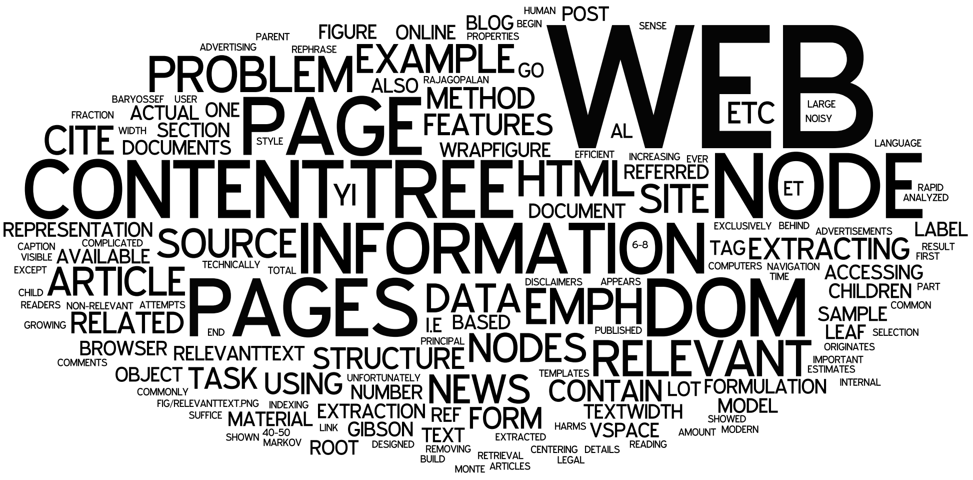
CHALMERS



Large-Scale Content Extraction from Heterogeneous Sources

Master's Thesis in
Engineering Mathematics and
Computational Science

DANIEL LANGKILDE

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2015

Large-Scale Content Extraction from Heterogeneous Sources

Examiner: CHRISTOS DIMITRAKAKIS

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2015

Abstract

In this thesis report we describe a novel approach to large scale content extraction from heterogenous web sources. This task is a very important step in a range of web crawling, indexing and data mining tasks. The described approach makes calculations on the Document Object Model (DOM) in order to uncover which nodes contain relevant content, and which do not. We set out with the hypothesis that the DOM tree can be modeled as a hidden Markov tree model where the hidden state of each node indicates if its relevant content or not. Using Gibbs samling we uncover the hidden states of the node, and show that competative performance can be achieved using this approach.

Acknowledgments

I would like to thank my academic supervisor Christos Dimitrakakis who introduced me to the theory of Gibbs sampling, which lays the foundation for the extraction method proposed in this thesis. Without his patient supervision my attempt to implement the suggested method would never had succeeded. I would also like to express my deep gratitude to Staffan Truvé who introduced me to the task of information retrieval and posed the question that this thesis attempts to address. I would also like to thank Mr. Truvé for the opportunity to apply my work at Recorded Future. Many thanks also to my peers at Chalmers and Recorded Future for their kind advice and to my girlfriend for her patience. Finally, thanks to our puppy Alice, who kept me company during the first month of this project.

Daniel Langkilde, Gothenburg, June 10th, 2015

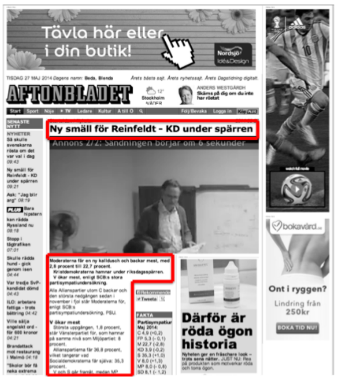
Contents

1

Introduction

With the rapid growth of published material available online there is an ever increasing need of efficient content extraction methods. Unfortunately web pages are designed for human readers, accessing web pages through browsers, rather than for computers reading the data automatically. The task is further complicated by the large amount of non-relevant information on a web page. Modern web pages contain a multitude of information besides the actual content, such as navigation menus, user comments, advertising, related documents, legal disclaimers etc. Estimates by Gibson et al. [7] showed that in 2005 40-50% of material on the web was of a template nature, and at the time growing as a fraction of total content by 6-8% per year.

We define content extraction as the task of extracting relevant content, for example a news article or blog post, from a web source specified by a URL address. The content extracted should be relevant in the sense that it belongs to the main article or blog post on the web page. For example, when viewing a news article on an online news site, a lot of information that is not part of the actual article is visible, such as related news, snapshots from other articles, advertisements etc. An example of this is shown in figure 1.1.



Ny smäll för Reinfeldt - KD under spärren

Moderaterna får en ny kalldusch och backar mest, med 2,8 procent till 22,7 procent.
Kristdemokraterna hamnar under riksdagsspärren.
V ökar mest, enligt SCB:s stora partisympatiundersökning.

Alla Allianspartier utom C backar och den största nedgången sedan i november i fjol står Moderaterna för, enligt SCB:s partisympatiundersökning, PSU.

V ökar mest

Största uppgången, 1,8 procent, står Vänsterpartiet för, som hamnar på samma nivå som Miljöpartiet: 8 procent.

Allianspartierna får 36,8 procent, vilket tangerar vad Socialdemokraterna får själva: 35,3 procent.

V och S går framåt, medan MP

Partisympatier Maj 2014:
C 4,9 (+0,2)
FP 5,3 (- 0,1)
M 22,7 (-2,8)
KD 3,9 (-0,2)
S 35,3 (+1,0)
V 8,0 (+1,3)
MP 8,0 (- 0,8)
SD 8,1 (- 1,2)

NYHETER
Så skulle svenskarna rösta om det var val i dag 09:43
Ny smäll för Reinfeldt - KD under spärren 09:21
Ask: "Jag blir arg" 08:19
PLUS Bara hipstern kan rädda Ryssland nu 08:18
Stopp i tågtrafiken 07:01
Skulle rädda hund - gick genom isen 04:44
Var tredje SvP-kandidat dömd 04:43
ILO: arbetare fattiga - trots bättring 04:42
Ville sälja engelskt ord - för 600 kronor 04:21
Brandattack mot restaurang i Malmö 04:18
"Skolor bör få neka extrema

Därför är röda ögon historia
Nyheten ger en fräschare look – trots sena nätter. JUST NU: Rea på produkten som motverkar röda och torra ögon.

Figure 1.1: Example of relevant content marked with red.

1.1 Related Work

The problem of accurately extracting relevant content from web pages is important for several different tasks such as web crawling, indexing, data mining, readability tools etc. If too much redundant information is included when mining a web page there is risk of topic drift when classifying and categorizing the page. Such redundant information also increases the index size of search engines. Little research on content extraction exists from before the late 1990's, but since the founding of search engines such as Lycos (1994), AltaVista (1995) and Google (1998) the amount of research available has exploded.

Most commonly the structure of the web page is analyzed based on its Document Object Model (DOM). The DOM is a convention for representing objects in HTML documents. One formulation of the content extraction task is that of extracting templates, i.e. the common content that appears in the same form on multiple pages from a site. This formulation was first introduced by Bar-Yossef and Rajagopalan in 2002 [5] as a way to reduce redundant, repetitive information when classifying and categorizing sites. They proposed a technique based on a segmentation of the DOM tree and selection of nodes using properties such as the number of links in the node.

Yi et al. [12] and Yi [13] phrase the problem as the task of removing noisy features, i.e. features that contain information which harms mining. They sample pages from the same site in order to build a Site Style Tree (SST). Baluja [4] employs decision tree learning and entropy reduction for template detection at DOM level. The method suggested by Baluja relies on information from the rendered page, such as the bounding box for HTML elements, along with the DOM. Others have employed methods using visual segmentation based on appearance [6] and exploiting term entropies [8].

The redundant information is sometimes referred to as boilerplate material. Kohlschütter [10] performs a quantitative segmentation of text based on token density revealing two fuzzy classes of text covering content and boilerplate respectively. Based on this conclusion Kohlschütter et al. [11] have devised a method for removal of boilerplate text based on shallow text features such as number of words, average word length, number of full stops etc. The algorithm is provided as an open-source package called Boilerpipe [9]. Boilerpipe employs a number of rule based regular expressions in addition to the shallow text analysis. Currently Boilerpipe appears to be the only open-source content extraction package maintained regularly.

Apple's browser Safari has a built in function for cleaning up articles and presenting them in a more reader friendly way, known as the Safari Reader. Although Safari Reader is a proprietary algorithm it is known to be built upon the Readability project by Arc90 [3]. The Readability algorithm is essentially a collection of rules and heuristics. According to a comparison by Kohlschütter [2] the Boilerpipe ArticleExtractor outperforms the Safari Reader, see figure 1.2. The comparision is performed on the L3S-GN1 dataset and measures recall, precision, and the F-measure as the harmonic mean of the two. Recall is defined as the probability that actual content was labeled content, while precision is the probability that non-content is labeled content. The measures are defined at token-level, which means that a node with little text will matter less if it is mislabeled.

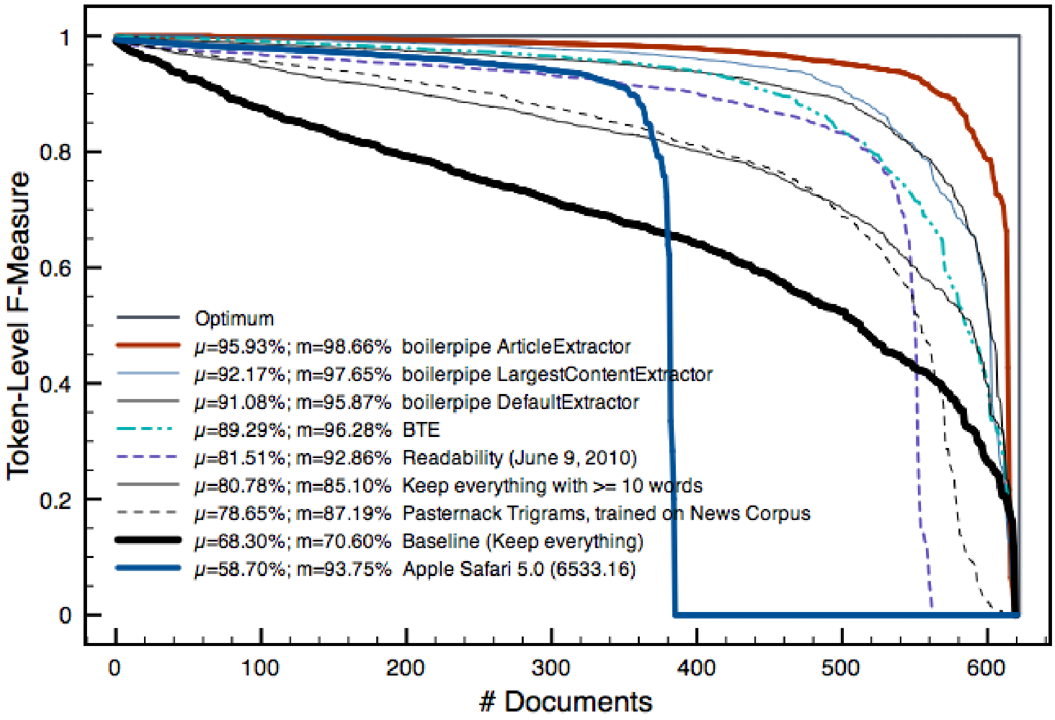


Figure 1.2: Comparision: Boilerpipe (red), Safari (blue), baseline - keep everything (black). Measures recall and precision presented using the F_1-measure, i.e. harmonic mean of the two. Recall is defined as the probability that actual content was labeled content, while precision is the probability that non-content is labeled content. The measures are defined at token-level, which means that a node with few words will matter less if it is mislabeled. μ is the average recall while m is the average precision of the different algorithms.

1.2 Context and Goals

The content extraction task we attempt to solve is part of a product provided by Recorded Future [1] called the Temporal Analytics Engine^{TM}. The purpose of the Temporal Analytics Engine^{TM} is to provide a forecasting and analysis tool by scanning web sources. As shown in figure 1.3 the content extraction step provides input for further analysis. The goal of the complete analysis is to extract information about named entities and their planned activities, referred to as events. The quality of the content extraction step greatly impacts the accuracy of the end result. Recorded Future currently use Boilerpipe as their primary content extraction tool for full-page harvesting. Although Boilerpipe performs well on certain sources it often breaks or returns erroneous information. Also, it does not provide sufficient detail in its classification of content. When a URL is passed to the content extraction method the ideal would be for it to return information from the source categorized as either of the following categories:

- **Headline**
- **Fulltext** - The text body of an article, blogpost or similar
- **Supplemental** - General meta data about the document such as author, publishing time, publishing place, related material, fact boxes, tables, image captions etc.
- **Comments** - User comments on the article
- **Not content** - Everything else

We will henceforth refer to the above categories as the available *types*. The category a specific node belongs to is referred to as its type.

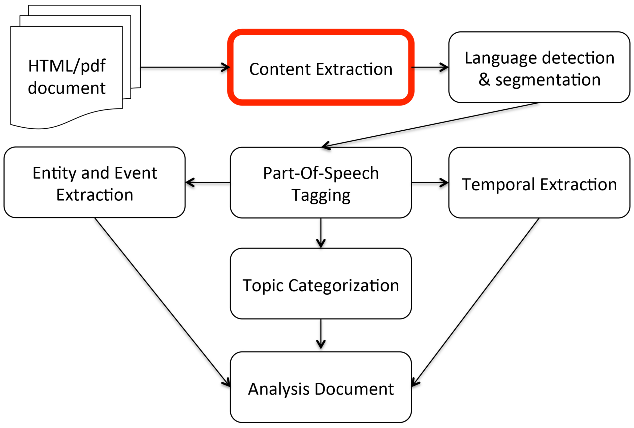


Figure 1.3: Context of content extraction task

2

Bakground

After studying the publications mentioned in section 1.1 we conclude that the most common approaches to content extraction take into account some combination of the DOM, the properties of the rendering of the page, and text features. Before detailing our conclusions regarding which information to take into account we will review the concept of the Document Object Model (DOM) and introduce those of hidden Markov tree models and Gibbs sampling.

2.1 Representation of Web pages

Web pages as they are perceived by humans accessing them using a browser are the result of their source HyperText Markup Language (HTML) code and Cascading Style Sheets (CSS) being rendered by the browser. The HTML source provides a structured representation of the web page that forms the basis of the Document Object Model (DOM). The DOM is a tree data structure consisting of all the HTML elements found in the source.

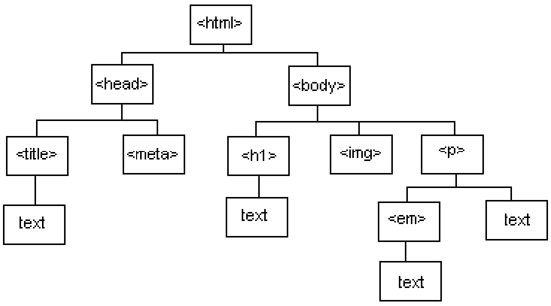


Figure 2.1: Example of a small portion of a DOM tree data structure

The DOM tree of a web page originates in a *root node*, the <html> tag, and branches down, each node labeled with their respective HTML tag. A small sample of such a tree can be seen in figure 2.1. Each node in the DOM tree, except the root, has one unique *parent node*. Each node can also have a number of *child nodes*. Any node with children is referred to as an *internal node*. Any node that does not have children is referred to as a *leaf node*. Typically, but not exclusively, leaf nodes are those that contain text. The connection between two nodes is referred to as a *link* or *edge*. The *height* of a node is the longest downward path to a leaf from that node. The height of the root is the height of the tree. The depth of a node is the length of the path from the node to its root. The *distance* between two nodes is measured by shortest number of edges that need to be traversed in order to get from one node to the other.

2.1.1 Rendering of Web Pages

A web browser renders a web page using a layout engine such as Blink (Chrome, Opera), Trident (Internet Explorer), Gecko (Firefox) or WebKit (Safari). The layout engine takes the marked up content (such as HTML, XML, etc.) and the formatting information (such as CSS, XSL, etc.) and displays the formatted content on the screen. We will make use of the properties of the rendering of web pages. We do this by assigning each node of the DOM the coordinates of its bounding box in the rendering using an open-source package called CSSBox. CSSBox is essentially a light-weight browser written in Java. We will not go into the details of web page rendering as it is beyond the scope of this report, but we note that CSSBox does not support JavaScript rendering. Possible solutions to this issue are discussed in section 6 .

2.2 Markov Models

This section is to be regarded as a very basic introduction to Markov models in general, and particularly hidden Markov tree models. A **Markov process** is a stochastic process that satisfies the Markov property. The Markov property for discrete time chains can be described as

$$Pr(X_{t+1} = s_j \mid X_0 = s_k, \ldots, X_t = s_i) = Pr(X_{t+1} = s_j \mid X_t = s_i). \tag{2.1}$$

In a general sense this means that a process satisfies the Markov property if one can make predictions for the next step based only on the current state of the process.

A **Markov chain** refers to a sequence of random variables $(X_0, \ldots, X_n)$ generated by a Markov process. In a regular Markov chain the state of each variable is directly visible to the observer and therefore the transition probabilities are the only parameters. In a ***hidden*** **Markov model** (HMM) the state of the variables are not directly visible, but instead the observer sees some type of output dependent on the hidden state. Each state has a probability distribution over the possible output types.

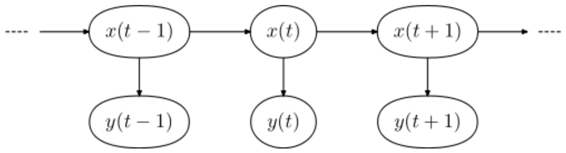


Figure 2.2: An illustration of a general hidden Markov model. $x(t)$ is the state of the hidden variable at time t, while $y(t)$ is the state of the visible output at time t.

Assume we have a general HMM where the random variable $x(t)$ is the hidden state at time t and the random variable $y(t)$ is the observable output at time t, as shown in figure 2.2. Further assume that the state space of the hidden variable is a finite, discrete set such that $x(t) \in \{x_1, \ldots, x_N\}$ is modeled as a categorical distribution. The parameters of a hidden Markov model are of two types, *transition* and *emission* probabilities. The transition probabilities govern the way the hidden state at time t is chosen given the hidden state at time $t-1$, while the emission probabilities govern the distribution of the observed variable at a particular time given the state of the hidden variable at that time.

Next we will formulate a description of a basic hidden Markov model extended to a tree data structure in a Bayesian setting. However, before we can do that, we need to introduce some theory concerning Bayesian inference.

2.2.1 Bayesian Inference

There are two dominant interpretations of probability theory, the Frequentist and the Bayesian. Bayesian reasoning starts with some prior assumption about the probability of an event, and then updates that belief as more observations are made. In order to give a formal description of Bayesian inference we need to define a set of variables.

- $\mathbf{x}$ - a general data point
- θ - the parameter of the data point's distribution, i.e. $x \sim Pr(x \mid \theta)$
- α - the hyperparameter of the parameter, i.e. $\theta \sim Pr(\theta \mid \alpha)$
- $\mathbf{X}$ - a set of n observed data points, i.e. $x_1, \ldots, x_n$
- $\tilde{x}$ - a new data point whose distribution is to be predicted

The **prior distribution** is defined by the parameters of the data points distribution before any data is observed, i.e $Pr(\theta \mid \alpha)$. One interpretation is that α describes our hypothesis about the behaviour of the data. The posterior distribution is the distribution after data has been observed. The **posterior distribution** is calculated using Bayes' rule:

$$Pr(\theta \mid \mathbf{X}, \alpha) = \frac{Pr(\mathbf{X} \mid \theta, \alpha) Pr(\theta \mid \alpha)}{Pr(\mathbf{X} \mid \alpha)} \propto Pr(\mathbf{X} \mid \theta) Pr(\theta \mid \alpha) \tag{2.2}$$

$Pr(\mathbf{X} \mid \theta)$ is called the **likelihood**. One way to understand this is when its viewed as a function of the parameter, $L(\theta; \mathbf{X}) = Pr(\mathbf{X} \mid \theta)$, i.e. the likelihood of seeing $\mathbf{X}$ given θ. The **marginal likelihood** is the distribution of the observed data marginalized over the parameters, i.e.

$$Pr(\mathbf{X} \mid \alpha) = \int_\theta Pr(\mathbf{X} \mid \theta) Pr(\theta \mid \alpha) d\theta \tag{2.3}$$

Bayes' rule is readily derived from the basic principles of conditional probability, i.e. that

$$Pr(\mathbf{X} \mid \theta) = Pr(\mathbf{X} \cap \theta) Pr(\theta) \tag{2.4}$$

where

$$Pr(\mathbf{X} \cap \theta) = \frac{Pr(\mathbf{X})Pr(\mathbf{X} \cap \theta)}{Pr(\mathbf{X})} \tag{2.5}$$

and

$$Pr(\theta \mid \mathbf{X}) = \frac{Pr(\theta \cap \mathbf{X})}{Pr(\mathbf{X})} \tag{2.6}$$

If the posterior distribution $Pr(\theta \mid \mathbf{X},\alpha)$ is in the same family of distributions as the prior distribution $Pr(\theta \mid \alpha)$ then the prior and posterior distributions are called **conjugate distributions**, and the prior is called a **conjugate prior** for the likelihood function. A family of distributions that is conjugate to itself is called self-conjugate. The form of the conjugate prior can generally be determined by inspection of the probability density function of a distribution.

To illustrate, lets look at a simple coin tossing example. For a succession of n coin tosses the probability of getting k heads will be given by the binomial distribution as

$$k \mid n,\theta \sim \text{Binomial}(\theta, n) \tag{2.7}$$

where θ is an unknown variable to be learned.

If we let $\chi^{k,n}$ denote the set of all successions of n throws that contain exactly k heads, then the binomial likelihood can be derived as

$$Pr(k \mid \theta) = \sum_{\mathbf{x} \in \chi^{k,n}} Pr(\mathbf{x} \mid \theta, n) = \sum_{\mathbf{x} \in \chi^{k,n}} \theta^k (1-\theta)^{n-k} = \binom{n}{k} \theta^k (1-\theta)^{n-k} \tag{2.8}$$

Assume that we want to estimate the probability of success for the coin. A common choice of prior for binomial distributions is the beta distribution. It will eventually be clear why, so bear with me. Using this we get that the prior distribution of θ is

$$Pr(\theta \mid \alpha, \beta) \propto \theta^{\alpha-1}(1-\theta)^{\beta-1} \tag{2.9}$$

Since this density is required to integrate to 1 we may write:

$$Pr(\theta \mid \alpha, \beta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{\int_0^1 u^{\alpha-1}(1-u)^{\beta-1} du} = \frac{1}{B(\alpha,\beta)} \theta^{\alpha-1}(1-\theta)^{\beta-1} \tag{2.10}$$

where $B(\alpha, \beta)$ is the beta distribution. Eq (2.10) exploits the fact that $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$. It will now be clear why the beta distribution is used as a prior for binomial data. Our posterior distribution over θ is given by:

$$\begin{aligned} Pr(\theta \mid n,k,\alpha,\beta) &= \frac{Pr(k \mid n,\theta)P(\theta \mid n,\alpha,\beta)}{Pr(k \mid n,\alpha,\beta)} \\ &\propto Pr(k \mid n,\theta)P(\theta \mid n,\alpha,\beta) \\ &= Pr(k \mid n,\theta)P(\theta \mid \alpha,\beta) \\ &= \binom{n}{k} \theta^k (1-\theta)^{n-k} \times \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \theta^{\alpha-1}(1-\theta)^{\beta-1} \\ &\propto \theta^k (1-\theta)^{n-k} \times \theta^{\alpha-1}(1-\theta)^{\beta-1} \\ &= \theta^{k+\alpha-1}(1-\theta)^{n-k+\beta-1} \end{aligned} \tag{2.11}$$

This is exactly the same function as in (2.9). That is, our posterior is also a beta distribution. More generally put, the posterior is proportional to the prior times the likelihood function. The conclusion is that if

$$\begin{aligned} k \mid n,\theta &\sim \text{Binomial}(\theta, n) \\ \theta \mid \alpha,\beta &\sim \text{Beta}(\alpha, \beta) \end{aligned} \tag{2.12}$$

then

$$\theta \mid k,n,\alpha,\beta \sim \text{Beta}(\alpha + k,\beta + n - k) \tag{2.13}$$

Using our newfound knowledge of Bayesian inference we now return to the description of a hidden Markov tree model in a Bayesian setting.

2.2.2 Hidden Markov Tree Model

We will begin our description by extending the type of hidden Markov model introduced in section 2.2 to the type of tree data structures described in section 2.1. The idea behind hidden Markov tree models is that every node of the tree has a hidden and an observable variable, labeled Y and X respectively in figure 2.3. Similar to a discrete time hidden Markov model, such as the one depicted in figure 2.2, where the hidden state $x(t+1)$ depends on $x(t)$, we assume that the hidden state of each node in the tree depends on the state of its observation and the hidden state of its parent. This means that Y_2 depends on X_2 and Y_1 in figure 2.3.

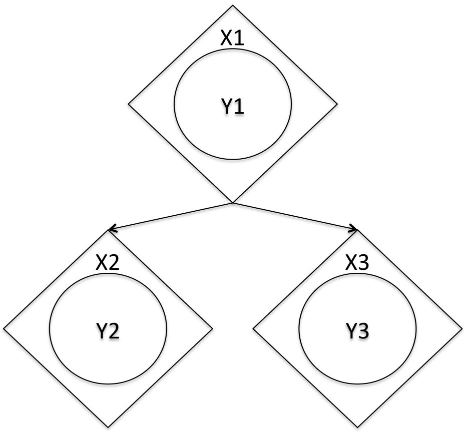


Figure 2.3: Illustration of nodes in a tree data structure with hidden states (Y) and observable variables (X).

Lets imagine a tree with n nodes, the hidden state of node k labeled Y_k. The vector describing all hidden states simultaneously is denoted $\mathbf{Y} = \{Y_1, \ldots ,Y_n\}$. Assume the hidden state is from a finite, discrete set such that $Y_k = Y^n \in \{Y^1, \ldots ,Y^m\}$. The observable variable for node k is labeled X_k. Assume that there is some fixed, known parameter θ governing transmission and emission probabilities. If we know $\mathbf{X} = \{X_1, \ldots , X_n\}$, we can calculate the probability distribution of the hidden state of a particular node k over all possible hidden states as

$$Pr(Y_k = Y^i \mid X_k, \theta, \alpha) = \frac{Pr(X_k \mid Y^i, \theta, \alpha) Pr(Y^i \mid \theta, \alpha)}{\sum_{j=1}^{m} Pr(X,Y^j \mid \theta, \alpha)} \tag{2.14}$$

such that

$$\sum_{i=1}^{m} Pr(Y_k = Y^i \mid X_k, \theta, \alpha) = 1 \tag{2.15}$$

If we instead of having a fixed θ only know the distribution of it, we can compute the probability distribution of the hidden state using the fact that

$$Pr(\mathbf{Y} \mid \mathbf{X}, \alpha) = \int_\theta Pr(\mathbf{Y} \mid \mathbf{X}, \theta, \alpha) dPr(\theta \mid \alpha) \tag{2.16}$$

where $dP(\theta \mid \alpha) = Pr(\theta \mid \alpha)$ and α is the hyperparameter of θ. Finally, assume we know all $(\mathbf{X},\mathbf{Y}) = \{(X_1,Y_1), \ldots, (X_n,Y_n)\}$ but not θ. In this case we can estimate the probability distribution of θ as

$$Pr(\theta \mid \mathbf{X}, \mathbf{Y}, \alpha) = \frac{Pr(\mathbf{X},\mathbf{Y} \mid \theta, \alpha) Pr(\theta \mid \alpha)}{Pr(\mathbf{X},\mathbf{Y} \mid \alpha)} = \frac{Pr(\mathbf{X} \mid \mathbf{Y}, \theta, \alpha) Pr(\mathbf{Y} \mid \theta, \alpha) Pr(\theta \mid \alpha)}{P(\mathbf{X},\mathbf{Y} \mid \alpha)} \tag{2.17}$$

If we train a model, i.e. estimate θ, then we can proceed to determine the probability that a node k has hidden state Y_k using

$$Pr(Y_k \mid X_k, \mathbf{X}_{\setminus k}, \mathbf{Y}_{\setminus k}, \alpha) = \int_\theta P(Y_k \mid X_k, \theta, \alpha) dP(\theta \mid \mathbf{X}_{\setminus k}, \mathbf{Y}_{\setminus k}, \alpha) \tag{2.18}$$

You may notice that this means that we condition the probability distribution of the hidden state for node k on the hidden state of all other nodes. In order to solve equation (2.18) we need the results from equation (2.17). Eq. (2.18) is not computationally tractable, i.e. it cannot be computed in closed form or using explicit numerical methods. There are a variety of different techniques that can be utilized instead, such as the forward-backward algorithm. Rather than using the forward-backward algorithm will make use of something called Gibbs sampling. Gibbs sampling is the final concept we need to introduce before describing the suggested content extraction algorithm.

2.3 Gibbs Sampling

Gibbs sampling is a form of Markov chain Monte Carlo (MCMC) method. Generally, the point of MCMC methods is that they make it possible to avoid computing the marginal likelihood explicitly. The idea behind Gibbs sampling is that given a multivariate distribution it is simpler to sample from a conditional distribution than to integrate over a joint distribution. The Gibbs sampling algorithm generates an instance from the distribution of each variable in the joint distribution in turn, conditional on the current values of the other variables. It can be shown that the sequence of samples constitutes a Markov chain, and the stationary distribution of that Markov chain is just the sought-after joint distribution.

Putting it in a more mathematical way, lets say we have a hidden Markov tree with n nodes each labeled $N_1 \dots N_n$. The probability of a particular node N_k begin of particular type Y_k is given by Eq. (2.18). Our goal is to obtain a sample $\mathbb{Y} = (Y_1 \dots Y_n)$ from the joint distribution function $Pr(Y_1, \dots Y_k)$. For reasons that will be clear in the outline of the algorithm we denote the ith sample as $\mathbf{Y}^i = (Y_1^i \dots Y_n^i)$.

Outline of Gibbs Sampling Algorithm

The implementation of a Gibbs sampler for hidden Markov tree proceeds as follows:

- **Step 0: Random initial state** - Assign each node of the tree data structure a random hidden state.
- **Step 1: Sample the conditional distribution** - For every iteration i, for each node N_k^i sample the conditional distribution $Pr(Y_k^i \mid Y_1^i, \dots, Y_{k-1}^i, Y_{k+1}^{i-1}, \dots, Y_n^{i-1})$. Put another way we sample the probability distribution for the type of node N_k at time i conditioned on the hidden state of all other nodes using their most recently sampled hidden states. After sampling we update the hidden state of node N_k.
- **Step 2: Update our posteriors** - After we have update the hidden states of all nodes we temporarily add the resulting tree data to our calculation of posteriors. These are then replaced in the next iteration, i.e. the data set does not grow.
- **Step 3: Repeat step 1-2 until convergence.** Its complicated to define convergence but usually the best way to determine if the sampler has converged is to run two samplers in parallel. After some initial burn in time the results of both samplers are compared. When they are similar for a long enough time convergence is determined to be reached. The Gibbs sampler only has one deterministic convergence state which means that both samplers, although stochastic in nature, will converge in the same state. In our implementation we have set the number of iterations to 30, averaging the result over the last 10 iterations, rather than comparing the state of two converging chains.

3

Method

Armed with the concepts introduced in chapter 2 we now return to the task of designing our content extraction method. Technically content extraction in this context is equal to classification of the nodes of a DOM tree. The content extraction method might as well be called a DOM tree node classifier. In the field of machine learning classification is considered a form of supervised learning, i.e. it requires a training set of labeled examples. Its easy enough to create annotated data by labeling nodes of DOM trees by hand. When designing a classification algorithm it is important to have a good model describing the properties and behavior of the input data. Based on the model we can evaluate each input and assign it the correct label.

3.1 Data Set

Through out the design and testing process we will rely on an annotated data set created by Kohlschutter et. al. [**?**] called L3S-GN1. The data set originally consisted of 621 articles from 408 different sources collected during the first half of 2008. Since its creation some of the articles have disappeared from the web, and currently 599 articles are accessible. For each article in the data set there is the source URL along with two files; the original HTML file and an annotated HTML file. In the annotated files each node is labeled as one of the following types:

1. Headline
2. Full text
3. Supplemental
4. Related content
5. Comments
6. Not content

The text corpus in the dataset follows Zipf's law and is of such size and variety that it is considered sufficient for evaluation.

3.2 Model of Input Data

We notice in section 1.1 that three sources of features are popular; the DOM, the properties of the rendering of the page and text features. Inspired by a common technique in the field of natural language processing called *part-of-speech tagging* we hypothesize that the problem can be viewed as a sequence labeling problem, i.e. that the context of the node in the DOM tree matters when determining its type. The most common statistical models in use for sequence labeling make the assumption that the sequence satisfies the Markov property (remember Eq. (2.1)).

Based on this reasoning we hypothesize that the DOM can be modeled as a hidden Markov tree model where the hidden state of each node is the type of that node (i.e. if its headline, fulltext, supplemental etc.) and the observable variable is a vector composed of features of the node in question. The features are based on information from the DOM tree, the rendering and the text in the node. We will now develop a more formal, mathematical description of such a model specifying the features we will make use of.

First we address the question of which features of the input data to consider. Assume a tree of n nodes, $\{N_1, \ldots ,N_n\}$, each with one of m possible types. Lets define the set of all nodes with type k as $\mathbb{N}_k = \{N_k^1, \ldots ,N_k^i\}$ such that the nodes of the entire tree form the set $\mathbb{C} = \{\mathbb{N}_1 \cup \cdots \cup \mathbb{N}_k\}$. Each subset of $\mathbb{C}$ is pairwise disjoint since each node must have one and only one type. The best choice of features would be such that given a random node N^j we maximize the probability of placing it in the correct set $\mathbb{N}_k$.

The following features have been evaluated as candidates:

1. **HTML tag** - fig. 3.1
2. **Number of children** - fig. 3.2
3. **Type of children** - fig. 3.3
4. **HTML tag of children** - fig. 3.4
5. **Number of siblings** - fig. 3.5
6. **Type of siblings** - fig. 3.6
7. **HTML tag of siblings** - fig. 3.7
8. **Type of parent** - fig. 3.8
9. **HTML tag of parent** - fig. 3.9
10. **Number of words in node** - fig. 3.10
11. **Number of full stops** - fig. **??**
12. **Node offset from top of page rendering** - fig. 3.11
13. **Node offset from left edge of page rendering** - fig. 3.12
14. **Node width in rendering** - fig. 3.13
15. **Node height in rendering** - fig. 3.14

Before we detail our selection of features to include we will give a few examples of how a feature is used in the model. Let us begin with an example of how we can use the number of children of a node as a feature. Figure 3.2 shows the distribution of the number of children for nodes labeled headline, fulltext and supplemental respectively. As we can see each of the content types have similar propensity to have children, while nodes labeled *Not content* appear to have a higher probability of having children.

For this example we assume that the number of children is the only feature we take into account in our model. In that case the hidden state of a node is given as $Y_k = Y^n \in \{\text{headline, fulltext, supplemental}\}$ while the observable variable X_k is the number of children. We want to estimate the emission probability connecting the observable variable to the hidden state. By inspection of fig. 3.2 we see that the number of children of node k, C_k, for each type could be described by a geometric distribution. This obviously isn't a perfect match, but sufficient to capture the difference, and suitable for a demonstration. Described mathematically we can say that

$$C_k \sim \text{Geometric}(p(Y_k)) \tag{3.1}$$

Using Bayesian inference we can estimate the probability distribution of the parameter for the geometric distribution of each node type. We do this using conjugate priors. The conjugate prior of the geometric distribution is the beta distribution and its hyperparameters are α and β such that

$$p(Y_k) \sim \text{Beta}(\alpha_{Y_k} + n, \beta_{Y_k} + \sum_{i=1}^{n} x_i) \tag{3.2}$$

Using an uninformed prior we then simply count the number of children for each node of each type, entering the values into Eq. (3.2). To calculate the probability that a node N_k is of a specific type Y_k given its number of children C_k we use the estimate in equation (3.2) and plug that into

$$Pr(Y_k \mid C_k) = (1-p)^{C_k - 1} p \tag{3.3}$$

Next we look at the type of parent as a feature. The possible hidden and observable variable of node k are now both the set of all occurring node types. We are now trying to estimate the transition probability from one hidden state to the next. The transition probability takes the form of a multinomial distribution. A multinomial distribution is defined by a set of event outcomes and their respective probability. For n independent trials, each of which results in one of the possible event outcomes, the multinomial distribution gives the probability of any particular combination of numbers of times each event outcome occurs. Put mathematically

$$Pr(Y_k \mid Y_k^{\text{parent}}) \sim \text{Multinomial}(q(Y_k^{\text{parent}}) \tag{3.4}$$

The conjugate prior of the multinomial distribution is the Dirichlet distribution such that

$$q(Y_k^{\text{parent}}) \sim \text{Dirichlet}(\gamma + \sum_{i=1}^{n} x_i) \tag{3.5}$$

where the $\gamma \in \mathbb{R}^n$ is a vector describing our prior knowledge of the number of occurrences of each possible outcome. Ultimately the probability of a node N_k being of a specific type Y_k given that its parent is Y_k^{parent} is calculated using

$$Pr(Y_k \mid Y_k^{\text{parent}}) = \frac{n!}{Y_1! \cdots Y_k!} q_1^{Y_1} \cdots q_k^{Y_k} \tag{3.6}$$

Following this same treatment we can use any feature of the data as long as it has a suitable conjugate pair of distributions for priors and posteriors.

In figure 3.1 - 3.14 we have computed the distribution of each of the parameters presented earlier for the dataset at hand. Several methods are available to evaluate the best set of features to select. We have chosen choose features by inspecting the graphs, rather than implement a clustering algorithm, but we suggest this as a future improvement in section 6. It is our informal experience that selecting good features is very time consuming for industry applications. Generally you want to leverage your intuition about which features matter, while at the same time check that your training data covers the feature space in a meaningful way. If you engineer features that are not sufficiently represented in the training data, they are useless. Since we only wish to prove the concept of this method we have chosen to include all features at this point. The graphs are included to help the reader appreciate the value of the features from visual inspection.

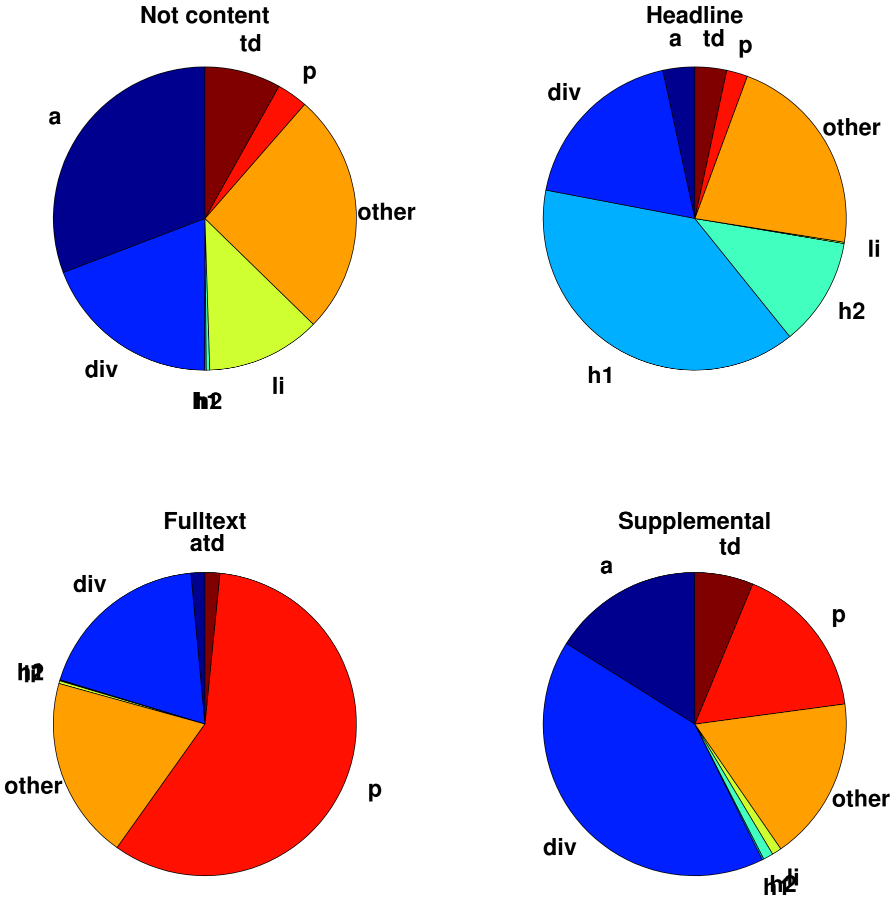


Figure 3.1: Distribution of tag of nodes.

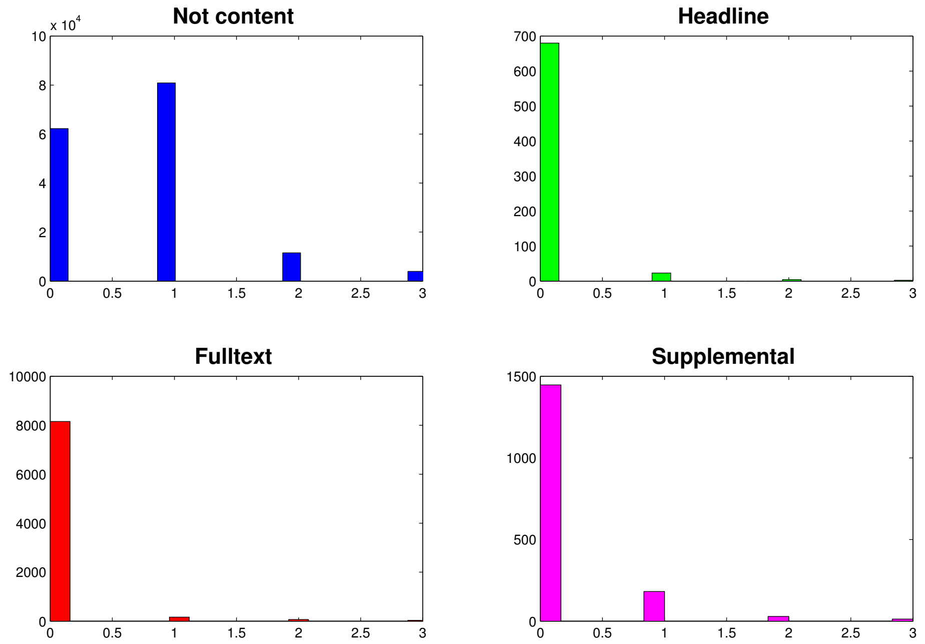


Figure 3.2: Distribution of number of children.

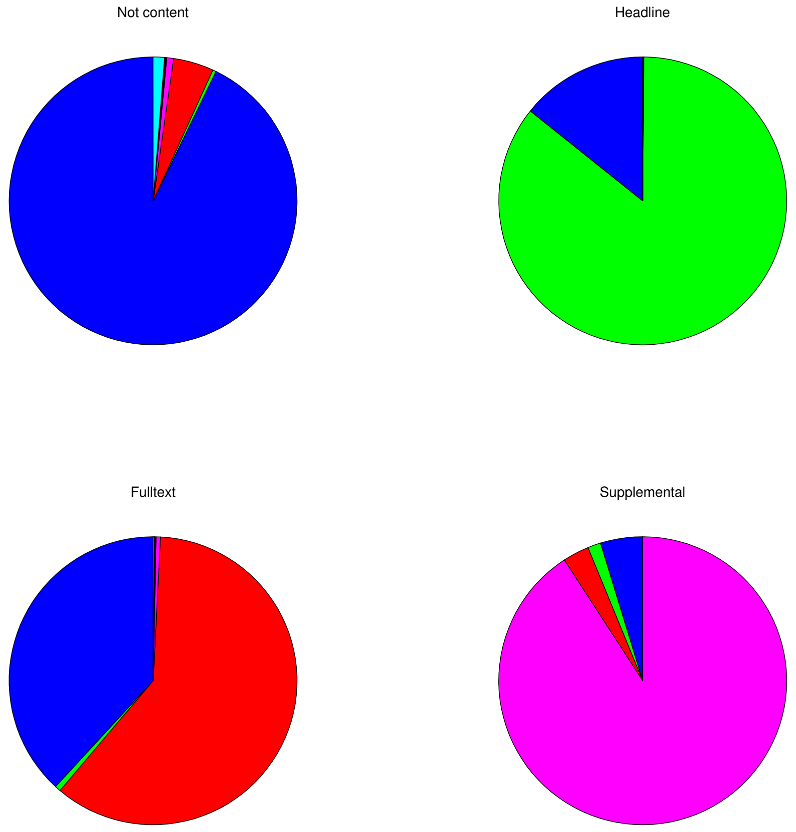


Figure 3.3: Distribution of type of children. Blue: Not content, Red: Fulltext, Green: Headline, Magenta: Supplemental.

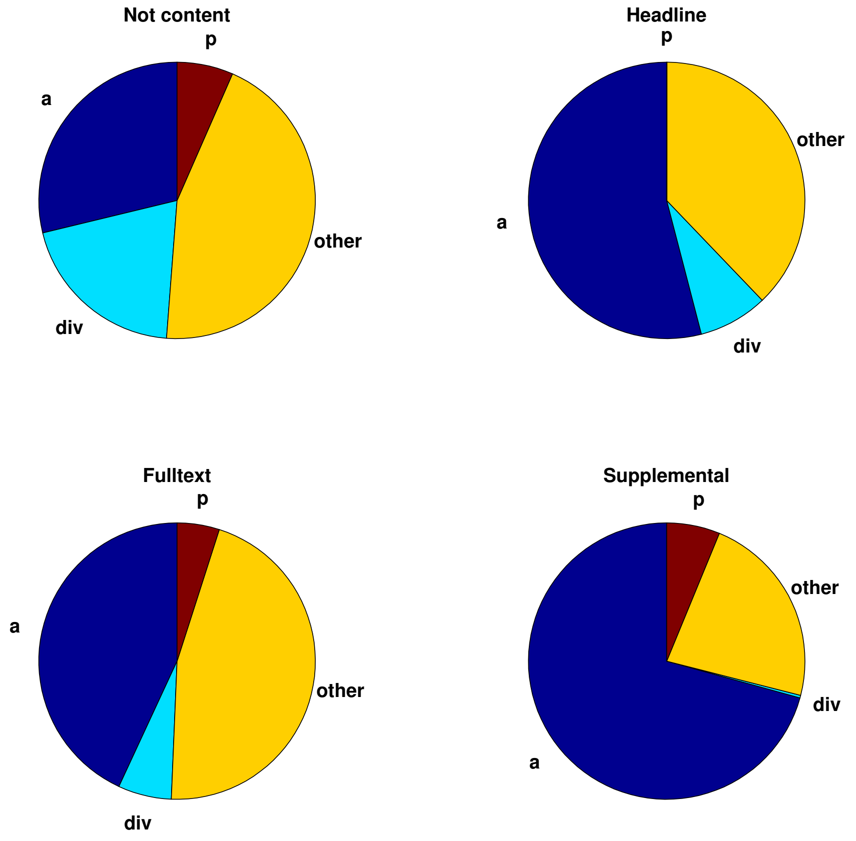


Figure 3.4: Distribution of tag of children.

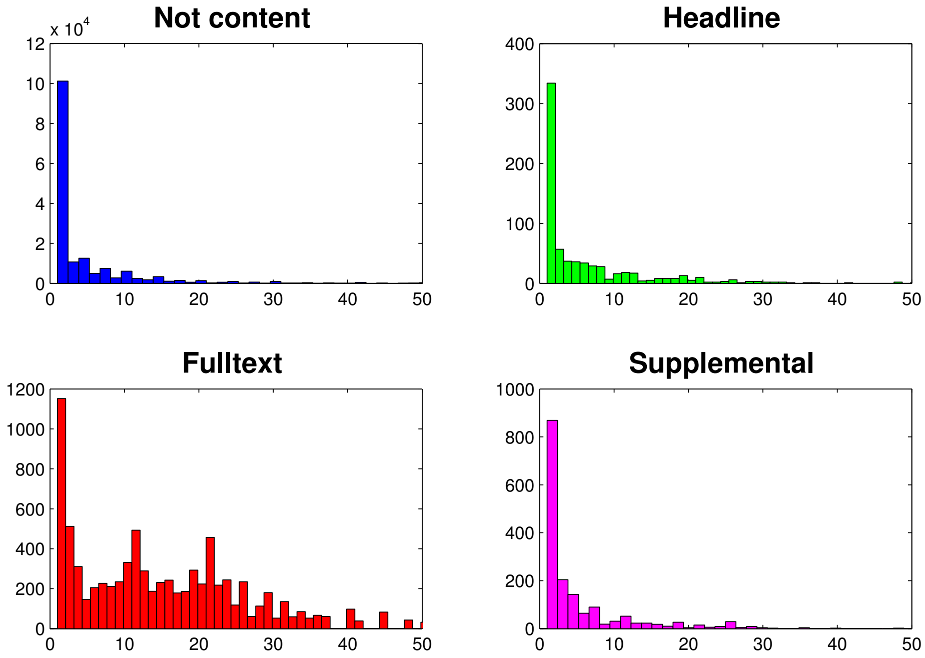


Figure 3.5: Distribution of number of siblings. Blue: Not content, Red: Fulltext, Green: Headline, Magenta: Supplemental.

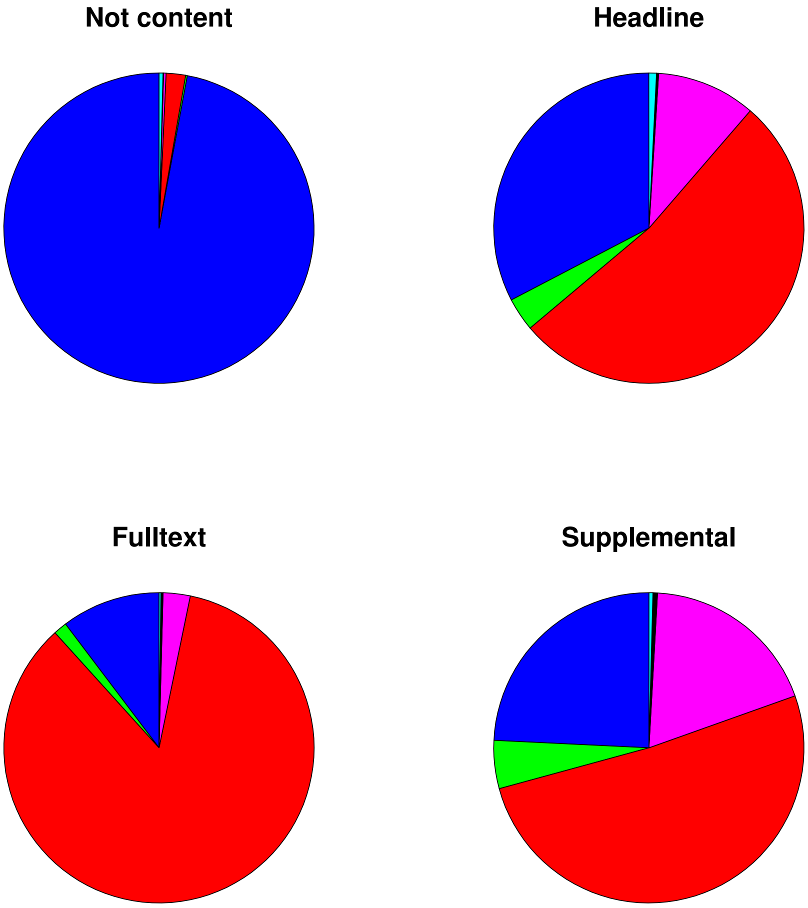


Figure 3.6: Distribution of type of siblings. Blue: Not content, Red: Fulltext, Green: Headline, Magenta: Supplemental.

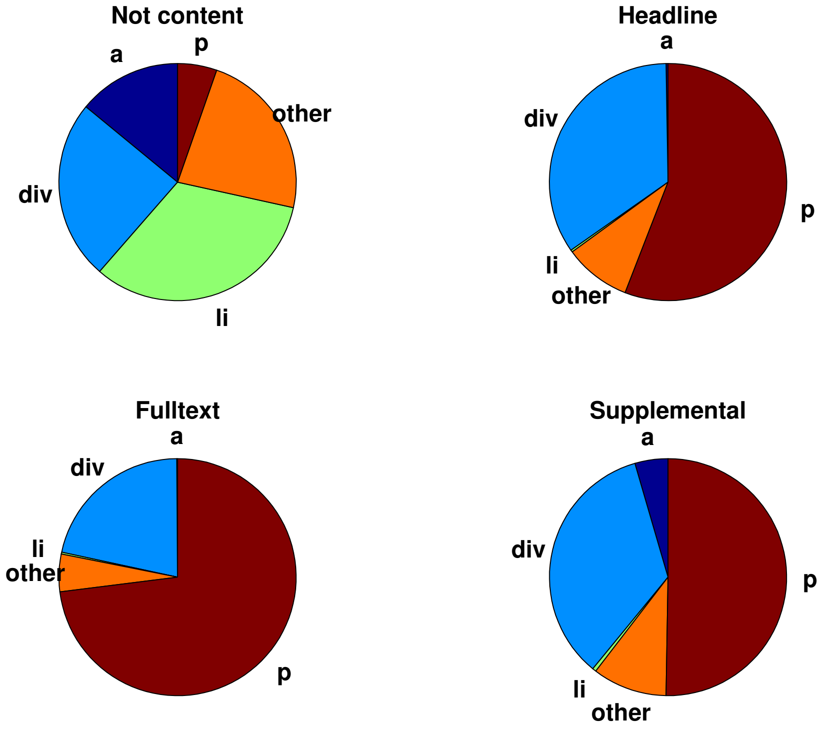


Figure 3.7: Distribution of tag of siblings.

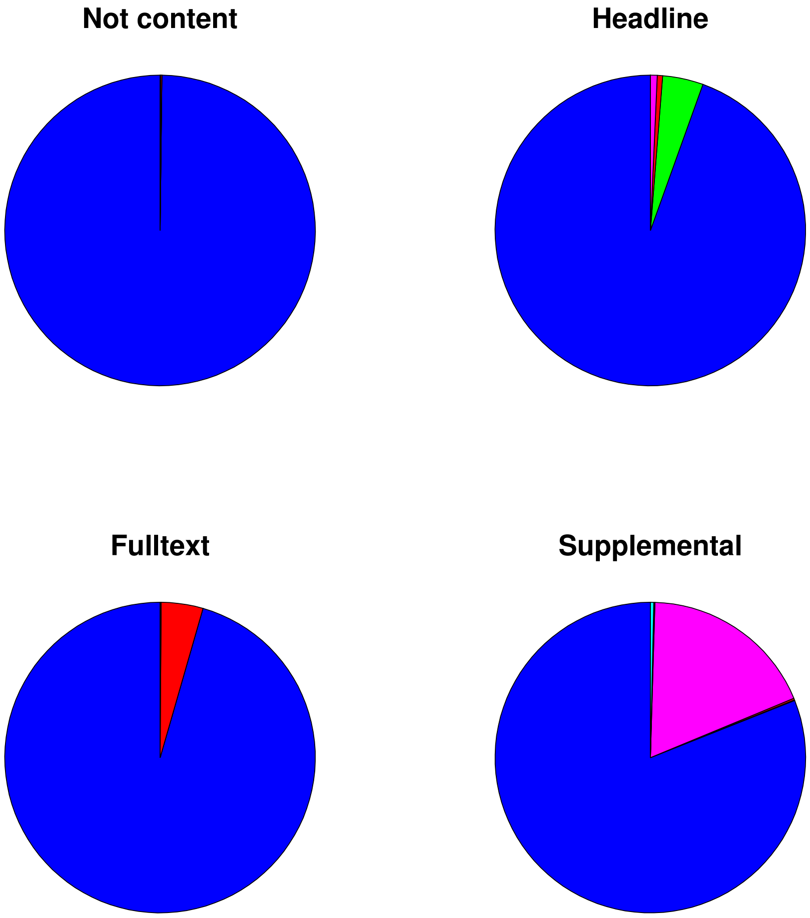


Figure 3.8: Distribution of type of parent nodes. Blue: Not content, Red: Fulltext, Green: Headline, Magenta: Supplemental.

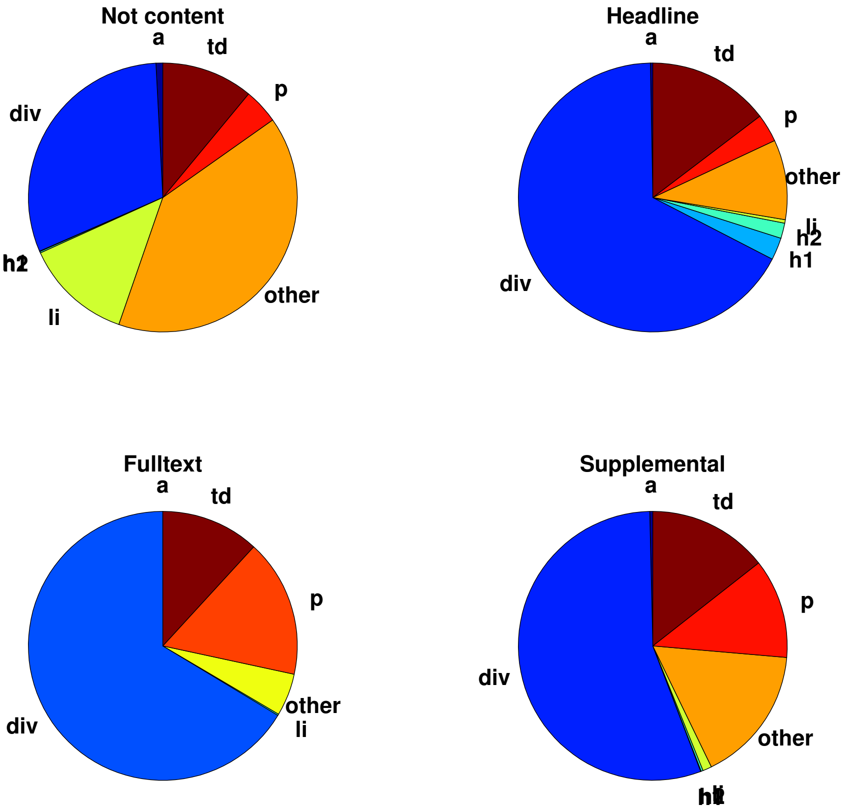


Figure 3.9: Distribution of tag of parent nodes.

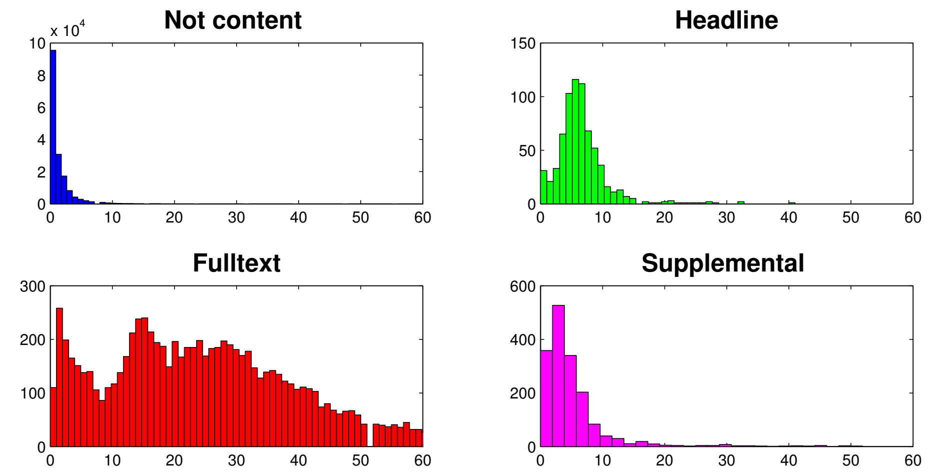


Figure 3.10: Distribution of number of words in each node.

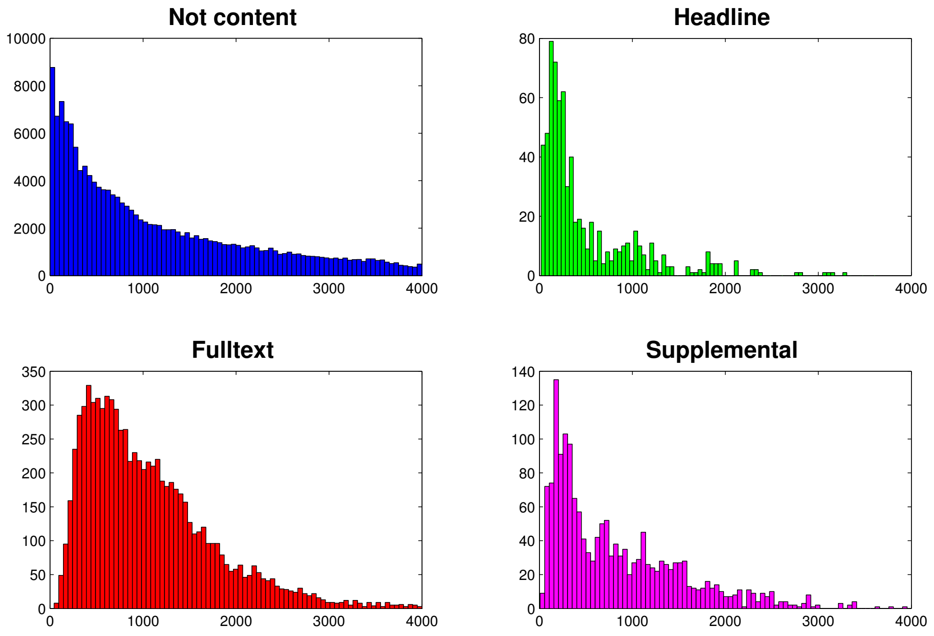


Figure 3.11: Distribution of node offset from top.

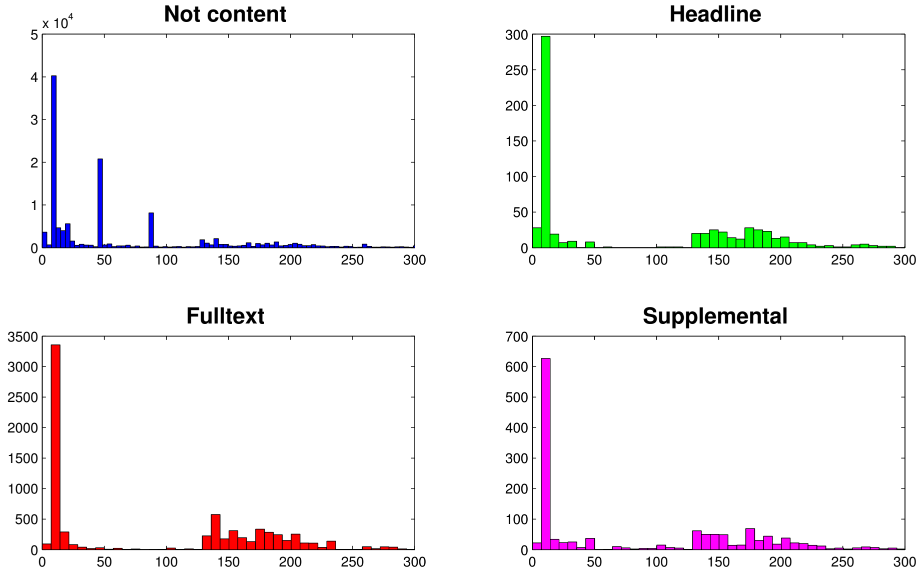


Figure 3.12: Distribution of node offset from left.

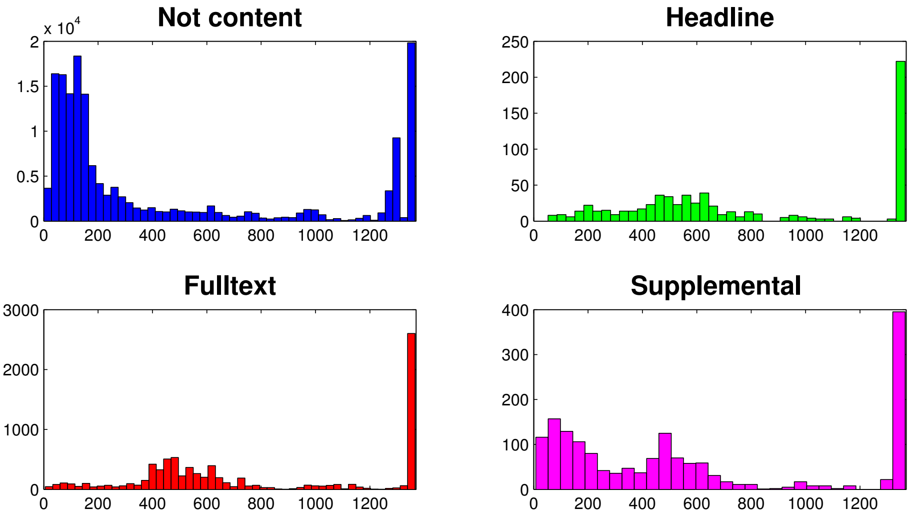


Figure 3.13: Distribution of width of rendering of nodes.
Max width of rendering is 1366 px.

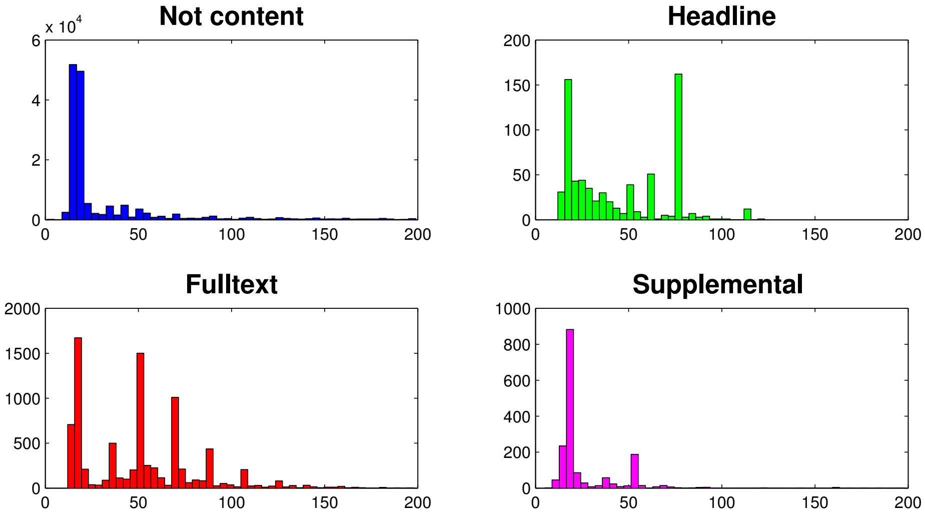


Figure 3.14: Distribution of height of rendering of nodes.

4

Results

The dataset is divided into two parts, a training set and a validation set. In order to test on all the documents we randomize the selection of training and validation data for each run. After fitting the model parameters to the training dataset evaluation is performed by running the Gibbs sampler on each of the documents in the validation dataset. For each document the hidden state of each node is estimated. Finally the estimated states are compared to the actual states.

Two measures build the foundation for our results; **recall** and **precision**. We define recall as the probability that we correctly label a token as the right type. This means that if we correctly label a node containing ten tokens, we count that as ten correct labels. Recall is defined as the probability that a token which we estimate to be content is actually content. The reason for presenting our results on token level is to enable comparison with Boilerpipe. We also make use of the F_1-measure defined as

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4.1}$$

For each document we compute the F_1-measure. In our presentation of the results we then order the documents ranging from highest F_1-measure to lowest, same as in figure 1.2. Kohlschütter et. al. presented their results computed on token level, rather than node level. We compute and present results for both. For the purposes of content extraction it makes sense to use results on token level, since nodes with little or no text have less importance for the quality of output.

In order to compare the effect of different configurations of the method and the performance on different types of nodes we present results for:

- **Complete analysis** Recall and precision measure based on exact type match for all available types.
- **Only maintext** Recall and precision measure considering only maintext nodes. Everything else is ignored.
- **Keep every node with >10 words** Recall and precision measure only differentiate between content and not content. Significantly easier than a measure based on exact type match. This difference is since a naive classifier based on the number of words in a node can only give a binary answer.
- **Keep all text** Same as for the results with >10 words kept.

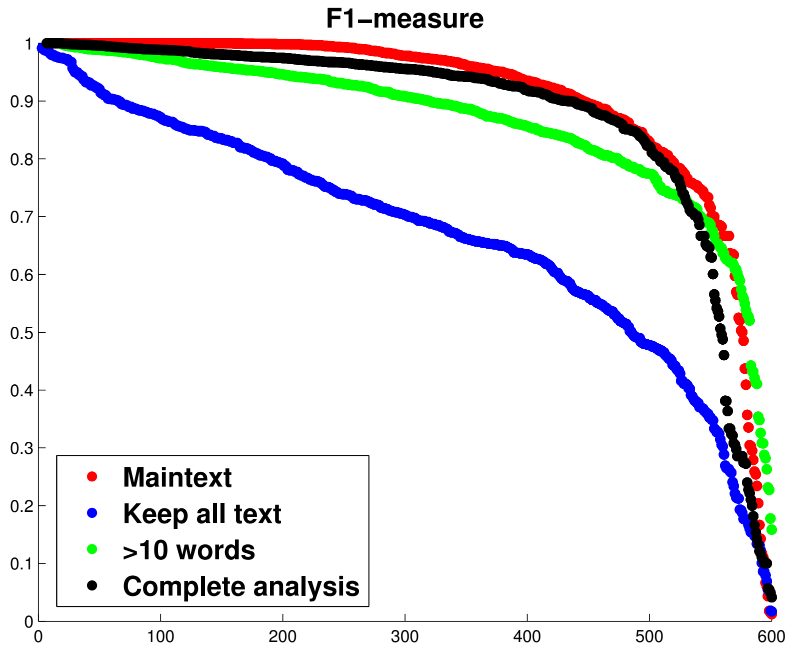
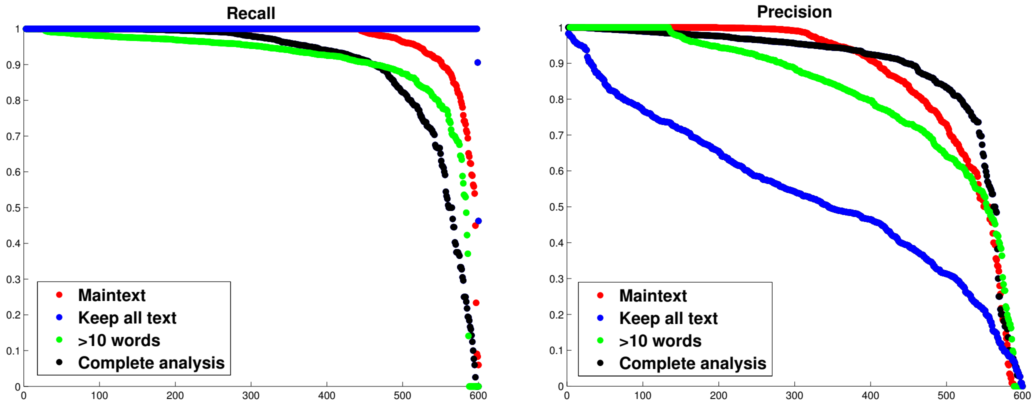


Figure 4.1: All types



(a) Recall

(b) Precision

5

Conclusions and Discussion

We can conclude from the results that there is potential to improve state of the art for content extraction through the introduction of a hidden Markov tree model. However, its difficult to ascertain for sure until all methods compared can be made to run on the exact same data set with the exact same set of pre- and postprocessing machinery.

5.1 Comparison with baseline

We present two different baselines; keeping all text and keeping text from all nodes with more than ten words. It turns out that for a binary classification between content and not content keeping all text from nodes with more than ten words proves to be rather efficient when evaluated on the token level. Looking at fig. 3.10 we see that this makes sense since most full text has more than ten words and therefore will be correctly classified. Since that also makes up the wast majority of the total number of tokens the results will appear quite good. But as we can see a lot of the headline and supplemental material, as well as a not insignificant amount of full text, will be missed. For the purpose of a multicategory classification with high demand for recall and precision this clearly would not be a good algorithm, although it may look similar at first glance of fig. 4.2b. Same with the even worse method keeping all text.

5.2 Comparison with Boilerpipe

Judging from fig. 1.2 it may seem like Boilerpipe outperforms our suggested method. This may however not be the case since Boilerpipe runs a postprocessing step that cleans up certain common mistakes. The most common error in output from our suggested method is mislabled user comments at the bottom of an article. This text is usually presented in a manner similar to the main text of the article. This could be taken care of with rule based postprocessing. This is not implemented in the method from which we have gathered our results. We suggest development of an evaluation environment ensuring equal conditions as a future work.

As we can see in figure 1.2 Kohlschütter [2] chooses to benchmark Boilerpipe against other algorithms on token level. The choice of comparing at token level results in smaller chunks of text being misclassified having smaller impact. This may be reasonable for some applications, but not for the context in which our method is meant to be used. Its enough to get one sentence with a critical entity-event relationship that is misclassified as content to get a strange signal. Therefore it would make more sense to compare the algorithms on node level. We suggest this as a future investigation. From our own inspection it appears that our algorithm outperforms in this case, but further investigation is needed to establish this with certainty.

5.3 Quality of Training Data

The training data set used for our evaluation is from 2008. This means that the standards employed in it are outdated. The reason we chose to still use it is to enable comparison with Boilerpipe. In order to use this method in a production environment better training data is required. A benefit of the method suggested is that for sources where high precision and recall is critical a particular model can be trained, for use on only that specific source.

5.4 Challenges

As mentioned the largest difficulties lie in finding smaller chunks of information such as supplemental information and related content. These nodes are much more similar and may be difficult if not impossible to distinguish from each other and *Not content*-nodes, regardless of feature vector. A potential way to overcome this could be to establish a context of the article using the fulltext, since that is much easier to find. Based on the established context, related and supplemental nodes can be distinguished depending on their similarity to the main text.

5.5 General Drawbacks with Hidden Markov Tree Models

A drawback with introducing the hidden Markov tree model is that they are computationally expensive and slow to converge. Especially when relying on information from the rendered page, which is expensive to harvest. We have not taken the computational expense into account in this project, but suggest it as a topic for future work, see 6. This thesis instead serves as a proof of concept regarding the introduction of hidden Markov tree models.

6

Future Work

A challenge in every project is to limit the scope and set clear parameters for success. Through the process of designing our content extraction method, several ideas for improvements have been stumbled upon. As investigate all of them would make the project impossible to finish we provide a list of ideas for improvements to be investigated in the future:

Optimization of selection of features

The selection of features of the data based on which the computations are performed have significant impact on the quality of the output. The fewer the number of features, the faster the extraction method returns a result. However, without enough features it will fail to recognize subtle differences between node states. It should be possible to write an algorithm to cluster the feature vectors of all nodes based on different combinations of features. By doing this it should be possible to determine the smallest set of features that creates distinct clusters of data, i.e. make it possible to differentiate between different type nodes.

Ruleset for special cases

Looking at the most common sources of errors we conclude that there is the possibility of introducing a set of rules that perform a sanity check on the output. For example nodes with certain html tags can be pruned away by default.

Optimization of algorithm

There is a lot to be done to reduce the run time of the Gibbs sampler. Presently the most computationally expensive part of the algorithm is instantiating the Dirichlet and Beta generators. Due to the design of the random generators in use they have to be instantiated for each set of input parameters. This makes for fast computation of successive samples from the same distribution, but slow computation of a single sample for distributions with different parameters.

Better datasets

Key to every machine learning based method is a large, high quality training data set. The data set used in this project is outdated, and therefore not representative of the structure of modern web pages. To create suitable datasets an annotation tool would be useful. Such a tool could be developed using JavaScript.

Better Layout Engine Implementation

The layout engine in use, CSSBox, is not capable of handling for example JavaScript rendered code. The is a good candidate substitution for this in the form of PhantomJS, which should be evaluated. Also, since standards for webdevelopment are prone to change quickly its important to use some package that is maintained and kept up to speed with current developments.

Make more use of the Recorded Future entity extractors

The presence of certain entities, i.e. organizations, journalists or individuals, in a string would be a useful feature to include in the data model. Especially when combined with a context based model. If for example the full text refers to a certain set of entities and events, any related or supplemental information is likely to be related to those entities. An even stronger relationship is likely present between the headline and full text.

Bibliography

[1] Recorded future. `www.recordedfuture.com`.

[2] Boilerpipe benchmark against safari. `https://code.google.com/p/boilerpipe/wiki/Benchmarks`, November 2010.

[3] Readability. `http://blog.arc90.com/2010/06/07/safari-5-another-step-towards-better-reading-on-the-web/`, June 2010.

[4] Shumeet Baluja. Browsing on small screens: Recasting web-page segmentation into an efficient machine learning framework. *WWW*, 2006.

[5] Ziv Bar-Yossef and Sridhar Rajagopalan. Template detection via data mining and its applications. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, pages 580–591, New York, NY, USA, 2002. ACM.

[6] Deng Cai, Deng Cai, Shipeng Yu, Ji-rong Wen, Wei-ying Ma, Deng Cai, Shipeng Yu, Ji-rong Wen, and Wei-ying" Ma. Vips: a vision-based page segmentation algorithm. *Microsoft Research*, 2003.

[7] David Gibson and David Gibson. The volume and evolution of web page templates. *WWW*, ACM 1-59593-051-5/05/0005., 2005.

[8] Hung-Yu Kao, Jan-Ming Ho, and Ming-Syan Chen. Wisdom: Web intrapage informative structure mining based on document object model. *IEEE Trans. on Knowl. and Data Eng.*, 17(5):614–627, May 2005.

[9] Christian Kohlschutter. Boilerpipe source. `http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/`.

[10] Christian Kohlschutter. A densitometric analysis of web template content. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 1165–1166, New York, NY, USA, 2009. ACM.

[11] Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features. In *Proceedings of the Third ACM International*

Conference on Web Search and Data Mining, WSDM '10, pages 441–450, New York, NY, USA, 2010. ACM.

[12] Lan Yi, Bing Liu, and Xiaoli Li. Eliminating noisy information in web pages for data mining. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 296–305, New York, NY, USA, 2003. ACM.

[13] Lan Yi and Lan Yi. Web page cleaning for web mining through feature weighting. *IN INTL. JOINT CONF. ON ARTIFICIAL INTELLIGENCE (IJCAI*, pages 43–50, 2003.