORA alone ($\mu = 94.02, \sigma = 2.96$). This seems reasonable, as the HEP set in this case is larger than the CORA one, and appending the CORA dataset can assist model performance without overwhelming the HEP characteristics as it does for the *header* model. Figure 5.2 gives a confusion matrix for the *segmentation* model evaluated with the baseline features on the

HEP training set. The main diagonal (in dark blue) shows the majority correctness of the model. Notably, the <body> class (the lines belonging to the main article sections) shows a high true positive (TP) rate (observe the contrast in the <body> row), but also a high number of false positives (observe the confusion in the <body> column), equivalently, the false *negatives* of other classes. Thus, lower precision and higher recall. This is to be expected, as <body> is the dominant class for the *segmentation* model, and training error is most easily minimised with a bias to the dominant class (a similar result may be seen in Figure B.6 in Appendix B for the <abstract> class in the *header* model). The confusion matrix also reveals large amounts of header matter lost to the <body> class. This is of greatest concern, as the extracted data ought to supply the *header* model with its inputs at prediction time. One of our extensions to GROBID (Section 4.4.1) allows us to see which documents these misclassifications are most attributed to.

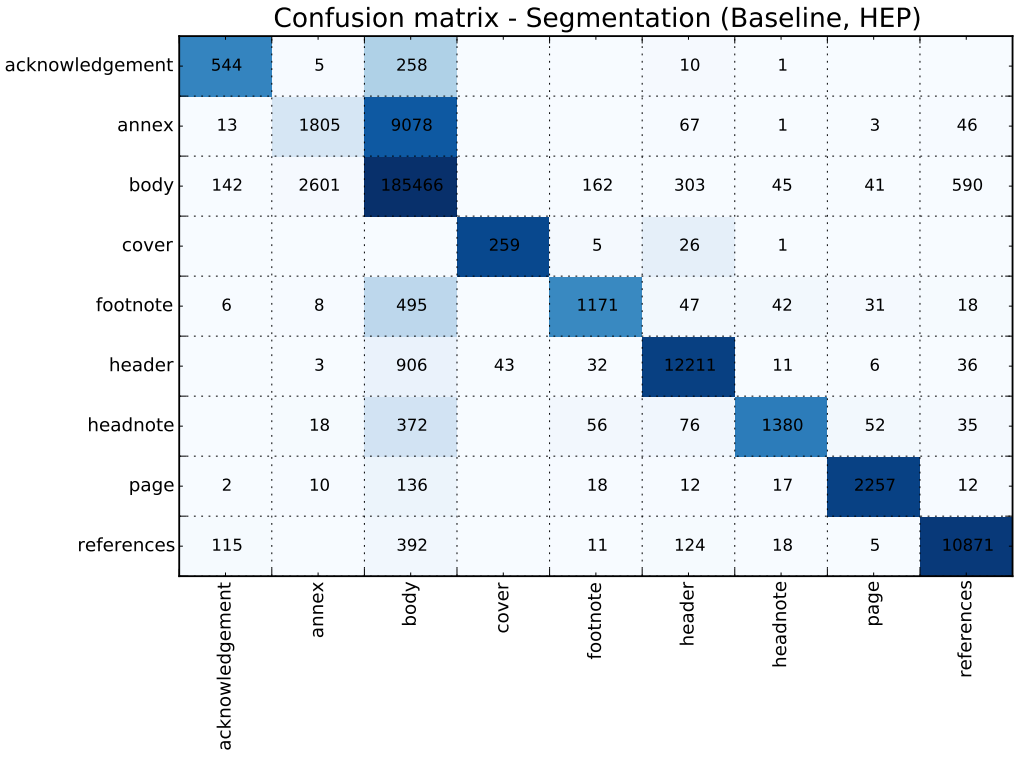


FIGURE 5.2: Confusion matrix for *segmentation* model with baseline features, trained on the pure HEP dataset. Counts are given, as well as a heatmap to indicate the most frequent classifications.

5.4.2 Block Size

The results for our block size features (Table 5.6) were not significant, with only the most marginal differences with respect to the baseline ($\mu = 90.19, \sigma = 2.63$). The category using both height and width features performed worst ($\mu = 88.47, \sigma = 3.1$), showing

Model	Variation	Data	Mean	Std.
Header	*Baseline*	*HEP*	*90.19*	*2.63*
	Height	**HEP**	**90.5**	**2.55**
	Width	HEP	90.46	2.75
	Height & width	HEP	88.47	3.1
	Area	HEP	90.42	2.55

TABLE 5.6: Mean and standard deviation for block size variations.

that combining features does not necessarily improve, and may even degrade, model performance, a finding we see elsewhere also[6]. The results are otherwise quite uniform. We may speculate as to the underwhelming performance of these features by noticing that block size is calculated for every token according to its membership in a contiguous group of tokens. As a result, contiguous tokens share identical information on block sizes, and these features do nothing to *characterise* the tokens individually in the way successful features such as *character classes* (Section 5.4.3) and *dictionaries* (Section 5.4.4) do.

5.4.3 Character Classes

Almost every character class strategy made an improvement over the baseline (with the exception of the simple *binary* variation). A visualisation of the results may be seen in Figure B.3 in Appendix B. Thus, the results support our intuitions about features based on character classes. The variations *binary only* and *decimal only* refer to the removal of the basic baseline features addressing punctuation and related line token characteristics. Interestingly, removing these baseline features *improved* performance, even for the baseline feature set alone. The best micro average result was for the *20-point* variation ($\mu = 93.75, \sigma = 2.56$) compared with the baseline, ($\mu = 92.98, \sigma = 0.79$), corresponding to an 11% reduction in error on the micro average level, as well as a 20% reduction in error (F_1) for <header>, and a 13% reduction in error for <references>, the two most important classes. The *20-point* variation used a finer discretisation than the others. Future work might therefore try further refining the discretisation strategies. However, the runner-up, *decimal only* ($\mu = 93.69, \sigma = 1.99$), gave a 24% reduction for the <header> class, and 21% for <references>. We therefore favour this variation, and short-list it for our further comparisons (see Section 5.5).

[6]Notably the combination of our best features for *segmentation*. Future work might examine the correlation of independent features to see if a methodology for combining features may be derived.

Model	Variation	Data	Mean	Std.
Segmentation	*Baseline*	*HEP*	*92.98*	*0.79*
	Binary	HEP	92.82	1.26
	Binary only	HEP	93.5	2.17
	Decimal (round down)	HEP	93.57	2.2
	Decimal only	HEP	93.61	1.99
	Decimal (round nearest)	HEP	93.37	2.27
	Decimal (20 point)	**HEP**	**93.75**	**2.56**

TABLE 5.7: Mean and standard deviation for character class discretisation strategies.

5.4.4 Dictionaries

Dictionary-based features, as described in Section 4.3.4, make use of domain knowledge to establish a vocabulary for *header* model tokens and give a binary indicator, modelling a token's memberships in a set of five dictionaries (titles, authors, journals, collaborations, and keywords), extracted from INSPIRE-HEP. We varied the degree of context-awareness provided to a token, creating features indicating the dictionary membership of a token's immediate neighbours, that is, the tokens either side of it in the text, in addition to its own membership, as well as second and third degree neighbours. The results for these variations are given in Table 5.8. Though these features were designed with the *header* model in mind, the same experimental variations were run for *segmentation*. The features were expected to be more meaningful for the *header* model, as they give a dimensionality reduction for a full token, mapping it to some combination of dictionaries. In the *segmentation* model, this is done only for the leading word per line, and it is hard to imagine this characterising a line effectively. Predictably, the *header* model benefited significantly more from dictionary-based features, with clear improvements over the baseline: $(\mu = 90.19, \sigma = 2.63)$ for the *header* model trained on HEP, and $(\mu = 92.98, \sigma = 0.79)$ for the *segmentation* model. The most successful variation was the 1^{st} degree for both *header* $(\mu = 90.75, \sigma = 2.5)$ and *segmentation* $(\mu = 93.91, \sigma = 2.07)$ models. When stop words were included (Table 5.9) the benefits were even greater for the *header* model. This reflects the significant varying of stop word frequency between classes discovered in Section C.1 in Appendix C. The most successful variation in this case was the 3^{rd} degree for both *header* $(\mu = 91.35, \sigma = 3.1)$ and *segmentation* $(\mu = 93.58, \sigma = 2.66)$ models. The results for the *header* models are selected for closer examination in Section 5.5. Of further note was the near uniform superiority of the pure HEP data configuration versus that of appending CORA data, reinforcing our findings about subsampling from the baseline results (Section 5.4.1. It is after this experiment batch that we choose to be more focused in the scenarios chosen, concentrating on the pure HEP dataset and the models expected to most benefit from the respective feature experiments.

Model	Variation	Data	Mean	Std.
Header	*Baseline*	*HEP*	*90.19*	*2.63*
	1st order	**HEP**	**90.75**	**2.5**
		HEP app. CORA	87.65	6.49
	2nd order	HEP	90.75	3.01
		HEP app. CORA	88.78	7.46
	3rd order	HEP	87.67	7.55
		HEP app. CORA	89.78	5.99
Segmentation	*Baseline*	*HEP*	*92.98*	*0.79*
	1st order	HEP	93.56	1.79
		HEP app. CORA	**93.91**	**2.07**
	2nd order	HEP	92.16	1.61
		HEP app. CORA	93.37	1.75
	3rd order	HEP	93.11	2.6
		HEP app. CORA	93.58	1.89

TABLE 5.8: Mean and standard deviation for dictionary features.

Model	Variation	Data	Mean	Std.
Header	*Baseline*	*HEP*	*90.19*	*2.63*
	1st order	HEP	91.21	2.03
		HEP app. CORA	88.43	5.92
	2nd order	HEP	88.9	7.25
		HEP app. CORA	89.87	5.03
	3rd order	**HEP**	**91.35**	**3.1**
		HEP app. CORA	89.76	5.99
Segmentation	*Baseline*	*HEP*	*92.98*	*0.79*
	1st order	HEP	93.21	1.13
		HEP app. CORA	93.48	2.04
	2nd order	HEP	93.03	1.78
		HEP app. CORA	93.24	3.49
	3rd order	HEP	93.25	2.77
		HEP app. CORA	**93.58**	**2.66**

TABLE 5.9: Mean and standard deviation for dictionary features combined with stop word features.

5.4.5 Levenshtein Distance

All Levenshtein distance feature thresholding strategies performed better than the baseline, and the scenarios combining multiple thresholds, ($T_1 = 0.1, T_2 = 0.4$) and *All*, show a clear superiority over the others. A visualisation of the results can be seen in Figure B.4 in Appendix B. The strategy, *All*, which used all of the listed thresholds to make a $5 - point$ categorical variable based on Levenshtein distance gave the best performance, ($\mu = 94.06, \sigma = 1.81$), compared with the baseline, ($\mu = 92.98, \sigma = 0.79$), corresponding to a 15% reduction in error on the micro average level, as well as a 20% reduction in error for <references>, and a 19% reduction in error for the <header>. These results therefore support our intuitions of the effectiveness of this feature category. Such as it

is, from the laborious data acquisition process (Section 4.2), we observed the frequency of the misclassification of delimiting token classes, <page> and <headnote>, as part of the <body> or <reference> sections, raising the false positive (FP) rate for these classes, thus lowering their precision and F_1 scores alike. It is therefore likely that the Levenshtein distance features aided these transitional corner cases in the document segmentation. The most successful Levenshtein distance variety is compared in more detail in Section 5.5.

Model	Variation	Data	Mean	Std.
Segmentation	*Baseline*	*HEP*	*92.98*	*0.79*
	$T_1 = 0.1$	HEP	93.51	1.37
	$T_1 = 0.2$	HEP	92.95	1.19
	$T_1 = 0.4$	HEP	93.44	1.28
	$T_1 = 0.8$	HEP	93.07	1.07
	$T_1 = 0.1, T_2 = 0.4$	HEP	93.78	1.83
	All	**HEP**	**94.06**	**1.81**

TABLE 5.10: Mean and standard deviation for Levenshtein distance thresholding variations.

5.4.6 Regularisation

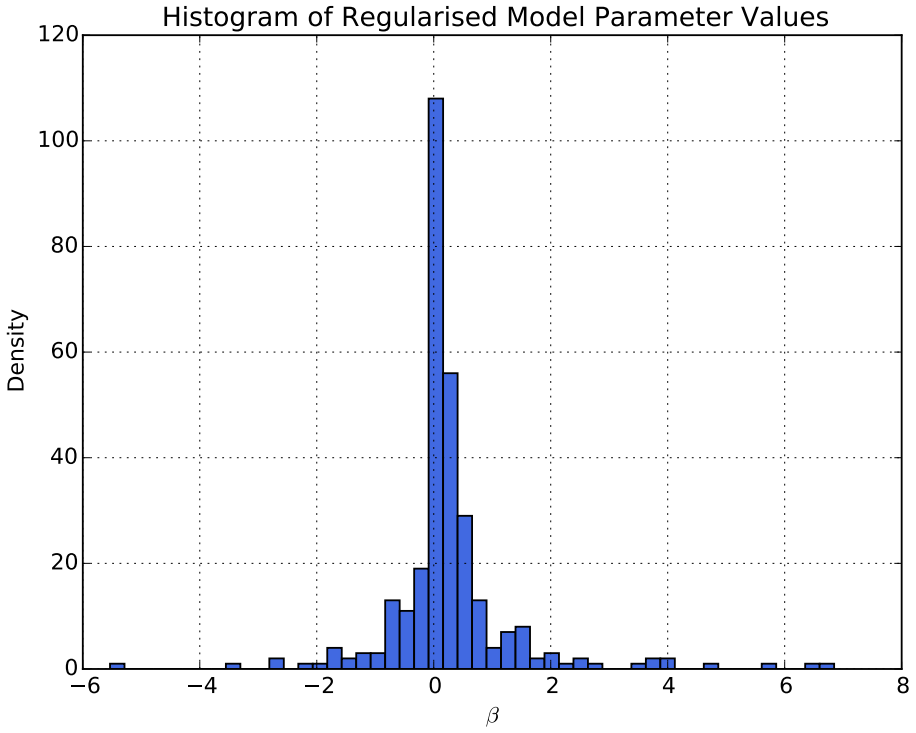


FIGURE 5.3: Distribution of model parameters with l_2 regularisation.

Our experiments in tuning the variance of the l_2 regularisation confirmed the earlier findings of (Peng and McCallum [2004]). There is little observable difference in the performance of different choices of variance parameter, which we give in Table 5.11. Performance is marginally worse at the extremities, namely where no penalty is imposed, and for the greatest penalty, $\sigma^2 = 10^{-3}$. Naturally, as a penalty is increased toward infinity, the model parameters go to 0, and performance degrades maximally. Figure 5.3 shows an empirical Normal distribution for a normalised model. That regularisation has little effect implies that it is difficult to overfit the model, and that the high dimensionality of the model perhaps provides in-built regularisation. Training with another algorithm supporting a different regularisation technique (such as l_1) could form the basis of future work.

Model	Variation	Data	Mean	Std.
Header	*Baseline*	*HEP*	*90.19*	*2.63*
	$\sigma^2 = 0$	HEP	90.4	2.58
	$\sigma^2 = 10^{-6}$	HEP	90.68	3.78
	$\sigma^2 = 10^{-5}$	HEP	90.64	3.78
	$\sigma^2 = 10^{-4}$	HEP	90.66	3.69
	$\sigma^2 = 10^{-3}$	HEP	90.44	2.77

TABLE 5.11: Mean and standard deviation for tuning the variance of the l_2 variance parameter.

5.4.7 Token Extensions

The results for token extensions (Table 5.12) are quite modest. No variation significantly exceeds the baseline result with the same data configuration ($\mu = 92.98, \sigma = 0.79$). Of note is the low variance across the variations compared with other feature categories. The result supports our earlier intuitions that though it may be helpful to establish a vocabulary that can differentiate between sections, the result of modelling each word in a line token individually is a feature space that is too diffuse to learn useful indicators, given that encountering the same combination of words occurs only rarely. Line tokens are therefore better characterised, for example, at the character- rather than word-level, such as by character class features (Section 5.4.3).

Model	Variation	Data	Mean	Std.
Segmentation	*Baseline*	*HEP*	*92.98*	*0.79*
	First 5	**HEP**	**93.38**	**1.6**
	First 10	HEP	92.8	1.08
	First 15	HEP	93.12	1.5
	First 20	HEP	93.29	1.75

TABLE 5.12: Mean and standard deviation for token extension variations.

5.5 Key Results

In this section we visualise and discuss the most interesting and significant comparisons drawn from the results in Section 5.4. We consider each model in turn, beginning with the *header* model.

5.5.1 Header Model

The complexity of the *header* model, modelling 17 classes (including our new <collaboration> class), made finding improvements difficult. Nevertheless, the model benefited from the five binary dictionary-based features, and further benefited from the additional stop word feature, functioning as a $6th$ dictionary. A comparison with the baseline on the HEP dataset is visualised in Figure 5.4, showing the micro average F_1 scores for each model. Dictionaries alone ($\mu = 90.75, \sigma = 2.5$) reduced error over baseline ($\mu = 90.19, \sigma = 2.63$) by 6%, and dictionaries with stop words ± 3 (that is, third degree contextual awareness), by 12%. The latter, our best result for the *header* model, corresponds with a 16% error reduction for <address>, 19% for <author>, 54% for <collaboration>, 14% for <date>, and 25% for <keywords>, unchanged performance for <abstract>, <affiliation>, and <title>, and a slight degradation of 8% for <pubnum>.

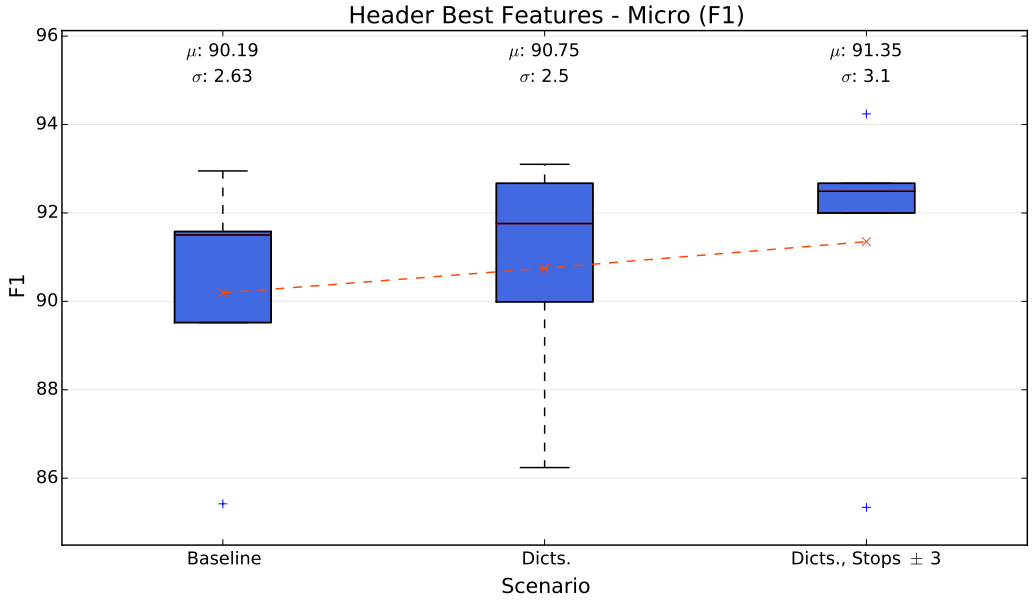


FIGURE 5.4: Comparison of baseline, dictionary, and stop word features for overall *header* model performance.

5.5.2 Segmentation Model

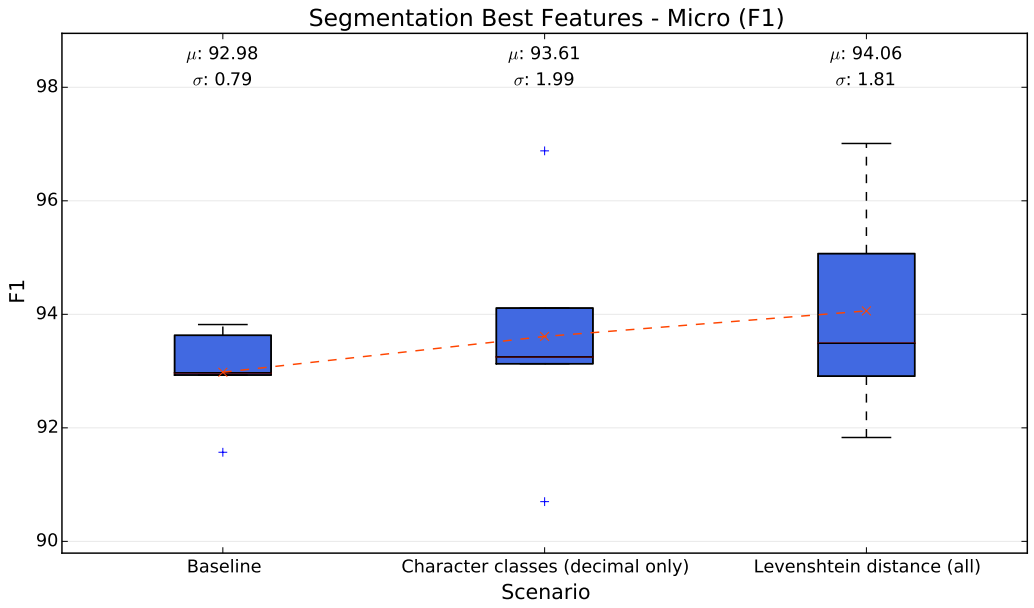


FIGURE 5.5: Comparison of baseline, Levenshtein distance and character class features for overall *segmentation* model performance.

For the *segmentation* model, two feature categories offered significant improvements over the baseline. These categories were those of character classes (CC) and Levenshtein distance (LD), and their best variations were respectively the decimal discretisation with related baseline features removed (decimal only), and the Levenshtein distance feature using all thresholds to create a quinternary categorical variable. Figure 5.5 shows a boxplot comparing the micro average of F_1 scores of these two variations with the baseline, evaluated on the pure HEP dataset. The CC variation gives an overall error reduction of 9%, and the LD, 15%. However, when we look at the most important fields, <header> (Figure 5.6) and <references> (Figure 5.7), we see that though again both models outperform the baseline, now CC leads with a 24% error reduction for the <header> class, with 19% for LD, and a 21% error reduction for <references> versus 20% for LD. The character class model is therefore excelling in the cases it was explicitly designed to improve.

The confusion matrix for character classes (decimal only) on HEP papers in Figure 5.8 shows a dramatic improvement over the baseline (Figure B.6). Of particular interest is the increase across the main diagonal, that is true positives (TP) in every class but one, <annex>, where it decreases from 1805 to 1573. Also of interest is the huge reduction (906 to 312) of false negatives for the <header> class to the <body>, explaining where the improvements are being made. Note that this was accompanied by a smaller increase

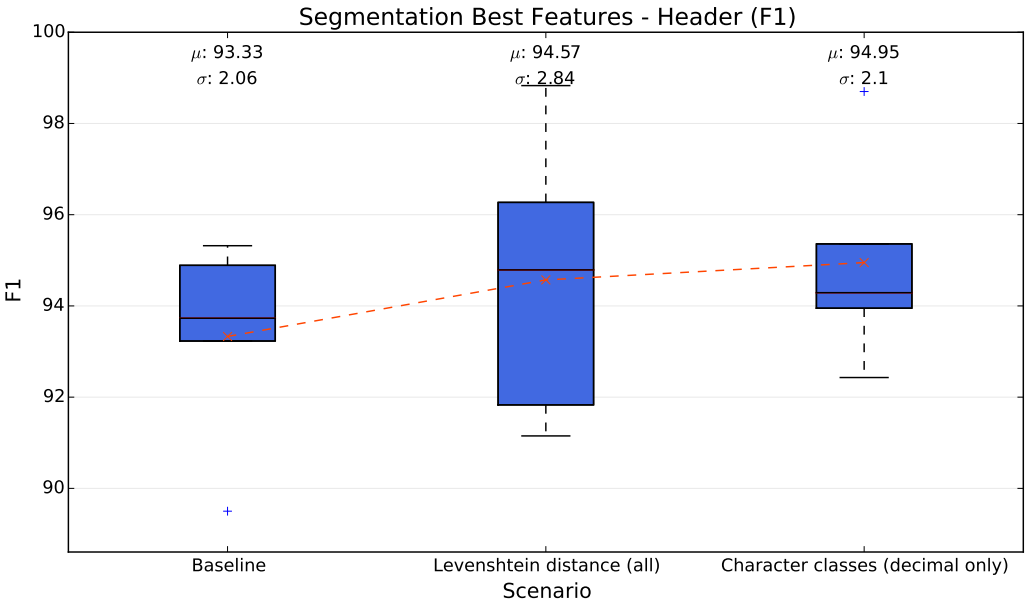


FIGURE 5.6: Comparison of baseline, Levenshtein distance and character class features for the header extraction.

of false positives from <body> to <header> (303 to 530). Another remarkable improvement is the reduction (from 115 to 0) of false positives for the <acknowledgement> class for an expected <references> class. In general, classifications of the <acknowledgement> class improved considerably. Also, <header> lines previously lost to the <cover> page were reduced from 43 to 1. We may conclude, finally, that though the overall performance

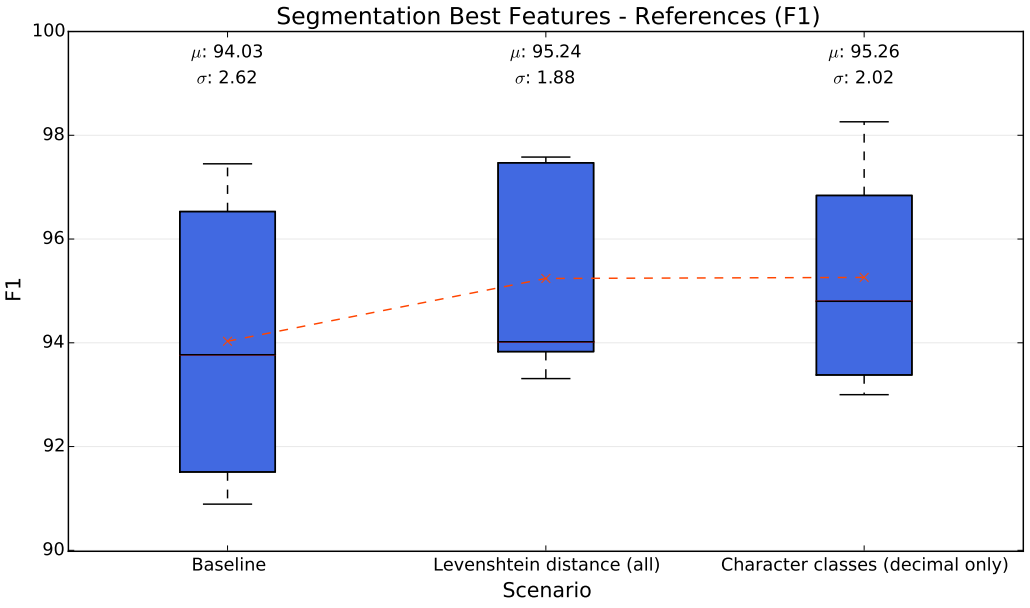


FIGURE 5.7: Comparison of baseline, Levenshtein distance and character class features for the reference extraction.

of CC was slightly lower than the LD, we should prefer the CC feature variation due to its outstanding performance in key classes, <header> and <references>. In combination, the two features did not yield any additional gains.

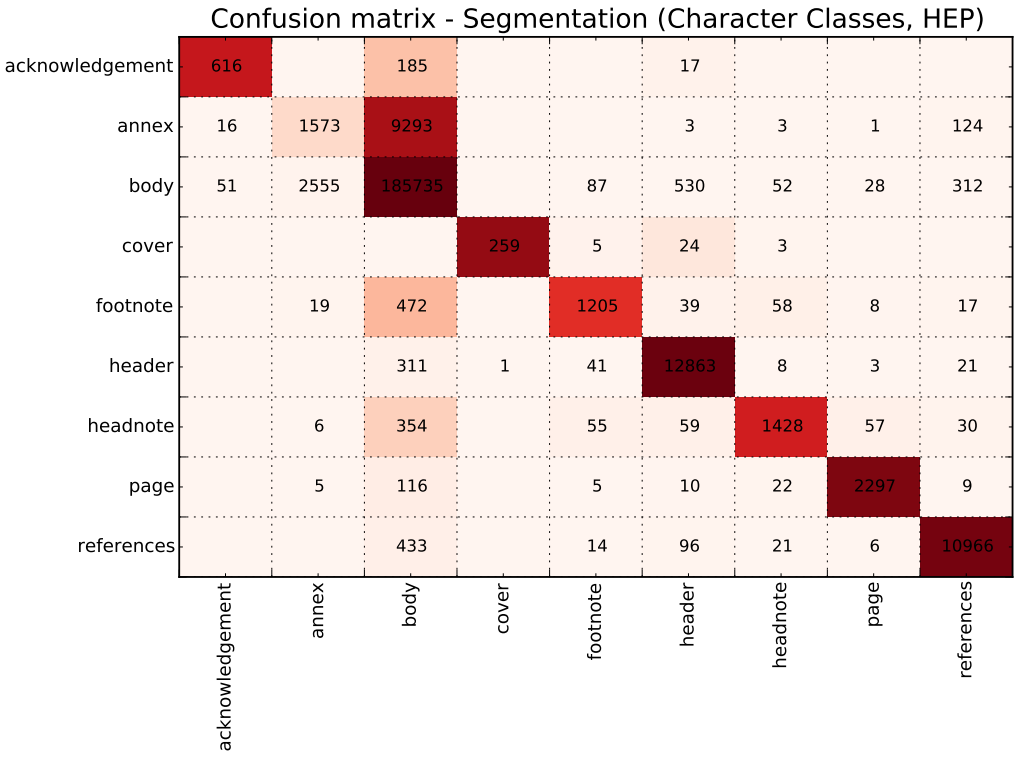


FIGURE 5.8: Confusion matrix for segmentation model with character class features, trained on the pure HEP dataset. Counts are given, as well as a heatmap to indicate the most frequent classifications.

Chapter 6

Conclusions

6.1 Summary

Our overarching hypothesis is that HEP papers are a special case and have qualitative differences to conventional scientific papers. This thesis demonstrated this in a number of ways, first through a qualitative inspection of HEP papers in Chapter 4, then quantitatively, through our experimentation in Chapter 5, such as through subsampling, which defied the conventional wisdom of expanding training data, indicating a problem in generalising over both HEP and CORA papers. Subsequent to this, our approach was:

1. to train models on a custom HEP training set, and;
2. to engineer specialised features conducive to better generalisation on HEP papers.

Compiling a custom training set required much tedious manual work, and furthermore had to be done for both *header* and *segmentation* models. One positive outcome of the data acquisition, however, was that we more than doubled the existing training set for the *segmentation* model. There may also be value in the 60 SCOAP^3 papers that we configured for training in our successful comparison of GROBID with *refextract* (Section 5.3). This also increases the *reference-segmenter* model training set substantially. This new data may be reused and incorporated into the open source GROBID project.

Our feature engineering ideas were based on domain-specific features, such as INSPIRE-HEP-derived dictionary features for the *header* model, or by addressing the unique structural characteristics of HEP papers in the *segmentation* model. Of particular success were the features based on the character class composition for line tokens in the *segmentation* model. This gave our best overall results, proving to be highly effective indicators for differentiating between token classes. Levenshtein distance between lines, realised

as a single categorical variable, also proved to be effective. These latter features further demonstrated the value in more elaborate and systematic approaches to feature engineering, whereas the baseline feature set is rather simplistic and heuristic. Our results suggest that the best features are those giving a *dimensionality reduction* to tokens, that is, those features offering a mapping to a lower-dimensional variable. Block shape features were unsuccessful because they did not characterise the individual token well, rather addressing rich information of token groups (blocks). In contrast, for the *header* model our best results were dictionary-based features, which map word tokens to a lower-dimensional group. This was best realised in our novel *stop word* dictionary, which achieved an error reduction in F1 of 12%, as well as large error reductions for several key classes. We may note additionally that applying conditional random fields to *line* tokens, as in the *segmentation* model, offers far greater scope for creativity and novelty than the *header* model. Our most significant overall improvement were our character class features, which achieved error reductions to key classes, <header> and <references>, of 24% and 21% respectively. Our Levenshtein distance features are exceptional in that they model differences between successive lines of text. This appears to have resolved many of the transitional misclassifications observed in our data acquisition (Section 4.2).

We may note finally that GROBID, including our extensions and custom data, will shortly be in use within INSPIRE-HEP.

6.2 Future Work

We hereby state some ideas for extending the work that we have completed so far:

1. An obvious starting point for improving upon our work is to continue expanding the HEP training sets for each of the models. A major finding of our work is that long-term improvement of the models will require the manual creation of HEP training data, and cannot be fully compensated by appending general CORA data.

2. Another extension to GROBID would be to model collaborations in the *citation* model, as we achieved for the *header* model (Section 4.4.1).

3. Not much came of our regularisation experiments other than to reinforce the findings in the literature. It would be interesting to try other forms of regularisation, such as l_1. This requires an optimisation algorithm that is not gradient-based, but Wapiti does offer such alternatives to L-BFGS.

4. It would be interesting to combine some of our feature ideas. With the exception of combining dictionaries with stop word features, we did not find any useful feature pairs. Our strongest pair of features for the *segmentation* model did not

yield additional gains in combination. Investigations could therefore be made into the interaction of feature categories, and a methodology derived for how best to combine them.

5. Our approach could be made easier with further enhancements to GROBID's evaluation utilities. For example, it would be useful to see exactly which tokens are being misclassified when misclassifications occur. We went some way towards this in 4.4.1 where we tracked the k documents most contributing to classification error.

Appendix A

Algorithms

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 $\mathbf{y}*$

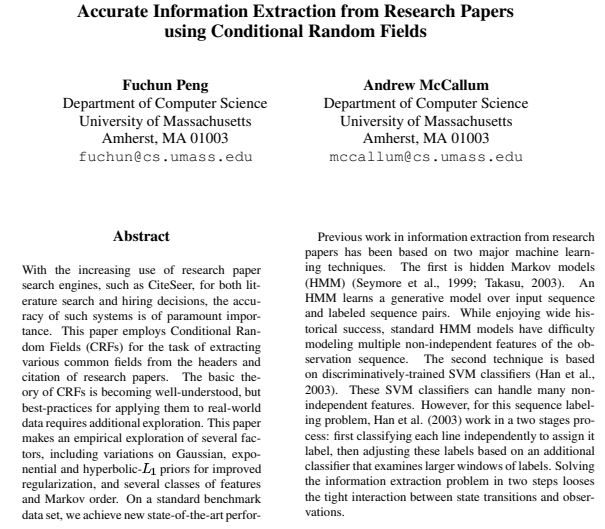
Algorithm 1: The Viterbi algorithm ($\mathcal{O}(T|S|^2)$) for computing the most likely hidden sequence for a given observation sequence of an HMM.

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

Appendix B

Figures



Accurate Information Extraction from Research Papers using Conditional Random Fields

Fuchun Peng
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
fuchun@cs.umass.edu

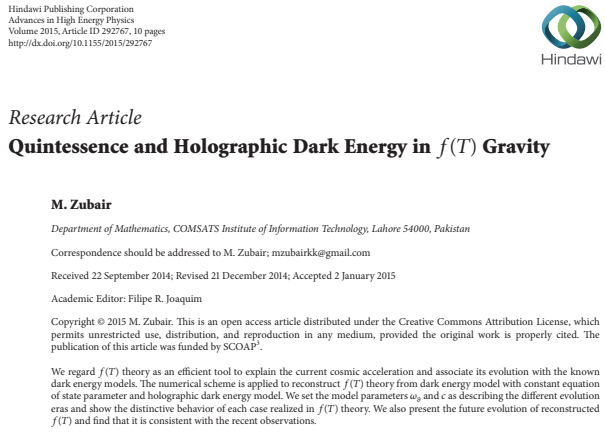
Andrew McCallum
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
mccallum@cs.umass.edu

Abstract

With the increasing use of research paper search engines, such as CiteSeer, for both literature search and hiring decisions, the accuracy of such systems is of paramount importance. This paper employs Conditional Random Fields (CRFs) for the task of extracting various common fields from the headers and citation of research papers. The basic theory of CRFs is becoming well-understood, but best-practices for applying them to real-world data requires additional exploration. This paper makes an empirical exploration of several factors, including variations on Gaussian, exponential and hyperbolic-L_1 priors for improved regularization, and several classes of features and Markov order. On a standard benchmark data set, we achieve new state-of-the-art perfor-

Previous work in information extraction from research papers has been based on two major machine learning techniques. The first is hidden Markov models (HMM) (Seymore et al., 1999; Takasu, 2003). An HMM learns a generative model over input sequence and labeled sequence pairs. While enjoying wide historical success, standard HMM models have difficulty modeling multiple non-independent features of the observation sequence. The second technique is based on discriminatively-trained SVM classifiers (Han et al., 2003). These SVM classifiers can handle many non-independent features. However, for this sequence labeling problem, Han et al. (2003) work in a two stages process: first classifying each line independently to assign it label, then adjusting these labels based on an additional classifier that examines larger windows of labels. Solving the information extraction problem in two steps looses the tight interaction between state transitions and observations.

FIGURE B.1: The header section of a scientific paper. Excerpt from Peng and McCallum [2004]



Hindawi Publishing Corporation
Advances in High Energy Physics
Volume 2015, Article ID 292767, 10 pages
http://dx.doi.org/10.1155/2015/292767

Hindawi

Research Article

Quintessence and Holographic Dark Energy in $f(T)$ Gravity

M. Zubair

Department of Mathematics, COMSATS Institute of Information Technology, Lahore 54000, Pakistan

Correspondence should be addressed to M. Zubair; mzubairkk@gmail.com

Received 22 September 2014; Revised 21 December 2014; Accepted 2 January 2015

Academic Editor: Filipe R. Joaquim

We regard $f(T)$ theory as an efficient tool to explain the current cosmic acceleration and associate its evolution with the known dark energy models. The numerical scheme is applied to reconstruct $f(T)$ theory from dark energy model with constant equation of state parameter and holographic dark energy model. We set the model parameters ω_0 and c as describing the different evolution eras and show the distinctive behavior of each case realized in $f(T)$ theory. We also present the future evolution of reconstructed $f(T)$ and find that it is consistent with the recent observations.

FIGURE B.2: The header section of a HEP paper. Excerpt from Zubair [2015]

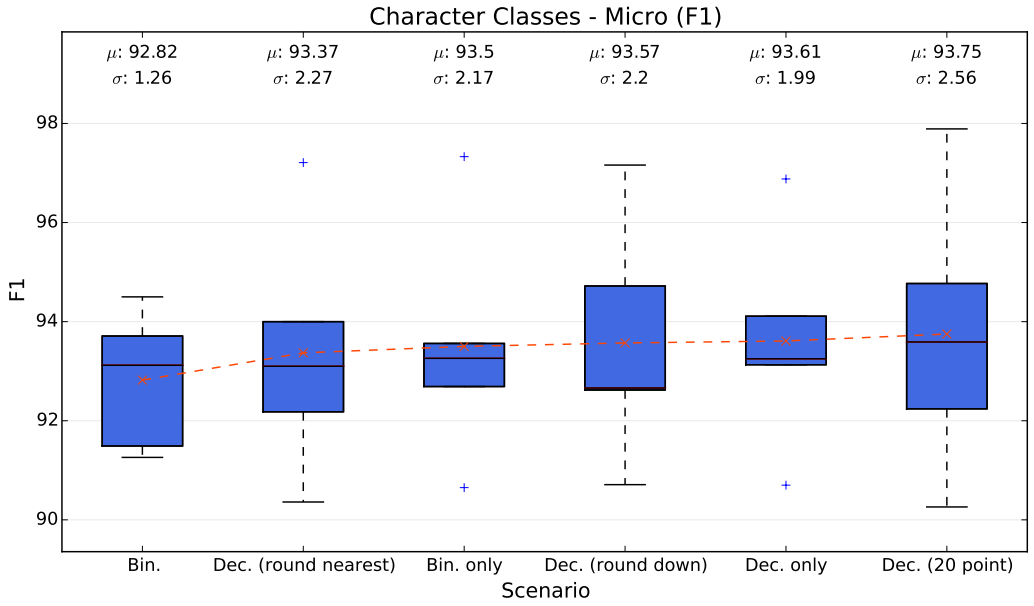


FIGURE B.3: Comparison of different character class feature discretisation strategies.

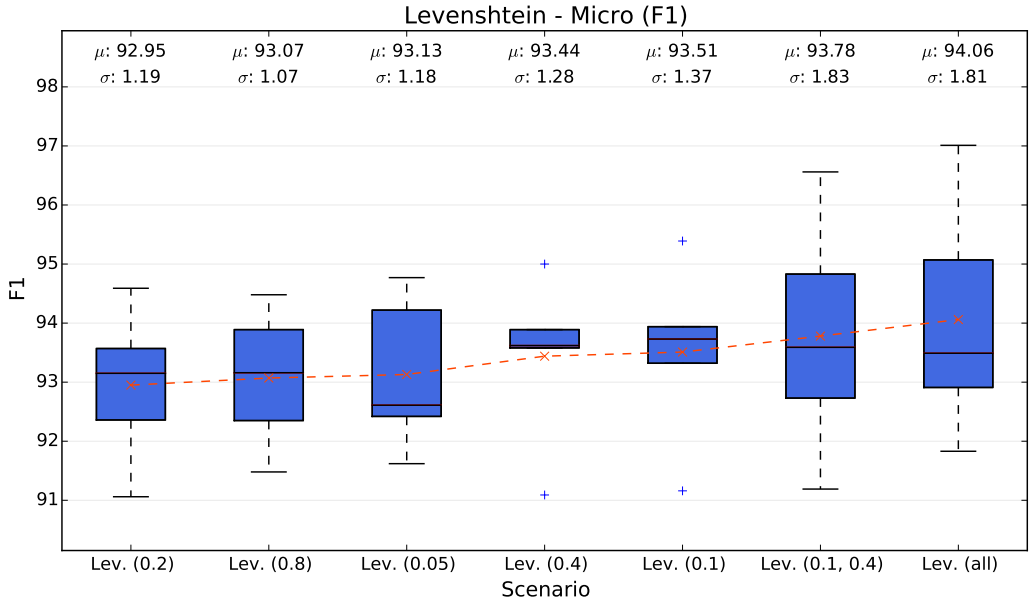


FIGURE B.4: Comparison of different levenshtein distance feature thresholding strategies.

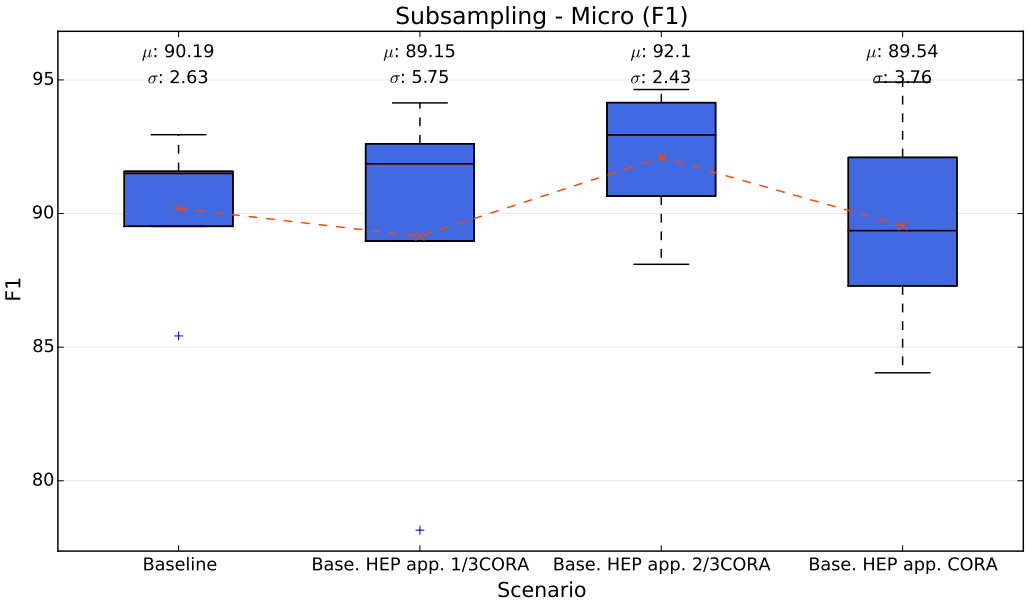


FIGURE B.5: Comparison of different data configurations subsampling the CORA dataset.

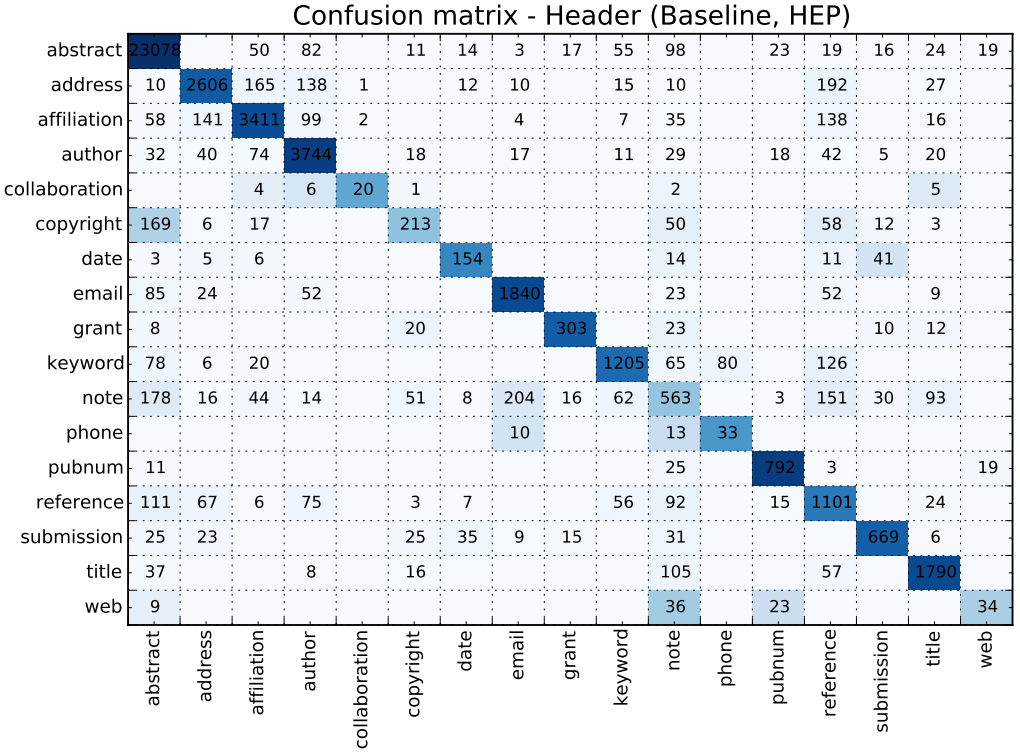


FIGURE B.6: Confusion matrix for *header* model, baseline features, trained on HEP data..

Appendix C

Statistical Tests

C.1 Stop Word Frequency

To substantiate our claim that stop word frequency varies according to header section, we computed the frequency of stops words in abstract, author list, and title sections for 20 HEP papers (`anova_data.py`). Plotting these frequencies (Figure C.1) showed a drastic difference.

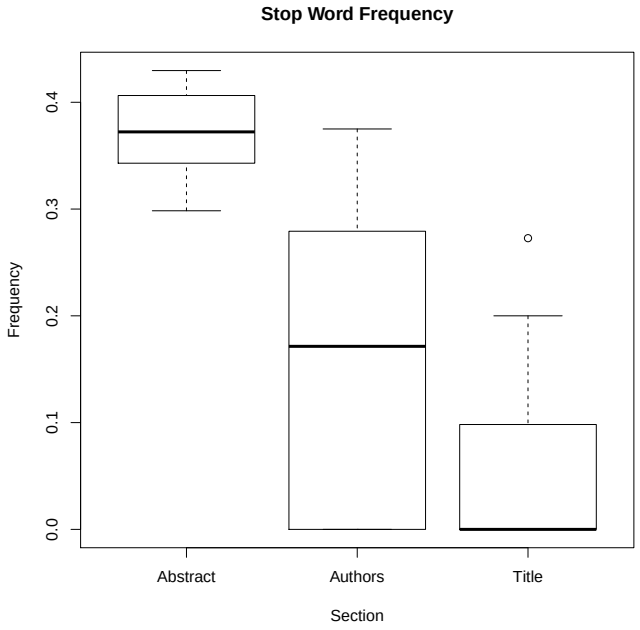


FIGURE C.1: Box plots of stop word frequency according to header section.

To confirm the significance of this result, we first performed an ANOVA in R (Figure C.2) on the frequency data. The test reported a p-value of (< 0.01), permitting us to reject the null hypothesis that the means are equal ($H_0 : \mu_1 = \mu_2 = \cdots = \mu_k$), thereby confirming

```
            Df Sum Sq Mean Sq F value  Pr(>F)
Section      2 1.0685  0.5342   57.28 2.3e-14 ***
Residuals   57 0.5317  0.0093
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

FIGURE C.2: ANOVA showed the average stop word frequency of header sections varies significantly.

```
        Pairwise comparisons using t tests with pooled SD

data:  stops and sections

        Abstract Authors
Authors 1.5e-08  -
Title   1.6e-14  0.0016

P value adjustment method: bonferroni
```

FIGURE C.3: Pairwise t-tests showed significance for each comparison.

the statistical significance of the varying means. We further performed pairwise t tests (Figure C.3) to show the significance of the result for each class, with each p-value of (< 0.01).

Bibliography

Andrew McCallum, Dayne Freitag, and Fernando CN Pereira. Maximum Entropy Markov Models for Information Extraction and Segmentation. In *ICML*, volume 17, pages 591–598, 2000a.

Eamonn Maguire, Philippe Rocca-Serra, Susanna-Assunta Sansone, Jim Davies, and Min Chen. Taxonomy-based glyph design–with a case study on visualizing workflows of biological experiments. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2603–2612, 2012.

Roel Aaij, B Adeva, M Adinolfi, A Affolder, Z Ajaltouni, S Akar, J Albrecht, F Alessio, M Alexander, S Ali, et al. Identification of beauty and charm quark jets at LHCb. *arXiv preprint arXiv:1504.07670*, 2015.

Fuchun Peng and Andrew McCallum. Accurate information extraction from research papers using conditional random fields. In *HLT-NAACL04*, pages 329–336, 2004.

M Zubair. Quintessence and Holographic Dark Energy in Gravity. *Advances in High Energy Physics*, 2015, 2015.

Anne Gentil-Beccot, Annette Holtkamp, Salvatore Mele, Heath B O'Connell, and Travis C Brooks. Information resources in High-Energy Physics: Surveying the present landscape and charting the future course. *Journal of the American Society for Information Science and Technology*, 60(1):150–160, 2009.

Alan Souza, Viviane Moreira, and Carlos Heuser. ARCTIC: metadata extraction from scientific papers in pdf using two-layer CRF. In *Proceedings of the 2014 ACM symposium on Document engineering*, pages 121–130. ACM, 2014.

Lawrence R Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

Eugene Charniak, Curtis Hendrickson, Neil Jacobson, and Mike Perkowitz. Equations for part-of-speech tagging. In *AAAI*, pages 784–789, 1993.

Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state Markov chains. *The annals of mathematical statistics*, pages 1554–1563, 1966.

Andrew J Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269, 1967.

John Binder, Kevin Murphy, and Stuart Russell. Space-efficient inference in dynamic probabilistic networks. *Bclr*, 1:t1, 1997.

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

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.

Hervé Déjean and Jean-Luc Meunier. A system for converting PDF documents into structured XML format. In *Document Analysis Systems VII*, pages 129–140. Springer, 2006.

Mario Lipinski, Kevin Yao, Corinna Breitinger, Joeran Beel, and Bela Gipp. Evaluation of header metadata extraction approaches and tools for scientific pdf documents. In *Proceedings of the 13th ACM/IEEE-CS joint conference on Digital libraries*, pages 385–386. ACM, 2013.

Patrice Lopez. GROBID: Combining automatic bibliographic data recognition and term extraction for scholarship publications. In *Research and Advanced Technology for Digital Libraries*, pages 473–474. Springer, 2009.

Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000b.