Automatic data extraction from online discussion boards

by

Erik Victor Hvistendahl, Xiongjie Chen

Master Thesis in
Information and Communication Technology

Faculty of Engineering and Science
University of Agder

Grimstad, June 2009

Abstract

There has been written many papers on field of mining data from structured web pages. However, few if any of these papers focus on the area of retrieving specific parts of discussion board postings. A discussion board page contains a set of postings, which can be considered data-records. Our goal is to provide insight on a specific approach to identify the locations of author, content and date+time, which are parts of a complete discussion board posting data-record.

Our approach consists of combining a Naive Bayes pattern classifier, structure classification and grammar to identify the sought after elements. We give a thorough evaluation of our Naive Bayes classifier and it's components in addition to how combinations of the different parts in our approach affected the overall result.

Our best results for identifying the location of the individual elements was 94% for author, 76% for content, 86% for date+time and 60% for getting every element of each post correct. While the result for getting the complete posts is not very good, it does depend a lot on the other results. We believe our approach shows promise and with further development and refinement, it will be a viable method for automatic extraction of data from on-line discussion boards.

Preface

This thesis was written as a part of our Master degree in Information and Communication Technology at the University of Agder. The subject was proposed by Integrasco A/S, to whom we are grateful for allowing us to perform this master thesis. Our supervisor throughout this project was Ole-Christoffer Granmo.

We would like to thank Ole-Christoffer Granmo for his support and patience. When we were struggeling with certain concepts, he walked us through them and showed us the finer details which allowed us to complete the project. We would not have been able to complete this project without his input and suggestions. We would also like to thank our contact persons at Integrasco A/S, which was Jaran Nilesn and Alexander M. Stensby. They have given us valuable input and suggestions throughout the process of writing this thesis, which helped us bring the thesis to it's current level of quality. Without the support of these people or Integrasco A/S, this thesis would not have been possible.

Grimstad, June 2009
Erik Victor Hvistendahl, Xiongjie Chen

Contents

List of Figures

List of Tables

Chapter 1

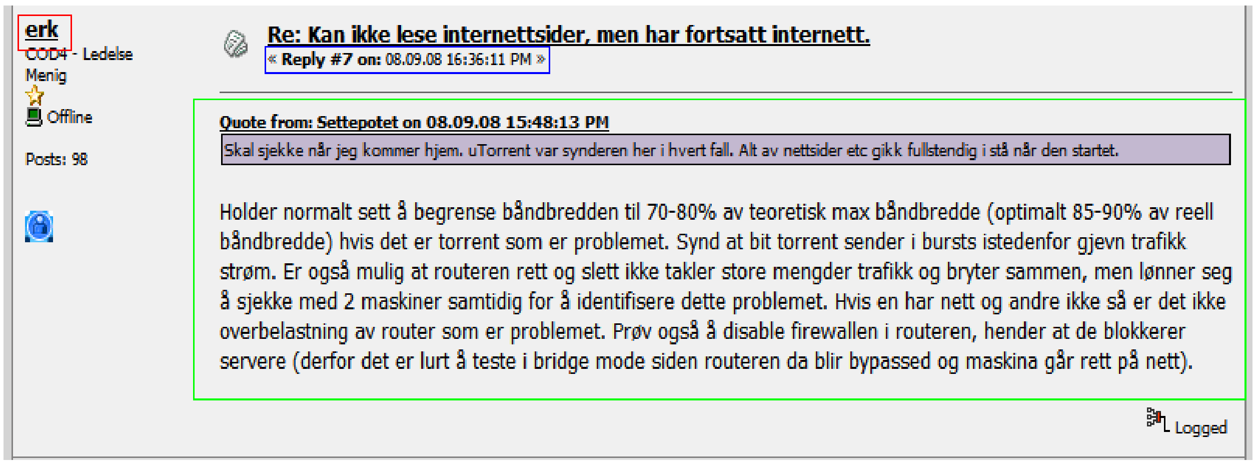
Introduction

In this chapter we give a short introduction to extracting data from on-line discussion boards, the methods we use in our approach to automating the data extraction and our contributions to knowledge.

1.1 Problem Statement

On-line discussion boards come in a wide variety of flavours. They provide the same basic functionality, but vary in the underlying code used to display web pages on computer monitors. The code behind the web pages is usually eXtensible Hyper Text Markup Language or XHTML for short, which conforms to XML. Actors who wish to gather data from discussion boards, must analyse this code in order to extract the desired data. Figure 1.1 shows an example of what a posting on a discussion boards can look like when shown on a computer monitor. The desired data has been marked with square boundaries. The author of the post is in the upper left corner, the date+time is in the upper right square and the main content of the posting is the square on the lower right.

When looking at a discussion board trough a web browser, it seems like an easy task to identify certain data elements within a posting. This is because the content that is displayed on your screen, has been designed to be human readable. When viewing the XHTML code in pure text, the task


erk
COD4 - Ledelse
Menig
Offline

Posts: 98

Re: Kan ikke lese internettsider, men har fortsatt internett.
« Reply #7 on: 08.09.08 16:36:11 PM »

Quote from: Settepotet on 08.09.08 15:48:13 PM
Skal sjekke når jeg kommer hjem. uTorrent var synderen her i hvert fall. Alt av nettsider etc gikk fullstendig i stå når den startet.

Holder normalt sett å begrense båndbredden til 70-80% av teoretisk max båndbredde (optimalt 85-90% av reell båndbredde) hvis det er torrent som er problemet. Synd at bit torrent sender i bursts istedenfor gjevn trafikk strøm. Er også mulig at routeren rett og slett ikke takler store mengder trafikk og bryter sammen, men lønner seg å sjekke med 2 maskiner samtidig for å identifisere dette problemet. Hvis en har nett og andre ikke så er det ikke overbelastning av router som er problemet. Prøv også å disable firewallen i routeren, hender at de blokkerer servere (derfor det er lurt å teste i bridge mode siden routeren da blir bypassed og maskina går rett på nett).

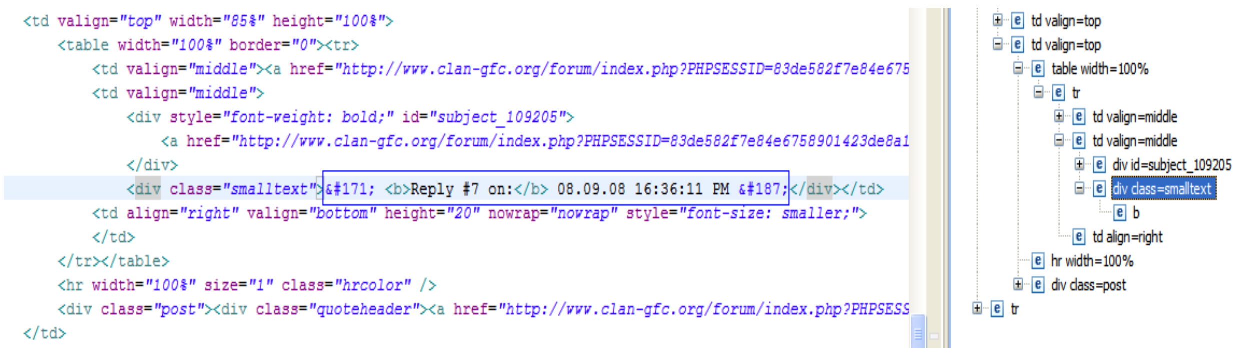
Logged

Figure 1.1: Discussion Board Posting

of reading the information in the discussion board becomes much harder. Figure 1.2 shows part of the XML code, where elements of the desired data from Figure 1.1 can be found. We have marked the date+time location of the posting shown in Figure 1.1. This is how that type of data is kept in the code behind on this specific discussion board. There are several ready made discussion board types that are in use across the web. It is not unusual for these to have been modified to suit the needs of the people hosting the discussion board. There are also some discussion boards that have been custom built and may differ greatly in structure from the most commonly used discussion boards. The location of the desired data will vary from discussion board to discussion board, even among those who use the same ready made discussion board as their basis. This makes it difficult to make a one fits all method for identifying the elements that one wants to extract. On the right side of Figure 1.2 you can see parts of the Document-Object-Model or DOM-tree which shows the structure of the XML.

1.2 Problem Introduction and Motivation

The sponsor of this thesis is Integrasco A/S, which is company that mainly focuses on mining and analysing data from on-line discussion boards. They retrieve vast amounts of data from many different on-line discussion boards and need methods for identifying different attributes like author of the post, main post content, posting date and time and other fields of interest. During this report,



```
<td valign="top" width="85%" height="100%">
    <table width="100%" border="0"><tr>
        <td valign="middle"><a href="http://www.clan-gfc.org/forum/index.php?PHPSESSID=83de582f7e84e675
        <td valign="middle">
            <div style="font-weight: bold;" id="subject_109205">
                <a href="http://www.clan-gfc.org/forum/index.php?PHPSESSID=83de582f7e84e6758901423de8a1
            </div>
            <div class="smalltext">&#171; <b>Reply #7 on:</b> 08.09.08 16:36:11 PM &#187;</div></td>
        <td align="right" valign="bottom" height="20" nowrap="nowrap" style="font-size: smaller;">
        </td>
    </tr></table>
    <hr width="100%" size="1" class="hrcolor" />
    <div class="post"><div class="quoteheader"><a href="http://www.clan-gfc.org/forum/index.php?PHPSESS
</td>
```

```
⊞ e td valign=top
⊟ e td valign=top
    ⊟ e table width=100%
        ⊟ e tr
            ⊞ e td valign=middle
            ⊟ e td valign=middle
                ⊞ e div id=subject_109205
                ⊟ e div class=smalltext
                      e b
              e td align=right
      e hr width=100%
    ⊞ e div class=post
⊞ e tr
```

Figure 1.2: DOM-Tree

we will refer to nodes that contain date and time as date+time. Integrasco A/S currently relies heavily on XSLT and Xquery transformation documents to extract the data that they want. They use Xquery to retrieve collections of XML data and eXtensible Stylesheet Language Transformations, or XSLT for short, for moving the data from XML into a standard form.

Writing Xquery and XSLT transformation documents involves large amounts of manual labour. In order to make their data extraction more dynamic and scalable, Integrasco A/S wants an automatic process of identifying the relevant parts of the discussion board. There are currently no methods available to them that can fulfil this requirement. They therefore desire research into ways of the teaching data extractors which information is relevant. The core problem is how to improve automated analysis and identification of the desired elements in the XHTML structures, in order to ensure correct and consistent extraction of relevant data. Integrasco A/S wants to move the work of transforming the data from on-line discussion boards from manual labour to fully automatic or semi-automatic with minimal user input.

Integrasco A/S has mined close to 500 different discussion boards and more than 7,000,000 new posts are retrieved each month from discussion boards world wide in different languages. The current method for retrieving this data is to write the transformation document manually for each new forum. It is normal for people to spend between one and one and a half day to write a completely new transformation for a single discussion board. If there is a template already available for this type of discussion board the time required is reduced to around half a day. Due the manual labour involved, errors in the transformation documents are quite common. Every day, between five and twenty errors needs to be corrected. Most of the errors come from unexpected exceptions that were not discovered during the transformation writing process. In addition to this, small changes to the structure of a discussion board often cause manually written templates for this board to become void.

Using manually written XSLT and Xquery transformation documents for retrieving data from on-line discussion boards carry several limitations and makes it difficult to scale the data mining operations. The most important limitations are:

- **Human resource**
- **Correctness**
- **Extendibility**

1.3 Main Problem

Certain data like author, post content, time and date and similar are elements in recurring structures in on-line discussion board pages. These elements needs to be identified in order for the data extractors to do their job properly. We will investigate if it is possible to identify these recurring structures and the specific elements, through the use of machine learning and pattern recognition.

1.4 Our Approach to the Problem

Analysis of the structure is the first part of providing the data extractors with data about which XHTML elements to read. The second part is to identify which elements contains author name, content and other information of interest. Missing one or more of the desired data elements will lead to an incomplete data set. For easy processing of the discussion, the data needs to be moved over to a standard form where date+time, author and similar are in known positions. To perform these operations, the data extractor needs to know what a post looks like in the given discussion board. This includes the location of each element that contains desired data. Our goal is to investigate methods for using machine learning and pattern recognition to provide this information. To this end, we will create a program that will serve as our test platform which makes use of pattern recognition techniques. The research data and experiences gathered while making the test platform will be used to create a simple prototype.

- **Classification of the board structure and finding the location of a discussion board posting**

  The XHTML is built as a DOM-tree which contains style settings and data for displaying web pages. We can find the location of the discussion boards postings by analysing this tree and looking for recurring patterns. Analysis of the DOM-tree to find data, is the recurring theme in the papers we have read on extracting data from web pages. By reducing the area of XHTML that we need to search through to find the desired data we will increase both the accuracy and efficiency of our algorithms.

- **Identification of elements which contain important data and their data type**

  Naive Bayes is an established method for pattern recognition. In our research we will investigate it's effectiveness for identifying which elements in the DOM-tree contains specific types of desired data. The Naive Bayes is especially good for recognizing text, which is something all the desired data elements contain. This makes it a good choice for classifying the nodes in the DOM-Tree.

- **Introduce EBNF style grammar for the structure classification**

  EBNF grammar is primarily used for building domain models for computer languages. It sets the rules for what for example a sentence can look like. We will use the grammar to give suggestions to the structure classification about what the it should look for in the different elements. The object of using grammar, is to give the classification of the XHTML code more guidelines to work with and help reduce ambiguities. In our thesis we will primarily investigate how much we can gain by using simple EBNF style grammar for specifying what certain elements look like. The grammar will primarily be used as a helper for the structure classification.

- **Combining the three methods above and investigating their influence on each other**

  The three methods mentioned above forms the basis of our research into extracting data automatically from discussion boards. Alone, neither method will be a complete solution to our problem. Integrating structure classification, Naive Bayes and grammar to form a complete program, that finds the desired data is therefore necessary. Investigating how the integration influences the results we get, will be a key part of this thesis.

1.4.1 Approach Summary

The primary goal of this project is to research how efficient a combination of structure classification, Naive Bayes and EBNF grammar is in finding the specific data elements in on-line discussion boards. Pattern classification and machine learning are key methods for this project. The research itself can contribute to the field of data mining. In addition, research in method of automatic data extraction will be of great value to Integrasco and other actors in the field of data mining in determining if this is as an approach that is worth pursuing.

1.5 Importance of Topic

There are two different groups that have interest in the topic of our Master Thesis. One group consists of companies that mine the web for information or do similar activities. The other group would be those interested in knowing about which types of approaches work and which are dead ends when it comes to classifying data in recurring structures.

1.5.1 Cost Reductions in Gathering Data From Discussion Boards

Currently companies that wish to mine data from discussion boards write transformation documents manually. This is time consuming and requires personnel to do the job for each message board. If something changes in the message board structure, the transformation document has to be rewritten to incorporate these changes. Using a program to read message boards removes the need for manual labour. It would also be able to handle changes in message board structures which ensures that the message boards can always be read. Due to the removal of manual labour, the program has the potential to increase the speed at which web crawlers can mine data from message boards while reducing the cost.

1.5.2 Increasing Knowledge About Approaches to Classifying Structured Data

From the data mining perspective, the Internet is a huge database where there is a lot of desired data. While our focus is on on-line discussion boards, we are researching how to get certain data from web pages with recurring data structures. The principles that we research is related to any type of web page where there is a pattern that can be recognised. Our research into the method of analysing the DOM-tree structure combined with Naive Bayes and EBNF grammar, aims to give indications of what worked well, the results certain adjustments yielded and which experiments turned out to be mistakes. To our knowledge EBNF grammar has not been used as a helper for the DOM-tree structure classification in combination with Naive Bayes. The amount that it helps or hinders us will be of interest to those who wish to further work in this field.

1.6 Research Approach and Methodology

Our approach in this project, was to use the pattern recognition part of machine learning to recognise discussion board postings and the specific data types that we wanted to identify. By creating a simple research platform we were able to gather research data. We then used this data to build a simple prototype, which we ran through two stages of testing. The first stage was to do qualitative testing until we felt the prototype was ready for the second stage, which was quantitative testing.

1.6.1 Qualitative Testing

The first step was to gather data by using our research platform. The data that we gathered during this phase was used to determine how we approached the task of identifying desired elements of the discussion boards. By using a very limited amount of example forums we were able to test the programs learning capabilities in a controlled environment. This allowed us to ensure the quality of our program.

1.6.2 Quantitative Testing

The goal of the prototype is to be able to find desired data from discussion boards in large quantities. We therefore needed to do quantitative testing as well on our program. However this was mainly done once we have gotten satisfying results in qualitative testing. We gradually increased the number of discussion boards in our experiments and continuously made improvements to our algorithms based on the results. This allowed us to judge how the algorithms and pattern classifiers worked and find their weaknesses.

1.7 Limitations and Key Assumptions

When making a program using machine learning and pattern classification one has to set certain requirements for the program while recognising it's limitations. For a pattern recognition engine there has to be a pattern to analyse. If each data structure occurs only once or elements of the same type occur in very different locations then there is no pattern that can be recognised and our algorithms will not work correctly.

1.7.1 Assumptions About the Material to be Analysed

The algorithms will be configured to work on proper XML. We therefore assume that the web page structure has been cleaned up, before our program is asked to analyse it. This can be done through a module in the prototype platform or in a separate program.

In a flat structured discussion board, where all the posts are shown in chronological order, every normal post should have the same basic elements. For example, there can be author, main content, signatures, and date+time in most of the posts. Even if a few posts on the same discussion board does not have a signature included, they should still have a similar structure when compared to the posts that include signatures. In this case author, content and date+time would be the basic elements. Another assumption that we make is that all the data like author, content and date+time is available in XML. If this is pulled dynamically through page scripting, there will not be a complete data set for the program to analyse.

1.7.2 Limitations of the Program and Algorithms

Since the algorithms works on the XML structure of the XHTML page, it will make it very difficult for us if the XHTML is incorrectly formatted. The XML structure can be verified and in most cases cleaned up by tools. In cases where we can not properly clean up the XHTML, it will not be possible for us to do an analysis.

A limitation of machine learning approach is that the algorithms might not be able to extract data from discussion boards that are radically different from the norm. This is because the pattern classifiers will not have been trained to recognise the elements that very unique discussion boards can contain. Another problem would be if changes to the standards for XHTML formatting are introduced. Both these cases may require modifications or new supervised-learning sessions.

Our pattern recognition techniques will not work on discussion boards that rely heavily on JavaScript and similar to display data in a dynamic fashion, outside of the XML structure. Our approach will work on analysing the XML, and therefore requires the data to be available in the XML code itself. For discussion boards where the data is retrieved dynamically, visual identification would most likely be required and is outside the scope of this thesis.

1.8 Contributions to Knowledge

We built a simple grammar for the desired elements in the XHTML code behind the discussion boards, which our pattern classifier used as guidelines. The differences between adding or removing the grammar is an important part of our results.

To our knowledge using EBNF style grammar in combination with Naive Bayes pattern recognition and structure classification has not been done before in the field of identifying XHTML elements in web pages. We are unsure if such a combination has been used much at all in any setting. During our thesis we show the benefits of using such a combination and the difference in the algorithms' ability to find desired data, when just using parts of this combination.

In order for us to train the classifier and be able to ensure that we have identified the correct nodes, we needed to mark the nodes in the DOM-Tree. To this end, we created a program for marking the nodes that contain a desired data element and created a page set that we used in our research. The process we used for marking the nodes is one contribution to knowledge. We also made a set of discussion board pages, where nodes that contain author, content and date+time has been marked. This page set can be used by others who wish to do research into the same field. The location of the complete page set can be found in Appendix A.5

We came up with features for our Naive Bayes, which can be used when classifying a complete DOM-Tree. We gave our reasoning behind them and evaluated their quality in experiments. These descriptions and results can be used by people who wish to work on similar problems, when they want to select features for a Naive Bayes.

We have given results and indications on how the different combinations of methods influenced our ability to find specific data records in discussion boards. Our descriptions of what worked and didn't work can help others who wish to do work in similar fields, with deciding how to approach their problem.

We made a simple prototype which contains the implementation of our approach. The location of the source code files can be found in Appendix A.6.

1.9 Report Outline

Chapter 2 is where we present background material that is relevant to our thesis. It gives a short introduction of the papers that we based our research on, the Naive Bayes classifier and the tools and computer languages that we used in our project. We use chapter 3 to present our initial research and the tools we made. Chapter 4 presents our pattern classifier, structure classification and grammar, along with the reasoning behind them. Chapter 5 contains our experiments and an analysis of the results we got. Chapter 6 is our engineering chapter. It contains a quick overview of our prototype from an engineering perspective. The final chapter is chapter 7 where we give our conclusions and our ideas for further work on this topic.

Chapter 2

Background

Here we give a bit of background material that is related to our thesis. There is also a short description of most of the tools and technologies that we use in our thesis. The chapter is primarily for those who are not familiar with automatic extraction of data from web pages or the Naive Bayes pattern classifier.

2.1 Prior Research on Topic

We have focused on three parts in our search for prior research on this topic. The Naive Bayes classifier is an well established and documented method so we chose to not include a paper on this. Our main focus with the papers is toward classification of the web page structure itself. Papers [2], [3] are related to each other as [3] builds on the research in [2]. For the grammar we chose to use a paper about building a domain model for extracting meaningful sentences from quickly typed interviews and similar. At first glance that paper might seem unrelated but we consider the pre-processing to be similar to what is done in the papers that deal with structure classification.

2.1.1 The Naive Bayes Classifier

Naive Bayes is also called ”idiot” Bayes since it it evaluates all the features of an object independently even if the value of the features depend on each other. The classifier is based on Bayesian decision theory and statistics.

The first step in building a Naive Bayes classifier is to assign a class for each data type that you are looking for. You then decide on the features that describes each class. For a sentence, a possible feature could be average number of words. Another feature could be number of words that belong to a certain class. Once the features have been decided upon, the classifier needs to be trained. The training, of the Naive Bayes classifier, consists of gathering statistical data. The statistics are about how the features you have selected, look like in samples where data has been marked as belonging to a certain class. After the statistical data has been gathered, a statistical distribution needs to be selected. Each class that shares a feature has an independent probability distribution for the feature, since it is to be evaluated on a class by class basis. The distribution is used used to determine the probability that a feature value that has been retrieved from an unknown sample, belongs to a certain class. Once all the features in the sample have been evaluated, the probabilities are combined to give the final probability of the sample belonging to a certain class.

The Naive Bayes formula can be described as:

$$Posterior = \frac{PriorXLikelihood}{Evidence}$$

A more thorough description of Naive Bayes and the theoretical background behind it can be found in [1].

2.1.2 Data Structure Classification in Web Pages

RoadRunner: Towards Automatic Data Extraction from Large Web Sites [2]

The focus in roadrunner is to compare XHTML pages and generate a wrapper based on their similarities and differences. This wrapper is then used to extract data from the web pages. The basic method is to compare sources between different pages and analyse the matching level. First it will get a sample page, and then compare the it with a second sample page. During the comparison there should be some elements which match and mismatch between the samples. They use this data to generate their wrappers and extract data. The roadrunner paper dates back to 2001 and the examples shown deal with a different type of web pages than we are going to analyse. The ideas found in the paper does however still carry relevance to the research we will be doing in our master thesis.

Extracting Structured Data from Web Pages [3]

Paper about how to recognize recurring data structures in web pages, which builds on the Roadrunner paper. It deals with the problems encountered when some items, like books in a book listing have different number of authors etc. and how to recognize this. The focus in this paper is on the data structure and how to extract content from it. It attempts to answer how to deal with ambiguity inside of the data set and presents their algorithm EXALG. It works on the basis that important elements or fields of equivalence will occur frequently. They use equivalence classes and a module that will match a certain term with placement in the structure. If the same term occurs in the same place in the structure it will be considered the same equivalence class. Once their algorithm is done with the classification it constructs a template and extracts the data values.

Web Data Extraction Based on Partial Tree Alignment [4]

On the basis for their research, the assumption is the expectation that data which starts in one part of the DOM-tree, does not end in a completely different part of the DOM-tree. Their experiments show that this assumption is correct in most cases. The technique that they use, builds on their previous algorithm which they call MDR. This technique does the data extraction in three steps. Their first step is to take the HTML elements and build the DOM-tree. The second step is to mine entire data regions before they identify the individual data records in the third step. This method is not aware of which data records are interesting to the user, instead it extracts all the data records. Their new technique produces one rooted tree for each data record and then combines them into a tree that contains all the data records. They then use their partial tree alignment algorithm to combine related nodes while keeping the hierarchical structure intact. They tested their algorithms ability to extract records data on 72 randomly chosen web pages. They had a result of 97.10 precision with their old MDR method while they had a 99.82 precision rate for their new algorithm. Only the new algorithm, was tested on extracting specific data records. They noted that it performed poorly at this task, due to quite a few errors in finding right data records.

2.1.3 Building the Structure Definition Templates

Specific Domain Model Building for Information Extraction from poor quality corpus [5]

This paper deals with getting data from quickly typed interviews. In quickly typed interviews there will be typing errors, which they deal with by doing pre-process. That part is not of much interest to us. However the majority of the paper is on how to make rules for the grammar, which their extraction engine uses for classification and how to use these rules. It also deals how to make use of these rules in an extraction engine. They use two types of rules, constituent rules and predicative rules. Constituent rules define how certain types of expressions can be used, while the predicative rules define a set structure around an object. The rules are defined in BNF grammar and stored in a database. This is of interest when it comes to building the rules for our algorithms.

2.2 Tools and Formats

During the project we have used some external tools and work with certain standards. In this sub-chapter we will give a quick introduction of these.

2.2.1 Tag Soup

Tag Soup is program which cleans up HTML to make it conform to XML standards. Below is a description of Tag Soup taken from their homepage:

> *This is the home page of TagSoup, a SAX-compliant parser written in Java that, instead of parsing well-formed or valid XML, parses HTML as it is found in the wild: poor, nasty and brutish, though quite often far from short. TagSoup is designed for people who have to process this stuff using some semblance of a rational application design. By providing a SAX interface, it allows standard XML tools to be applied to even the worst HTML. TagSoup also includes a command-line processor that reads HTML files and can generate either clean HTML or well-formed XML that is a close approximation to XHTML.*[6]

2.2.2 Xerces

Xerces [7] is an XML parser, which is part of the Apache XML Project [8]. Description below is taken from the Xerces home page:

> *The Xerces Java Parser 1.4.4 supports the XML 1.0 recommendation and contains advanced parser functionality, such as support for the W3C's XML Schema recommendation version 1.0, DOM Level 2 version 1.0, and SAX Version 2, in addition to supporting the industry-standard DOM Level 1 and SAX version 1 APIs. [7]*

2.2.3 Java and JavaScript

Java [9] is an object oriented programming language developed by Sun Microsystems [10]. The code is normally compiled to byte code which then runs on a virtual machine. The virtual machine lies as an intermediate layer between the compiled code and the operating system. This allows Java programs a very high degree of portability.

JavaScript is a script language that can run in web-browsers, and was created by Netscape [11]. One of the uses in web-browsers is to handle client logic and display content. There are some differences in how the different web-browsers handle JavaScript.

2.2.4 XML, DOM-Tree, HTML and XHTML

XML stands for 'Extensible Markup Language' and is a specification for a markup language. New markup languages can be be created with XML as the basis. XHTML is an example of a markup language that extends XML. The format of XML consists of elements which may contain other elements. Each element either has a start tag and end tag or a tag which starts and ends within it self. The tags are enclosed within '<' and '>' and may have attributes. The first token in a tag gives the name, while the next tokens are attributes. Closing of the tag is done with the '/' token. Below is a simple example of XML. The first tag has name 'body' and the attribute 'name' with the value 'start'. Then there is a bit of text separated by a tag with name 'br' which closes itself and some more text. Finally there is the closing tag for 'body.'

```
<body name=”start”>
    Some Text
    <br />
    More Text
</body>
```

The DOM-Tree or ’Document Object Model’ is a data tree, which is built using the XML elements and represents their hierarchy. The tree for the XML example above would have 2 nodes. The root node would be ’body’ which has 1 child named ’br’.

HTML is short for ’HyperText Markup Language,’ which is a language used in web pages. XHTML is a stricter version if this language which conforms to XML standards. Unfortunately a lot of web-pages do not conform completely to the XHTML standard which makes cleanup of the pages necessary.

2.2.5 GreaseMonkey

GreaseMonkey is an add-on [14] for Firefox[13], which allows users customize how web pages are displayed by using Javascript in Firefox. Besides common JavaScript Appliccation Programming Interfaces or APIs, GreaseMonkey also supports a few special APIs for some purposes. For instance, there is an API to prevent default registered events.

Chapter 3

Preliminary Research and Our Resarch Tools

In this chapter we present the tools that we made and our initial research. This formed the basis for our research into automatic data extraction from discussion boards.

3.1 Initial Research

We decided early in the project that we would use Naive Bayes to score every node in the DOM-Tree, for the likelihood of it being one of the data fields that we were searching for. The next step would be to use the Naive Bayes results in a structure classification where desired data would be identified and related elements connected. To be able accomplish this, we needed to increase our knowledge about what discussion boards looked like in XHTML code. We therefore decided to make a research platform that could later be expanded into a testing platform and simple prototype.

Our analysis of on-line discussion boards required pages that we could gather data from. For consistency reasons we built sets of test pages by downloading offline copies of the pages from selected discussion boards. This made us immune to changes that may occur in the pages after we have collected them.

At the start of the thesis we had some assumptions about the nodes which contained desired data. We expected author and date+time nodes to be leaf nodes or close to leaf nodes, while the content nodes would have a large sub-tree. The author and date+time were assumed to have all the text in the element node or in it's direct children. For the content, we expected the text to be found across many nodes which that were children, children's children and further down in it's sub-tree. Throughout this thesis we refer to the nodes in an element's sub-tree as sub-nodes.

We needed to confirm our assumptions about the XHTML code behind the discussion board pages, before we could select features for the Naive Bayes and make the structure classification. This was accomplished by gathering a total of 24 pages from 12 different discussion boards. We named these pages 'Page Set 1.' The location of the pages can be found in Appendix A.1. We then analyzed 1 page from each of the discussion boards in the page set. The number of sub-nodes for each data type, in each page is listed in Table 3.1. During our analysis we found that date+time nodes would sometimes have a few sub-nodes and in some discussion boards they even contained additional text. Author was always leaf with no additional text, while content varied a lot in the size of it's sub-tree within the same page. The text in the content was spread across nodes in it's sub-tree, except in the few cases where content was a leaf node.

Sub-node Count for Data Nodes			
Page Name	Time+Date	Author	Content
Page 1	3	0	1-142
Page 3	0	0	0-18
Page 5	0	0	2-34
Page 7	6	0	9-30
Page 9	1	0	3-40
Page 11	4	0	0-17
Page 14	0	0	1-40
Page 15	2	0	0-34
Page 17	2	0	0-84
Page 19	0	0	10-43
Page 21	2	0	0-84
Page 23	0	0	4-41

Table 3.1: Sub-node Count for Data Nodes

Our analysis was done on pages cleaned up with Tag Soup [6], which transforms the code behind the web page into valid XML. Differences between the original code and the cleaned up code may be introduced during cleanup. This means that it is necessary to always use the same

XHTML cleaner in order to keep the differences consistent.

3.2 Research Platform

We wanted to gather data on the features of the XHTML elements, which contains desired data. This required us to load the XHTML page's DOM-Tree into memory. The idea behind our research platform was to make a Java program which retrieves the DOM-Tree from cleaned up XHTML files. We would then analyse each node in the DOM-Tree to gather data, which could be used during our manual analysis of the discussion board pages. This data was then used to decide the features for our Naive Bayes classifier and the direction we wanted to take with our approach to structure classification. Our research platform was later developed into our prototype. In this sub-chapter we only give an outline of our research platform. A more engineering based description of our research platform can be found in Chapter 6.

3.2.1 Loading the Pages Into Memeory

We used Tag Soup [6] for the cleanup of the discussion board pages. This ensures that the XHTML code we analyse conforms to XML. The process has not been integrated into our research platform, instead we cleaned up the pages when we download them.

For the parsing of the XHTML pages we use the Xerces [7] XML parser, which gives us the DOM-Tree. We then copy selected information from the DOM-Tree into our own tree which keeps the original XHTML element structure intact. Our custom tree uses fields and classes for holding data in the nodes, which can later be used for analysis, classifier training and structure classification.

Each XHTML element in the DOM-Tree has a name field and a node map which contains the attributes. The text is contained in separate nodes from the element which it belongs to. We decided to make two text strings for each node in our custom tree. One string which only contains the text which is kept directly in the element that the nodes represents. The other string consists of the combined text from the node's sub-tree. The reason for this, was that we wanted to analyse both the text that was kept locally and the text that belonged to the node's entire sub-tree.

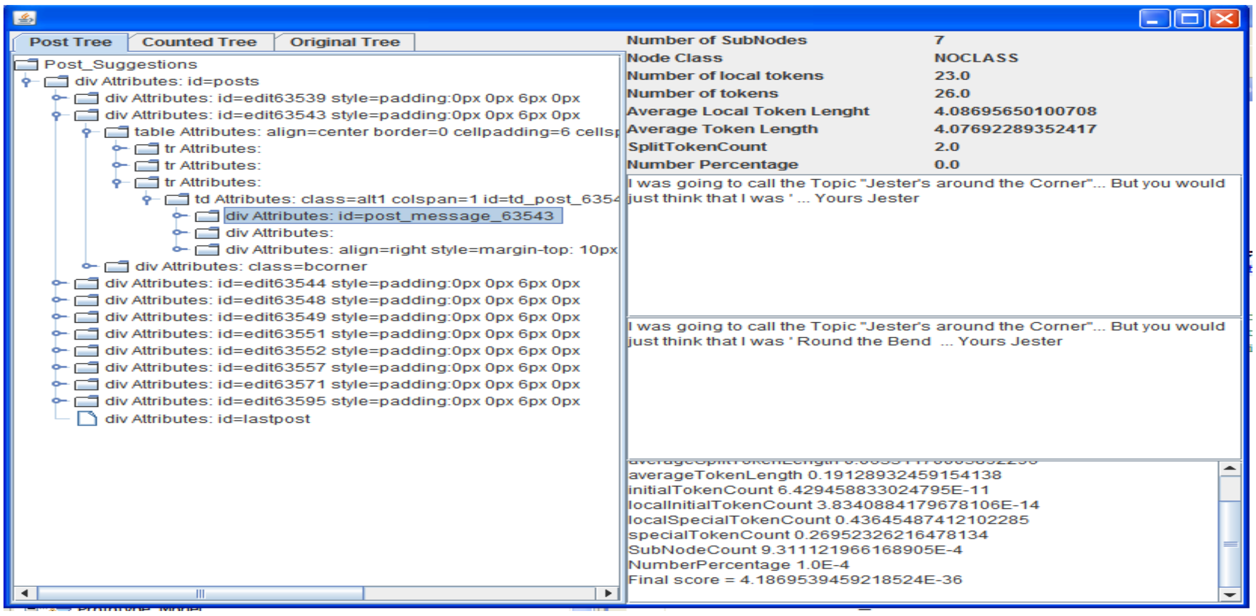
3.2.2 Retrieving Data From the DOM-Tree

The first thing we wanted data on, was the number of sub-nodes that belongs to each element that contains desired data. To calculate this, we walked the DOM-Tree depth first, by using a recursive method which walks until it finds a leaf node. This is done for all the children of each node. Once it hits a leaf node the current method call then ends, and it returns to the previous method along with giving a return value which is the number of children. This value is 0 when the node is a leaf. For a node which has 3 leafs and 1 node which has a child node that is a leaf, the direct child list would be 4 while there would be 1 node which has 1 child. The return values this node would get is 3 times 0 and 1 times 1. The 1 would be added to direct children which is 4, and you get the number of sub-nodes which is 5.

The algorithm for getting the text has two components. One which gets the local text and one which gets all the text in it's sub-tree. We needed to make sure we retrieved the text in correct sequential order. This is done while adding the XHTML element nodes into our custom tree. We build the text string for each element by running it's sub-tree depth first, with a recursive method which has return value type of string, until all paths has been exhausted. On the return the method returns the text that belongs to the nodes below the one that the method gathered text from. This allows us to make a text string that accurately represents the text as it is found in the DOM-Tree while stripping away all the XHTML element nodes from the text.

3.2.3 Tools for Analysing Nodes in the DOM-Tree

We decided to add algorithms that would analyse certain aspects of the elements in the DOM-Tree, in order for us to evaluate what we could use as features for Naive Bayes. The results are then displayed in the GUI or Graphical-User-Interface of the analysis tool. Our main focus with the features was on the text that belongs to the XHTML elements. We decided on several features which could work for the Naive Bayes classifier by implementing algorithms and displaying the results in our research tool. One of these algorithms counts the tokens in the local and the global text for an XHTML element. We consider the text that is found in a node's sub-tree to be global. This provides two of the Naive Bayes features. The complete list of features can be found in Chapter 4.1.3. We reviewed the results for each feature manually before we decided whether or not to use the feature in our Naive Bayes implementation.



```
Post_Suggestions
  div Attributes: id=posts
    div Attributes: id=edit63539 style=padding:0px 0px 6px 0px
    div Attributes: id=edit63543 style=padding:0px 0px 6px 0px
      table Attributes: align=center border=0 cellpadding=6 cellsp
        tr Attributes:
        tr Attributes:
        tr Attributes:
          td Attributes: class=alt1 colspan=1 id=td_post_6354
            div Attributes: id=post_message_63543
            div Attributes:
            div Attributes: align=right style=margin-top: 10px
      div Attributes: class=bcorner
    div Attributes: id=edit63544 style=padding:0px 0px 6px 0px
    div Attributes: id=edit63548 style=padding:0px 0px 6px 0px
    div Attributes: id=edit63549 style=padding:0px 0px 6px 0px
    div Attributes: id=edit63551 style=padding:0px 0px 6px 0px
    div Attributes: id=edit63552 style=padding:0px 0px 6px 0px
    div Attributes: id=edit63557 style=padding:0px 0px 6px 0px
    div Attributes: id=edit63571 style=padding:0px 0px 6px 0px
    div Attributes: id=edit63595 style=padding:0px 0px 6px 0px
    div Attributes: id=lastpost
```

Number of SubNodes	7
Node Class	NOCLASS
Number of local tokens	23.0
Number of tokens	26.0
Average Local Token Lenght	4.08695650100708
Average Token Length	4.07692289352417
SplitTokenCount	2.0
Number Percentage	0.0

```
I was going to call the Topic "Jester's around the Corner"... But you would
just think that I was ' ... Yours Jester
```

```
I was going to call the Topic "Jester's around the Corner"... But you would
just think that I was ' Round the Bend  ... Yours Jester
```

```
averageTokenLength 0.19128932459154138
initialTokenCount 6.429458833024795E-11
localInitialTokenCount 3.8340884179678106E-14
localSpecialTokenCount 0.43645487412102285
specialTokenCount 0.26952326216478134
SubNodeCount 9.311121966168905E-4
NumberPercentage 1.0E-4
Final score = 4.1869539459218524E-36
```

Figure 3.1: Our research tool

In Figure 3.1, you can see what our research tool looks like. The different trees that we create from the DOM-Tree are shown the left side. We have the original DOM-Tree which is what we get from the Xerces parser, our counted tree which contains copies of the XHTML nodes and a tree that contains a copy of a sub-tree from the counted tree. On the right side there are four parts. The top part displays some of the feature data for that specific node. The first text box contains the text that is local to that XHTML element, while the second text box shows all the text that is present in the sub-tree, which belongs to the selected XHTML element. The lowest text box on the right side contains the Naive Bayes score for all the features found in that specific XHTML node. Displaying the results for each node in this manner, allowed us to do manual analysis of the features and the results they produced. We also used a similar approach when we wanted to get new features.

3.2.4 Marking the XHTML Nodes

We decided to mark the nodes corresponding to the XHTML elements which contained desired data. This ensures that we know which nodes contained author, content and date+time. This was done through templates in XML format which described the DOM-Tree path to the nodes we wanted to mark. We then walked the tree paths down to the nodes, and labeled them with the specified class.

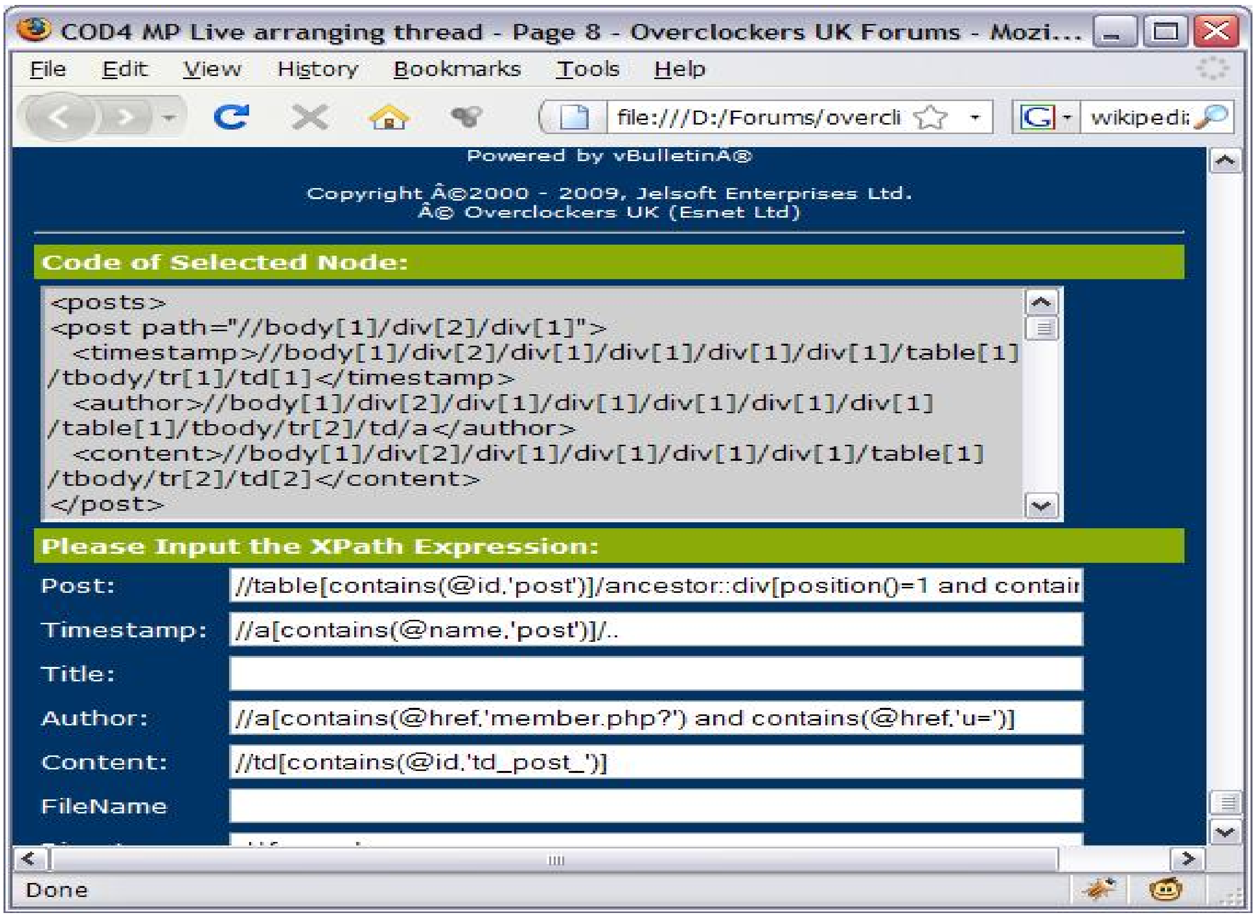
3.3 Templates for Marking the XHTML Elements

The Naive Bayes classifier works on statistical data on previously seen samples. This meant that we needed to make a set of pages for it to train on. We was decided that we would mark the nodes, which we wanted the Naive Bayes to gather training data from, with a set of paths in the DOM-Tree. To this end, we created a tool for generating templates which specified the nodes' path in the XHTML DOM-Tree. These templates was also used to get results in all of our experiments.

3.3.1 Tool for Generating Templates

A tool was designed to generate templates through the use of XPath expressions. Figure 3.2 shows the basic Graphical User Interface or GUI of the template generating tool.

First we input XPath expression for finding the post path into Firefox, which enables us to use the XPath functions. We then input the relative XPath of author, content and date+time based on the path of the post node. After click 'Generate' button, the tool generates the template in Firefox. For the post, the tool first executes the path expression and gets the nodes. Then the tool traverses all the parent node until the root node 'html' and composes the real XPath of the post node. Author path is generated by executing the XPath expression combining the post path and author expression. The same procedure is used for content and date+time as well. After generating the template, which is shown in the text area in the browser, we click 'Save' button to deliver the data to local server. The local server will write the template generated to a XML file.


COD4 MP Live arranging thread - Page 8 - Overclockers UK Forums - Mozi...
File Edit View History Bookmarks Tools Help
file:///D:/Forums/overcli
wikipedi
Powered by vBulletinÂ®
Copyright Â©2000 - 2009, Jelsoft Enterprises Ltd.
Â© Overclockers UK (Esnet Ltd)
Done

Code of Selected Node:

```
<posts>
<post path="//body[1]/div[2]/div[1]">
  <timestamp>//body[1]/div[2]/div[1]/div[1]/div[1]/div[1]/table[1]
/tbody/tr[1]/td[1]</timestamp>
  <author>//body[1]/div[2]/div[1]/div[1]/div[1]/div[1]/div[1]
/table[1]/tbody/tr[2]/td/a</author>
  <content>//body[1]/div[2]/div[1]/div[1]/div[1]/div[1]/table[1]
/tbody/tr[2]/td[2]</content>
</post>
```

Please Input the XPath Expression:

Post:	//table[contains(@id,'post')]/ancestor::div[position()=1 and contai
Timestamp:	//a[contains(@name,'post')]/..
Title:	
Author:	//a[contains(@href,'member.php?') and contains(@href,'u=')]
Content:	//td[contains(@id,'td_post_')]
FileName	

Figure 3.2: GUI of Templates Generating Tool

Chapter 4

The Data Extraction Algorithms and Classifiers

This chapter is about our algorithms and pattern classifier and. We give a detailed description about each part and the reasoning behind our decisions.

4.1 Naive Bayes

We chose to use Naive Bayes to score every node in the XHTML DOM-Tree for how likely it is to belong to a certain class. In order for us to be able to do this we needed a set of features that the Naive Bayes classifier could use to score how likelihood a node belonging to a certain class. We defined one class for each of the data fields that we are looking for.

4.1.1 Naive Bayes Training and Distribution Selection Process

The Naive Bayes classifier works by using statistics on how previously seen samples looks like. It therefore requires a training process to get the statistical data. This data defines how the likelihood for each feature, given a class, looks like. The first part of this process is to load the tree into memory using our research platform. We then load the templates that point to the nodes that belong to one of our chosen classes and mark the XHTML nodes in memory. The next step is to run our algorithms that analyses what the features look like for each node in the tree. The final steps in the training process is to print out .csv files with the statistical data. CSV stands for Comma-Separated-Values, which can be loaded up as a spreadsheet where columns are separated by a comma. The statistical data that is output can then be used to select statistical distributions for each of the features given a class. After the distribution selection process we calculate the parameters needed to set up a the distribution which will be used to evaluate the likelihood of unseen samples belonging to a certain class.

The statistical distributions were chosen by setting intervals for the values in the statistical data and counting the frequency in which the values appear in each interval. The frequencies are then used to plot graphs. We selected the distributions for each feature when given a class, by comparing the graphs with existing methods of statistical distribution.

4.1.2 Naive Bayes Classes

We assigned a class for each desired data type, which gives us the following classes:

- Author
- Date+Time
- Content

4.1.3 Features and Their Distributions

Here we present the features we started out with before algorithm adjustments. With the exception of the sub-node count, all of our features work on the text. The sub-node count and our percentage of numbers features, are the only ones with only a global component. The rest have a local component which works on the text in the node that is being evaluated and a global component that evaluates all the text that is found the sub-tree of the node which is being analysed. Each feature has a small description of what it does and the reasoning behind it. Throughout this sub-chapter we refer to date+time nodes as time.

- **Average Token Length**

  The Average Token Length feature consists of taking all the tokens split by whitespace and finding their average length. It has a local and a global component.

  Getting the average token length serves two purposes. The text in time and date nodes will have similar length across different discussion boards while author will vary a lot. By getting the average length of an author's name we get indications how likely a single token is to be to be an author when combined with the Token Count for the node that is being evaluated.

  The distribution used for both the local and global component uses Normal Distribution.

- **Average Split Token Length**

  This feature is the average length after we have split the text string by a series of special tokens, in addition to whitespace. The special tokens consists of ':', '.', '\', '/' and '-'. The feature has a local and a global component.

  We chose this feature to better distinguish author, date and time. From our observations during manual analysis of the XHTML nodes we found that the author nodes would rarely be split into more than one token by the special tokens we chose. Date and time would however be split into several small tokens, since date and time is often written in a format similar to: '03.22.09 23:59:59.' The average split token length here would be 2 compared to the average token length which would be 8.

  We chose Normal Distribution for both the local and global component of this feature.

- **Initial Token Count**

  The number of tokens in the text when split by whitespace. It has a local and a global component.

  From our research into the XHTML structure behind the discussion boards, we found that the node owning the main content of a discussion board posting has most of the text in it's sub-nodes. It's local token count will therefore often be quite low while the global token count is usually high. Author on the other hand will only have one token most of the time, though there are a few exceptions.

Both the global and local component of this feature uses Normal distribution.

- **Percentage of Numbers**

  Gets the percentage of numbers in the text, while discounting whitespace and our selected special tokens.

  After running a few experiments with our original features we found that our date+time class would not always be among the highest scored nodes for it's class. There would be other nodes which had a similar 'special token count' to our date+time nodes. The majority of the nodes which were scored highly as data+time, while being something else, had a low amount of numbers in them. We therefore decided to add the percentage of numbers as a feature, since date is usually written in a number format and time is rarely written without numbers.

  Author and content use the Poisson distribution, while the time class uses a special distribution. Since this feature uses a percentage value, the number will be between 0 and 100. When making the special distribution for time, we split this are into gaps of 2. That gives us 50 areas where we count the occurrences of time features during training. Due to the nature of patterns in time nodes, we get a lot of gaps with frequency of 0. We solve this problem by creating a linear curve between each non-zero gap and then use a sliding window algorithm to decrease the amplitudes of the curves gaps that are close to each other. This curve is used directly as a look up table for our time node likelihood.

- **Special Token Count**

  Counts the number of special tokens which we split the text strings by. The special tokens are ':', '.', '\', '/' and '-'. The feature has a local and a global component.

  This feature was primarily introduced to separate time nodes from other nodes. The count should remain rather stable for time, while it will fluctuate a lot in large texts. In author the special tokens should be close to non-existent.

After looking at the initial training results, we went with a Poisson distribution for both the local and global component of this feature.

- **Sub-Node Count**

  The number of nodes that are found in the sub-tree of the node which is being analysed.

  Introduced because author is in almost all cases leaf nodes. The time nodes will either be leaf nodes or have a small number of sub-nodes while the content node will contain a large number of nodes below it.

  The Poisson distribution is used for all of our classes in this feature.

4.2 Structure Analysis

The discussion board pages are well structured, since the post data-records are recurring and have similar internal structure for author, content and date+time. We are using structure analysis classification in our approach to find the desired data elements. We combine the structure classification with Naive Bayes and grammar in order to give our structure classification algorithms more support in it's decision making.

We used some terms related to structure analysis. In the DOM-Tree, a node index is the position in the same type of sibling nodes. The first position is 1 instead of 0. For example in the XHTML below, the index of the node *"<div>c</div>"* is 2.

```
<td>
    <div>a</div>
    <span>b</span>
    <div>c</div>
</td>
```

We use term 'group' for the nodes with paths where only a certain index differs. A date+time group is a group whose date+time nodes have same path with only one node index difference. This definition applies to author and content as well. A sub-tree which contains one author node, one content node and one date+time node, is considered a complete post. We use the term 'relative path' for the path of a node under a post, which starts from the post node to the node itself. The term 'equivalent sub-nodes' is used for nodes which have the same text and relative path in several of post sub-trees.

In order to make the structure classification work, we need some features which can be used for analyzing the structure. We came up with some general features which are applied to all classes and some that are class specific. Relative path is one of our general features for author, content and date+time. All the author nodes should have the same relative path in the same page. This applies to the content and date+time nodes as well. A feature for author nodes is that it should be a leaf node, while the amount of equivalent nodes in the sub-tree is a feature for content nodes.

Some of the structure features are strict. For instance, the relative path is strict for date+time nodes. Some features are not strict but works like pointers to improve the classification result. For instance, if there are more sub-nodes than others in a node, the node will more likely be a content. We use the strict features to do the first classification step, where we eliminate candidates which we deem unlikely to be of the class we are searching for. The non-strict features are then used to sort the remaining candidates. In this step, we combine the score for the structure features with Naive Bayes. In date+time and content classification there is a formula which is used to calculate the those scores. We use the term 'Date+time factor' and 'Content factor' for the formulas.

4.2.1 Structure Features

The general feature of date+time, author and content is relative path. For date+time nodes, they should have same path except one index that differs for each ones. Author and content nodes should have same relative paths. But there are some other features for different classes. Date+time has another feature which is the number of date+time nodes. Author has two additional features. The first is that author should be the leaf nodes. The second is that authors should not have the same text. Content has three additional features. The first is the equivalent sub-nodes count. The second is the number of sub-nodes. The third is that content shouldn't contain date+time and author.

Date+Time

The path of date+time is the feature that is used to find the path to the node which is parent to all the postings. The only difference in the path of the date+time nodes is one of the indexes. For instance, nodes with the following paths fit for this feature:

*//body[1]/div[1]/div[9]/***table[1]***/tbody[1]/tr[1]/td[2]/div[1]/span[1]*

*//body[1]/div[1]/div[9]/***table[2]***/tbody[1]/tr[1]/td[2]/div[1]/span[1]*

*//body[1]/div[1]/div[9]/***table[3]***/tbody[1]/tr[1]/td[2]/div[1]/span[1]*

The post nodes will be:

*//body[1]/div[1]/div[9]/***table[1]**

*//body[1]/div[1]/div[9]/***table[2]**

*//body[1]/div[1]/div[9]/***table[3]**

The differing index has been marked in bold. This feature is general in discussion boards. We select it as a main feature of date+time in structure classification. The feature is strict and can be used in selecting possible date+time nodes. We then use this as the first feature of date+time classification.

The number of date+time nodes in a date+time group is another feature for this class. We consider that if a date+time group has more date+time nodes than other possible groups, it is more like to be a correctly identified group. In the example group shown above, there are 3 date+time nodes, but if there is a group with 5 date+time nodes, we would consider the second group to be the most likely date+time group. We combine the number of date+time nodes and Naive Bayes score in an algorithm which is described in Chapter 4.2.2.

Author

We have a few requirements for author nodes. All the authors must be in the sub-tree of the post nodes, so the path strings of authors should start with the post node path. They should also be leaf nodes. Nodes in different post nodes which have the same relative path are can be selected as possible author nodes. For instance, if there are two leaf nodes whose paths are:

*//body[1]/div[1]/div[9]/***table[1]***/tbody[1]/tr[1]/td[1]/div[1]/span[1]/a[1]*
*//body[1]/div[1]/div[9]/***table[2]***/tbody[1]/tr[1]/td[1]/div[1]/span[1]/a[1]*

The post nodes should be:
*//body[1]/div[1]/div[9]/***table[1]**
*//body[1]/div[1]/div[9]/***table[2]**

The relative path would be:
/tr[1]/td[1]/div[1]/span[1]/a[1].

Since the leaf nodes have the same relative path, we would consider the two nodes to be possible author nodes. We group author nodes by the same relative path. If all of the author nodes in a group has the same content, then we will exclude that group of nodes from being of the type author. This is because there will be several instances of leaf nodes which have 1 word which is fixed for all nodes with that specific relative path. An instance of this could be a hyper-link with the text 'option,' which would fit very well with our classifier, but is definitely not an author name if it appears 15 times in a page with 15 posts. Only exception to this would be if someone was the only person to post in the page, but this very rarely happens.

Content

Like the author nodes, the nodes containing content have the same relative path in the post nodes' sub-trees, but the content nodes are not necessarily leaf nodes. The content nodes should not have identical sub-trees. In a group of content nodes, there should be at least one that has a sub-tree which is different. If they have same nodes, then they might be signatures, nodes containing author names or similar.

A group of content nodes should not contain too many equivalent sub-nodes. If it does contain a lot of equivalent sub-nodes, we consider the nodes to be of another type. It could be nodes containing author, author's title, register date, etc. Basically, the less equivalent sub-nodes a node has, the more likely it's a content. But in this case it's easy to classify the leaf nodes as contents because they don't have sub-node. To solve this problem, we add the number of sub-nodes as another feature. The more sub-nodes a node has, the more likely it is a content node. We made an algorithm which works on these two features combined with the Naive Bayes score in the prototype, which is described in Chapter 4.2.2.

Another feature is that the author and date+time nodes should never reside within content. If we do find one or both of these nodes inside the sub-tree of a possible content node, we conclude that it is not a content node.

4.2.2 Structure Classification Algorithm

There are 4 steps in the structure classification. The first step is date+time classification. The date+time is classified based on Naive Bayes score and structure features of date+time. Then the path of the post nodes can be identified based on date+time nodes path. The next step is to classify the author nodes based on the paths of the post nodes and the structure features for author. The final step is to classify the content nodes. Every step has two parts, selecting nodes groups with the same relative path nodes and sorting groups. The grammar rules, which are described in next section, are used during the node selecting.

Date+Time

The selecting of date+time nodes starts with a comparison of the nodes' path. First we compare the paths of all the nodes in the DOM-Tree. The paths which have one node index difference will be selected as groups. After that comparison, we can get several groups of nodes based on the path.

Then we filter out the nodes whose text can not be verified by date+time grammar. For instance, if the length of the text is smaller than 2, it is not a date+time node.

We then sort the date+time groups by Naive Bayes score and numbers of date time nodes.

After the Naive Bayes process for date+time, we get a list for every node in the the order of Naive Bayes score. In every date+time group we calculate the average position of the date+time nodes in the Naive Bayes list. For instance, there is a date+time group whose nodes text are *"2009-05-30 12:00"* and *"2009-03-10 11:05"*, which rank 2 and 3 in the score list. There is another date+time group whose node text are *"send topic 10983 to "* and *"send topic 12383 to "*, which rank 30 and 40 in the list. Then the average position of first group is 2.5, and the second is 35. Then the first group is more likely to be the date+time group we are searching for. We mark value of the average position as A. We also calculate the number of date+time nodes in a date+time group. We mark value of the number as B.

The group with the smaller A, is more likely to be the group is the one we are searching for. We also prefer if the group has a larger B. We use the following equation to calculate which group is the better match for date+time *"date+time score=A+'date+time factor'/B"*, where the *date+time factor* is a fixed factor which influences the final score.

After the sorting, the highest scoring date+time group which has lowest 'date+time score' will be considered the correct date+time group. We then compare the paths and consider the first part up until, and including the varying index, to be the paths to the posts. In addition, we sort the post nodes in a group by the differing index in ascending.

Author

The author classification is done after the post nodes have been found. The procedure is similar to the one that we use for date+time. First we select the possible nodes which are leaf nodes and have same relative path and filter out unwanted nodes using grammar. The remaining nodes are then placed in groups based on their paths and sorted by using the Naive Bayes score. The next step is to filter out all the author groups which has nodes with identical text. The final step is to get the average position of each group's nodes in a list that is sorted in descending order, based on Naive Bayes score. The group with the highest average position will then be our predicted author nodes.

Content

As with author and date+time, we select nodes with the same relative paths and group them together. However the procedure differs slightly, as we want nodes which has different sized subtrees, instead of equal size which we use for our other two classes. We then sort them by Naive Bayes score and equivalent sub-nodes count and total sub-node count.

The predicted author and date+time nodes should not reside within the content nodes' subtrees. If the node path contains the path of an author or date+time node, then the node will be excluded as possible content. Then we filter out the content groups whose content nodes have the same number of sub-nodes.

Then, like we did for author, we get a list that is ordered by the Naive Bayes score for content. The next step is to calculate the average position(A) of a content group in the same algorithm of date+time and author.

We set a property for the node, which is the size of it's sub-tree. In every content group we can easily calculate the average size of the sub-tree. This is the average total number(C). We also compare the content node in a content groups and get how many equivalent nodes(B) a content group have. Groups with low values for A and B and large values for C are more likely to be correct content groups. However if B is 0, there is a chance that all the nodes in the group are leaf nodes. We use the equality *"content score=A+'content factor'*(B+1)/C"*, where the *content factor* is a fixed number which influences the final score based on the experience.

4.3 Grammar

Grammar is useful when making rules for filtering some special characters. We used basic grammar in our experiments. The grammar is used when selecting nodes in the structure classification process.

The grammar defines rules for author and date+time. In addition, there is a rule for the connection between author, content and date+time. We use EBNF-style expression to describe the grammars. While it may not conform 100% to EBNF, the style itself is similar. Below we don't have enough space to list all the characters, so we use two variable to express any one character and any one character excluding colon:

character: any single character

characterExcludingColon: any one character excluding ':'

4.3.1 Author

The length of an author should be more than 2.

s=character,character;
s2=(character,s2) | s | (s2,character);
author=s2;

There shouldn't be character ':' in an author.

s=(characterExcludingColon,s) | s | (s,characterExcludingColon);
author=s;

4.3.2 Date+Time

The length of date+time should be equal to or larger than 2.

character;
s=character,character;
s2=(character,s2) | s | (s2,character);
dateTime=s2;

The length of digit characters should equal to or larger than 4.The date time should contain at least year,one month digit and one date digit.

digit='0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
s=(character,s) | s | digit | (s,character);
s2=s,s,s,s;
dateTime=s2;

The text should contains a string like ’,06’, meaning there is a special character before a two digit.

digit=’0’ | ’1’ | ’2’ | ’3’ | ’4’ | ’5’ | ’6’ | ’7’ | ’8’ | ’9’;
specialcharacter=’,’ | ’/’;
s1=specialcharacter,digit,digit;
s2=(character,s2) | s1 | (s2,character);
dateTime=s2;

There should be at least one date+time in the a date+time group whose digit length is larger 5.It means the time can be like ‘05-5-06’ or ’5-10-06’ or ’05-06-2006’ or ’jan,05,2006’.

digit=’0’ | ’1’ | ’2’ | ’3’ | ’4’ | ’5’ | ’6’ | ’7’ | ’8’ | ’9’;
s=(character,s) | s | digit | (s,character);
s2=s,s,s,s,s;
dateTime=s2;

4.3.3 Author, Content and Date+Time

A content node should not contain date+time node and author node.

dateTimePath;
authorPath;
tempPath=dateTimePath | authorPath;
contentPath;
others=(character,others) | character | (character,others);
contentPath=other,tempPath,others;

4.4 Relation to Previous Work

We primarily base our structure classification principles on the work done in [3] which builds on [2]. We also use principles some from [4]. Our approach combines the approaches used in [4], [3] to form our own structure classification algorithm. We use the equivalence class principle from [3] where we compare the structure that belongs to a several nodes with similar path. In order to build complete posts, we realign parts of the original tree, while making sure that the data stays under data-record which it is a part of. Our method differs slightly from the one used in [4], since we do not keep the hierarchy intact during our analysis. The reason for this, is that we will extract only parts of each data record through identifying the path to the data that we want to extract.

Our approach uses grammar and Naive Bayes, which is where it differs from previous work. While there occurrences of grammar in previous work, we have not seen Naive Bayes used as a helper for the structure classification for identifying specific data elements. We do believe our of combining the use of Naive Bayes, grammar and structure classification to extract specific parts of data records, has not been used before in the context of data extraction from web pages.

Chapter 5

Research Experiments and Results

Here we give a detailed description of our experiments. We also give our interpretation of the results.

5.1 Naive Bayes Evaluation

We did several experiments to evaluate the Naive Bayes features by themselves and the effectiveness of our classifier in finding the desired data nodes. The first part was to analyse the confusion between the different classes for each of the features. Based on our analysis, we made adjustments to the Naive Bayes and evaluated the performance of our Naive Bayes in full DOM-Trees, before and after the adjustments.

5.1.1 Class and Feature Confusion

The confusion matrix is a tool, that can be used to visualize the crossover between the different features and classes. This can be helpful in finding weaknesses in the classifiers features. The numbers in the columns represent the actual number of nodes that belong to a given class. The rows shows which class the feature evaluation placed the nodes in. In a 3x3 matrix, which is what we use, the ideal placement of all nodes would be in square (1,1), (2,2) and (3,3). We have author as the first row and column, content in the second row and column while time+date resides in the third row and column. If there are a number of nodes in the first column's second and third row, then nodes of class author has been falsely classified as content and time respectively. For the second column, nodes in the first and the third row are errors. For the third column, rows one and two contains wrongly placed nodes.

We did two different experiments that were related to a confusion matrix. The first was a traditional confusion matrix where we examine which class has the highest score for a certain feature. In this experiment we compare the likelihoods for a feature for each of the classes. The second experiment was done by sorting the nodes by their score for each feature, given a class. We then took the nodes that we expected to belong to the given class and checked if they were false or true positives. The experiment was performed for each of our classes. This was done, since even though all of our classes uses the same features, they have different distributions types on certain features. The second experiment is not a true confusion matrix, but is based on the same principles.

For the confusion experiments we selected 8 discussion boards from our page set and took 1 page from each. This gave us 8 pages that combined contained 135 discussion board posts with author, content and date+time. The total count of nodes, that belonged to our classes, was 405. The pages where then loaded into our research platform and run through algorithms that calculated our results. The distributions were achieved by training on 1827 discussion board postings. The location of the page set used in this experiement can be found in Appendix A.2.

Traditional Confusion Matrix

Our experiment with the traditional confusion matrix was done by retrieving the Naive Bayes scores for a feature when given the different classes from each node. Our next step was to compare the scores, which gives the likelihood for the feature belonging to a certain class. The feature is then placed as predicted to be of the class with the highest likelihood for the feature and is placed in the corresponding row. Finally we check which class the node really belongs to and this is used to select which column the node goes into. The process is repeated for all of our features and for the total Naive Bayes score.

For a feature that shows the highest likelihood for being author goes into the row for author. If it is really a date+time node then it goes into the column for time, and will primarily be referred to as time throughout this sub-chapter. The tables contains 3 columns and each column contains 135 nodes where the column gives the real node type, while the row gives the predicted node type based on the feature likelihoods for the different classes. The confusion matrix was done for all 10 features in addition to the Final Score from the multiplication of all the features. The Final Score is the Naive Bayes classification of the node. A thorough description of our features can be found in Chapter 4.1

In addition to the confusion matrix, we show the distributions that were used, and comment on the values. Our analysis of the confusion matrix results is presented after the confusion matrix itself. The analysis consists of finding reasons for why the feature works or does not work. We will make a short introduction of the distribution numbers and distribution type before the confusion matrix.

Average Local Split Token Length

The average local split token length uses a normal distribution for calculating the likelihoods. There is a healthy difference between the averages and the normal deviation is not large enough that there should be a lot of crossover between the classes.

	Distribution		
	Author	Content	Time
Average	7.13	0.79	2.72
Deviation	2.21	1.27	0.33

Table 5.1: Average Local Split Token Length Distribution

For the average local split token length, we see that author was miss-classified as time 12 times while only 1 time as content. Content is predicted as author 15 times and 7 times as time, while time is classified perfectly. The wrong prediction of classes happens because the length of the tokens after the splitting, is closer to what you would expect from a different class. For the wrongly classified authors that are classified as time are either very short user-names or are split by

Confusion Matrix				
		Actual:		
		Author	Content	Time
Predicted:	Author	122	15	0
	Content	1	113	0
	Time	12	7	135

Table 5.2: Matrix - Average Local Split Token Length

one of our selected characters. The one author that is classified as content has split content lengths that is abnormally short. A lot of the content nodes have no local text, which gives a low average split lengths since the total local token length is those cases is 0. The content that is miss-classified as time has very short words, while those predicted as authors have long words. Overall the feature distinguishes quite well between the classes and is very good for time nodes.

Average Local Token Length

With the average local token length, we see that author and time are rather close, with a standard deviation that will allow for confusion between the classes. Content's average is quite a bit lower, but has a large standard deviation which can lead to some confusion. This feature uses a normal distribution.

Distribution			
	Author	Content	Time
Average	7.35	3.02	5.19
Deviation	2.20	4.22	1.78

Table 5.3: Average Local Token Length Distribution

For this feature we get a lot of confusion for the authors. Almost half the authors are falsely classified as time nodes. Content and time has some confusion with 22 content nodes being classified as time and 30 time nodes being predicted as authors. The confusion comes from the standard deviation for author being large enough, that when subtracting it from the average value, we end up in the highest scoring range of time. Similar problems happens with content and time as well,

Confusion Matrix				
		Actual:		
		Author	Content	Time
Predicted:	Author	69	0	30
	Content	5	113	0
	Time	61	22	105

Table 5.4: Matrix - Average Local Token Length

but it does not have as large impact. Due to the confusing nature for this feature, when it comes to author, it should be considered if it is worth keeping it.

Average Split Token Length

From the distribution numbers, we see that there is little crossover between the distributions. The most likely to have some wrongly classified nodes is author, since the standard deviation is fairly large. Time and content has very low deviation which gives a low likelihood, when the feature of a node deviates a bit from the average value. This feature uses normal distribution for the likelihood.

Distribution			
	Author	Content	Time
Average	7.13	4.16	2.72
Deviation	2.21	0.47	0.34

Table 5.5: Average Split Token Length Distribution

The confusion matrix shows that author has 24 nodes predicted as content and 13 predicted as time. By looking at the distributions, it seems likely that the wrongly classified nodes are either short author names or contains one or more of the characters that we split the tokens by. The content that is classified as author is likely to have quite a few long words and an average split token length that falls on the upper side of the average+deviation of the content class. This moves it close or inside the highest likelihood for area for author. There is a high probability that the content that is classified as time, contains a lot of short words, since the gap between the classes

Confusion Matrix				
		Actual:		
		Author	Content	Time
	Author	98	10	0
Predicted:	Content	24	121	1
	Time	13	4	134

Table 5.6: Matrix - Average Split Token Length

is rather big and their deviation values are small. The 1 time node that was miss-classified, most likely either contains quite a bit of text in addition to date and time or has parts of date and time written as text. There is some confusion overall, primarily for author. We do believe this confusion is low enough that the feature will contribute in separating the classes.

Average Token Length

This feature is the global component of measuring the token length. The average values of content and time classes are quite close. The deviation for content is a lot smaller, so there is a good chance that nodes that has a feature value close to the time average will be classified as content. Author overlaps slightly with the top of time, so there is a good chance that features with length around time average will be classified as content, then a small gap where they are classified as time and then author takes over as the most likely class. On the lower side of content, time will have the largest likelihood again. This feature uses normal distribution.

Distribution			
	Author	Content	Time
Average	7.36	4.81	5.17
Deviation	2.20	0.71	1.8

Table 5.7: Average Token Length Distribution

Again the shorter author names are classified as content or time. Content nodes that contains words that are shorter or slightly longer than the sweet spot for the content likelihood, are falsely classified as time while content with very long words are predicted to belong to author. The time

Confusion Matrix				
			Actual:	
		Author	Content	Time
	Author	74	6	30
Predicted:	Content	25	114	17
	Time	36	15	88

Table 5.8: Matrix - Average Token Length

nodes that is predicted as author are likely to contain long time formats or words, while the ones that are classified as content has slightly tokens shorter than the average time node. Both author and time have quite a bit of confusion. The main problem with this feature is the crossover between the different classes' distribution.

Initial Token Count

The distributions for each of the classes have a good amount of separation for this feature. Wrongly predicted nodes will be a result of a large deviation from the norm for it's class, rather than the classes overlapping. This feature uses normal distribution.

Distribution			
	Author	Content	Time
Average	1.1	64.17	5.7
Deviation	0.18	48.38	3.12

Table 5.9: Initial Token Count Distribution

The results of the confusion matrix experiment shows that a few authors are wrongly classified. The explanation for this is that some authors contain two or more tokens, while the bell curve for the class' normal distribution is rather steep and the time class has a rather large deviation which makes the bell curve wide enough to give higher likelihood than author's when the feature has a two or more tokens. It is important to keep in mind that the feature only exists of integers, so 1,2,3... are the only type of values possible. The content that is classified as time, are very short posts while the 1 node that is classified as author must consist of one word. After looking through

Confusion Matrix				
		Actual:		
		Author	Content	Time
	Author	123	1	10
Predicted:	Content	0	116	3
	Time	12	18	122

Table 5.10: Matrix - Initial Token Count

the message boards in the confusion experiment, we conclude that the time class nodes that are classified as author only contains date written in the format MM-DD-YYYY. Since there is only the date part of date+time, there is only one token. The time nodes that are classified as content, most likely contains a lot of other text in the same node as time or in the sub-tree that belongs to the class time. The overall results show that this feature is worth keeping.

Local Initial Token Count

For this feature the only possible count for author probability to be larger is 1 token. At token counts 0 and 2-9 the time distribution has the highest probability. If the node has 10 or more local tokens, the content distribution will have the highest likelihood. This can cause some problems if the content node has very little text in it's top node and Authors that have more than 1 token will be miss-classified as time. Since the author is a leaf in all but 1 case in our total page set, it has the same distribution as in the initial token count feature. The distribution used for this feature is normal distribution.

Distribution			
	Author	Content	Time
Average	1.1	9.97	5.11
Deviation	0.18	16.06	2.39

Table 5.11: Local Initial Token Count Distribution

The results show 12 authors classified as time, which means that there are 12 authors with more than 1 token. The content prediction gives some surprising results. There appears to be 11 posts

Confusion Matrix				
			Actual:	
		Author	Content	Time
	Author	123	11	10
Predicted:	Content	0	124	15
	Time	12	0	110

Table 5.12: Matrix - Local Initial Token Count

with only one token in the top node for the content while the rest all have 10 or more tokens. The 10 time nodes that are falsely predicted to be authors are the ones who have only a date in the format of MM-DD-YYYY. The 15 time nodes that are miss-classified as content, most likely come from the same discussion board and contains a lot of other tokens than just date and time. The feature will help us in the determining the different classes, though it is a bit worrying that authors with more than 1 token gets classified as time class several times. This can move the overall Naive Bayes likelihood toward the author nodes being classified as time.

Local Special Token Count

The likelihood for this feature is calculated using Poisson distribution. This means that the average value is the only number used.

Distribution			
	Author	Content	Time
Average	0.07	0.6	2.07
Deviation	0.12	1.04	0.9

Table 5.13: Local Special Token Count Distribution

This feature works well for author but is very confused when it comes to content and time. The confusion matrix results, suggest that this feature should not be used for either of content and date+time.

Local Special Token Count				
			Actual:	
		Author	Content	Time
	Author	133	116	0
Predicted:	Content	1	8	65
	Time	1	11	70

Table 5.14: Matrix - Local Special Token Count

Percentage of Numbers

This feature was made specifically for the time class, so the average and deviation for time is never used. The time distribution is made by making 50 windows that cover 100%, meaning each window covers 2%. We then count the frequency of a percentage for each window. Since several of the windows will have no occurrences, we make a linear graph between each of the non-zero data points and then smooth this graph out to remove extreme values and give a more consistent likelihood. The author and content likelihoods use Poisson distribution.

Distribution			
	Author	Content	Time
Average	7.72	2.52	60.64
Deviation	11.84	2.92	26.22

Table 5.15: Average Local Split Token Length Distribution

Percentage of Numbers				
			Actual:	
		Author	Content	Time
	Author	4	24	0
Predicted:	Content	112	102	0
	Time	19	9	135

Table 5.16: Matrix - Percentage of Numbers

The results of our confusion matrix experiment shows that the feature works excellent for time nodes, and decently for content. The classification of author however, is a disaster. This is most likely due to a few author names containing several numbers while most author names do not. The distributions indicate that the authors that are classified as content, do not contain any numbers at all. The 19 authors that are classified as time, contains quite a few numbers and are most likely the ones that drives up the average and deviation values for the distribution. After analysing the results for this feature, we conclude that it should not be used for author as it just confuses the classifier. The feature does however work great at what it was designed for, distinguishing time nodes.

Special Token Count

This feature uses the Poisson distribution for all the classes. This means only the average value is used. There shouldn't be too much crossover between the classes.

	Distribution		
	Author	Content	Time
Average	0.07	6.01	2.15
Deviation	0.12	5.45	0.91

Table 5.17: Special Token Count Distribution

		Special Token Count		
			Actual:	
		Author	Content	Time
	Author	133	17	0
Predicted:	Content	0	75	21
	Time	2	43	114

Table 5.18: Matrix - Special Token Count

This feature works very well for author with just 2 wrong predictions. It does have problems distinguishing content with only 75 out of 135 nodes correctly classified. The rest spreads among time and author which both have a higher likelihood when the token count is low. This shows that the feature is not that well suited for distinguishing content, but works well for the author and time classes.

Sub-Node Count

This is our only feature that works on the structure of the DOM-Tree. It uses the Poisson distribution, and all the classes have a good amount of separation. The only reason the distribution for author is not 0 on both average and deviation, is because there is 1 author node among more than 1950 in our page set that has 1 sub-node. Content often has a lot of sub-nodes since bold text and italic text each adds 1 sub node.

Distribution			
	Author	Content	Time
Average	0.00054	14.25	1.56
Deviation	0.0011	11.45	0.83

Table 5.19: Sub-Node Count Distribution

Sub-Node Count				
			Actual:	
		Author	Content	Time
	Author	135	3	20
Predicted:	Content	0	89	0
	Time	0	43	115

Table 5.20: Matrix - Sub-Node Count

The feature works perfectly for author, while time has 20 nodes which is classified as author. This means there is only 20 time nodes in the confusion page set has date and time in the same node. The 3 content nodes that are classified as author contains the entire content in 1 node, while the content nodes that are predicted to be time contains 1,2,3,4 or 5 sub-nodes. These would be discussion board postings where the content contains little formatting. We are fairly happy with the results for this feature, though content could be better.

Total Naive Bayes Class Score

This is the complete Naive Bayes score which shows the ability for our complete classifier to distinguish between the classes.

Total Class Score				
		Actual:		
		Author	Content	Time
Predicted:	Author	120	0	0
	Content	1	132	0
	Time	14	3	135

Table 5.21: Matrix - Total Naive Bayes Class Score

The classifier has some problems with severals authors being predicted to be of time class. We believe this is mainly due to the emphasis our classifier puts on token counting, and that 12 of the 14 nodes that were miss-classified are the same 12 that showed up in the local token count and token count as time nodes. Except for these, 1 author is classified as content and 2 others are predicted to belong to time class. For content, 3 nodes are classified as time. These are most likely short posts with a few words and few or no sub-nodes. For time our classifier was perfect with 135 out of 135 nodes correctly classified.

Modified Confusion Matrix

The normal confusion matrix decides on how many nodes are false positives and how many are true positives. It works by analysing all the items that the experiment wants to find the confusion for. The predicted nodes are sorted by the actual class and the class it is predicted to be. This experiment is based on the traditional confusion matrix, but instead of comparing the score for the features when given the different classes, we sort the nodes by score when given a specific class. We still use the same confusion set so there is 135 nodes of each class, and therefore pick the 135 highest scoring nodes for each feature, when given a class, and see how many nodes are false or true predictions. This is done since the different classes have different distributions for the likelihood, and we want to know how much confusion there is when we pick the highest scoring nodes for a given feature with a given class. It should be noted that we are still only using nodes that belong to one of our classes in the confusion test set. The full DOM-Trees will in most cases contain more than 500 nodes that have no class, and in some cases over 8000 nodes that have no class. The reason for keeping only the nodes that belong to one of our classes, is because we want to see the classifiers ability to distinguish between them.

Author Confusion

In the author section of this experiment we sorted the nodes by the score for each feature when given the author class and picked the 135 nodes with the highest ranking. All correctly predicted nodes will be placed in the author column, while the nodes placed in the content or time column are wrongly predicted nodes.

Author Confusion			
Feature:	Author	Content	Time
Average Local Split Token Length	115	20	0
Average Local Token Length	89	1	45
Average Split Token Length	93	42	0
Average Token Length	66	24	45
Initial Token Count	124	1	10
Local Initial Token Count	119	11	5
Local Special Token Count	74	61	0
Special Token Count	120	15	0
Sub-Node Count	113	2	20
Number Percentage	33	98	4
Final Score	130	5	0

Table 5.22: Author Confusion

From the results, we see that the classifier has major problems with the features: Average Token Length, Local Special Token Count and Number Percentage. These three features most likely hurts the ability to find author nodes more than they help. Especially Number Percentage is confusing, with only 33 out of 135 nodes are correctly identified as author. It is even worse for the traditional confusion matrix with only 4 nodes correct. For the features: Average Local Token Length and Average Split Token Length, we get less than 100 of the nodes correctly identified, but they offer some help in finding author nodes. They get 69 and 98 nodes correct in the traditional confusion matrix.

The feature Average Token Length gets 74 nodes correct in the traditional confusion matrix and 66 in our modified confusion matrix. These results make a case for not using this feature at all for the author classification.

Local Special Token Count has problems in our modified confusion matrix, with 74 nodes correctly predicted, but works well in the traditional confusion matrix with 133 out of 135 nodes correctly predicted. This feature did however have major problems with content and time classes in our traditional confusion matrix, where a lot of content was falsely predicted as authors, while almost 50% of our time nodes were predicted as content.

The last feature that caused a lot of problems for author was Number Percentage. This feature was made specifically for identifying date and time nodes. Skipping this feature for author will give more accurate results.

The rest of the features correctly identifies 113 or more nodes correctly and work well. The Final Score predicts 130 out of 135 nodes correctly, which is good but not perfect.

Content Confusion

The procedure for this experiment is the same as for author, except the given class is content.

Content Confusion			
Feature:	Author	Content	Time
Average Local Split Token Length	1	113	21
Average Local Token Length	33	22	80
Average Split Token Length	14	120	1
Average Token Length	25	106	4
Initial Token Count	0	98	37
Local Initial Token Count	12	6	117
Local Special Token Count	41	29	65
Special Token Count	1	67	67
Sub-Node Count	0	94	41
Number Percentage	49	86	0
Final Score	34	101	0

Table 5.23: Content Confusion

The classifier has a lot of problems with the content classification. Four of the features get less than half the nodes correctly and the worst feature, which is Local Initial Token Count, gets only 6 out of 135 nodes correct. Average Local Token Length which is the second worst gets 22 out of

135 correct, and Local Special Token Count only gets 29 nodes correct. The last feature which has problems is Special Token Count. This feature classifies content and time equally at 67 nodes for each class when using the distribution for content. The rest of the features get between 86 and 120 nodes correct, which does help in the classification.

The Average Local Token Length does rather well in the traditional confusion matrix with 113 out of 135 nodes correctly identified, but is a complete disaster in our modified confusion matrix with only 22 correct. Local Initial Token Count performs similarly with 124 nodes correctly in the traditional confusion matrix, but only 6 correct in this experiment. Both of the features prefer time nodes over content nodes, which can lead to confusion in the total score.

Local Special Token Count does bad in both confusion experiments with 29 correct in the modified confusion matrix and only 8 correct in the normal confusion matrix. This shows that this is a terrible feature overall for content. Special Token Count gets 67 out of 135 right in this test and 75 correct in the traditional confusion matrix, which is not very good. This feature should be reconsidered for content.

The rest of the features work well enough that we feel they should be kept. The total scoring for the classifier gets 101 nodes correct, while 34 are falsely predicted as author nodes.

Time Confusion

This is the final experiment with the modified confusion matrix, and therefore uses the same procedure as author and content classes, but for the time class.

Only the Average Token Length performs terrible for the Time class. It gets only 25 nodes correct in our modified confusion matrix, but gets 88 nodes correct in the traditional confusion matrix. The feature definitely works below average in total. Average Local Token Length, doesn't perform very well in in this experiment either. It only gets 63 out of 135 nodes correct, but gets 105 nodes correct in the traditional confusion matrix.

Among the other features only special token count has less than 100 nodes correct in both of the confusion experiments. It scores 98 correct in this experiment and 114 correct in the traditional

Time Confusion			
Feature:	Author	Content	Time
Average Local Split Token Length	3	0	132
Average Local Token Length	50	22	63
Average Split Token Length	3	2	130
Average Token Length	37	73	25
Initial Token Count	13	12	110
Local Initial Token Count	25	0	110
Local Special Token Count	2	6	127
Special Token Count	2	35	98
Sub-Node Count	13	22	100
Number Percentage	14	1	120
Final Score	3	0	132

Table 5.24: Time Confusion

confusion matrix. We do feel it is worth keeping this feature, but it is not outstanding.

The rest of the features work well in both of our confusion experiments and are worth keeping. The overall classification gets 132 out of 135 nodes correct, with the last 3 nodes being authors.

Our Feature Recommendations

After analysing both of our confusion experiments, we scored each of the features on a scale from 1-5. Our recommendations are based on the confusion experiments and our analysis of discussion boards.

- **1** Features that are bad and should not be used
- **2** The feature can be used, but it is not recommended
- **3** A bland feature that can lead to some confusion, but arguments can be made for using it
- **4** A good feature that should be used
- **5** Gives very good results and we see no reason for not using the feature.

Feature recommendations			
Feature:	Author	Content	Time
Average Local Split Token Length	5	5	5
Average Local Token Length	3	3	4
Average Split Token Length	4	5	5
Average Token Length	3	5	2
Initial Token Count	5	4	5
Local Initial Token Count	5	3	5
Local Special Token Count	4	1	4
Special Token Count	5	3	5
Sub-Node Count	5	4	5
Number Percentage	1	4	5

Table 5.25: Time Confusion

5.1.2 Accuracy in Complete Discussion Board Pages

Our Naive Bayes implementation was primarily made for scoring every node in the DOM-Tree, to get the likelihood of each node belonging to a certain class. We have three classes for desired data which are author, content and time. All other nodes are considered to be nodes that have no class in the Naive Bayes. This experiment focuses on how well our classifier works, when finding nodes of a certain class in the full DOM-Tree.

The normal way of measuring accuracy of a Naive Bayes classifier when it comes to finding a certain class, is to use a cutoff value for the classifier score. Since we needed to know the classifiers ability to distinguish nodes of a certain class from the other two classes and all the nodes that do not belong to any of our classes. By using our node marking templates, we know which nodes are of the author, content and time classes, and which nodes have no class at all. For our cutoff value we decided to use the amount of nodes, from the class we are searching for, that has been found. The gaps in cutoff value is 10%. This means that with a discussion board that has 10 posts, and are looking for the author class, we have reached our first cutoff when 1 author node has been found and the fourth cutoff when 4 author nodes has been found. The cutoff values for this example would be 10% and 40% respectively. If a discussion board has 30 posts, then each gap contains 3 nodes of a certain class.

The first step in the experiment is to sort all of the nodes by the score for the class we are searching for. If we are searching for author class, we sort all the nodes by author's final score, with the highest scores at the top. We then search through the nodes, with descending score, until we have found the amount of nodes that make up a cutoff level. Once we have reached one of our cutoff levels, we calculate the percentage of nodes that we have checked which belongs to the class we are searching for. In a discussion board with 20 posts, we have our first cutoff level at 10% of the nodes which is 2. If we have to go through a total of 2 nodes to reach our cutoff level, then we have 100% accuracy. In another case we have to go through 10 nodes to find the 2 nodes that make up our 10% cutoff level we get 20% accuracy. If we have to go through 40 nodes to get all the 20 nodes of the class, which is 100% of the nodes as the cutoff, we would have 50% accuracy. Our accuracy depends on the number of samples that we have to go through, in order for us to find the amount of nodes, from the class we our searching for, that the cutoff level dictates. Accuracy increases when the amount of samples decrease, and increases when the amount of samples increase, for the same cutoff level.

We performed the experiment two times for each class. The first run was done with all the features for the Naive Bayes, while the second run was done with features that we scored 3 or higher in our confusion experiments. This allows us to see if removing features, improves the accuracy of our classifier. We used the final training set which contains 34 discussion board pages in total, from 17 different discussion boards. We decided not to include the discussion boards which use less than 10 posts per page, which left us with 30 pages from 15 discussion boards since two of the boards used 8 and 9 posts per page respectively. This left us with 470 posts with one node of each class per post. Appendix A.2 points to the location of the page set used in this experiment.

Author Accuracy

The diagrams for author class shows that the classifier is more accurate with all the features throughout the first 6 cutoff levels, which is 60% of the author nodes, except for the 2nd cutoff level. From the 7th through the 10th cutoff level the modified classifier is more accurate. The accuracy for finding all the nodes is 15.74% for the initial classifier while the modified classifier has a 18.39% for finding all the nodes. This is almost a 3% improvement which comes from the

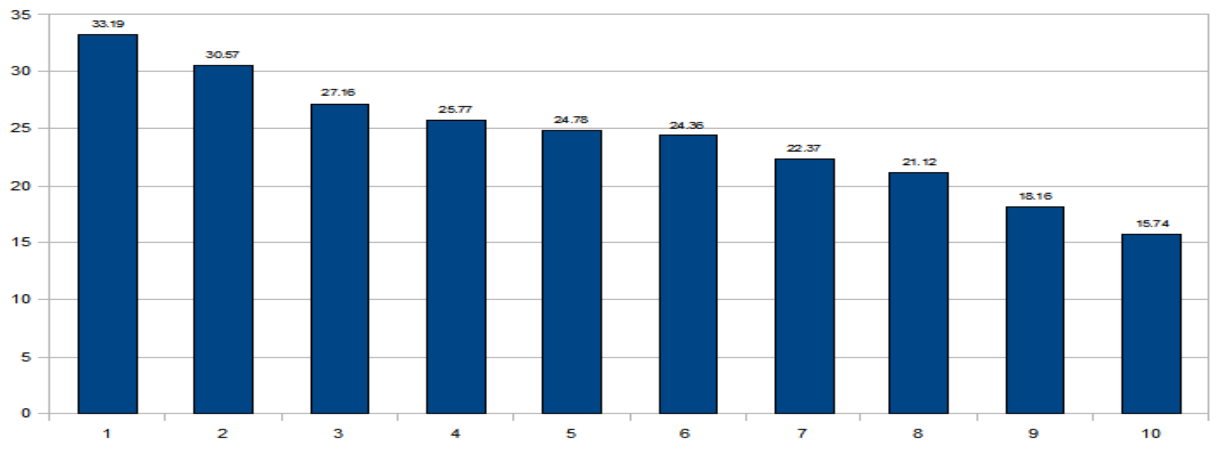


Figure 5.1: Author all features

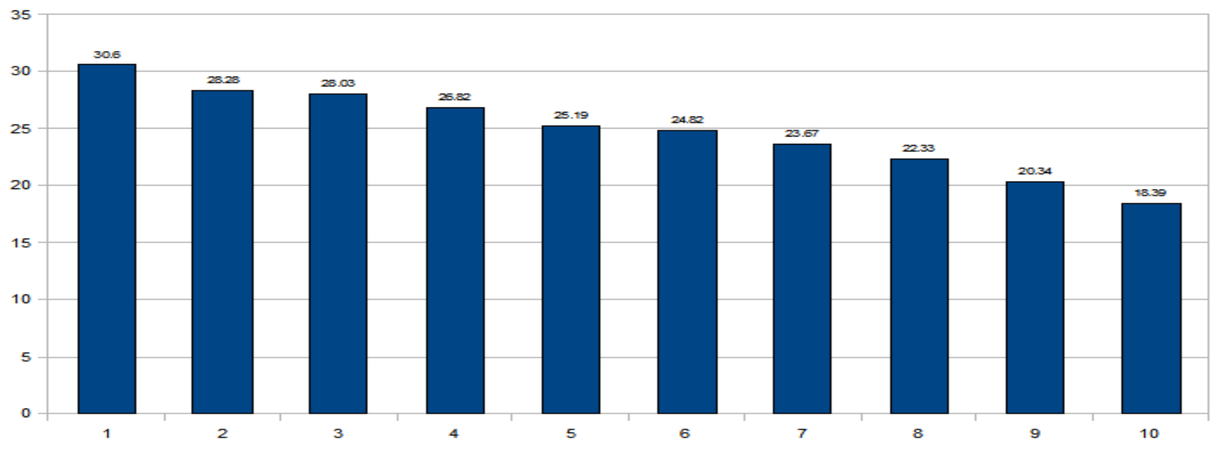


Figure 5.2: Author tweaked

modified classifiers improved ability to find author nodes that are difficult to classify. The best accuracy for all the nodes, show that we will need other measures than just the Naive Bayes in order to find the nodes of Author class.

Content Accuracy

The results for each cutoff level are very close, but the classifier that uses all the features is slightly better overall. While the confusion results showed that some features were a bit confusing for the

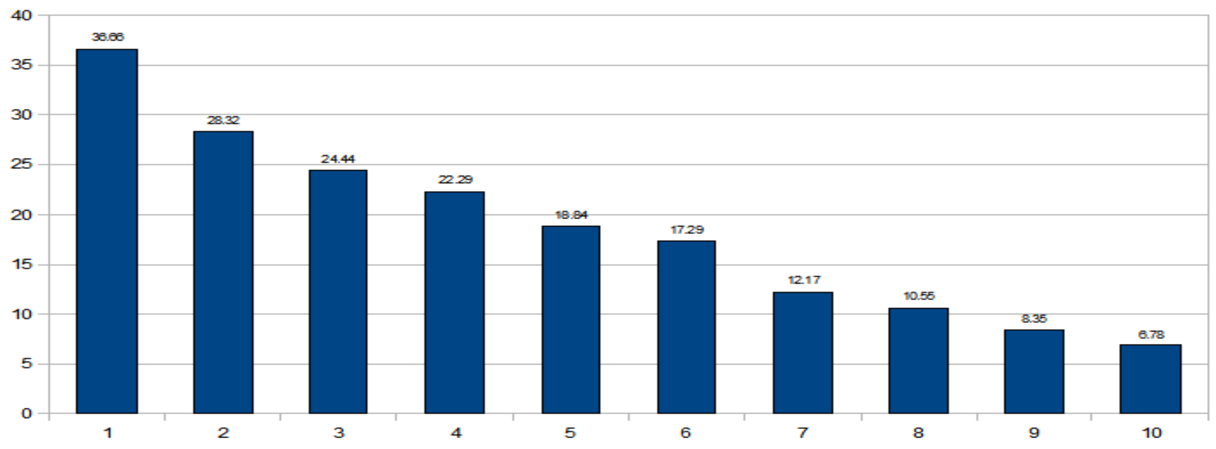


Figure 5.3: Content all features

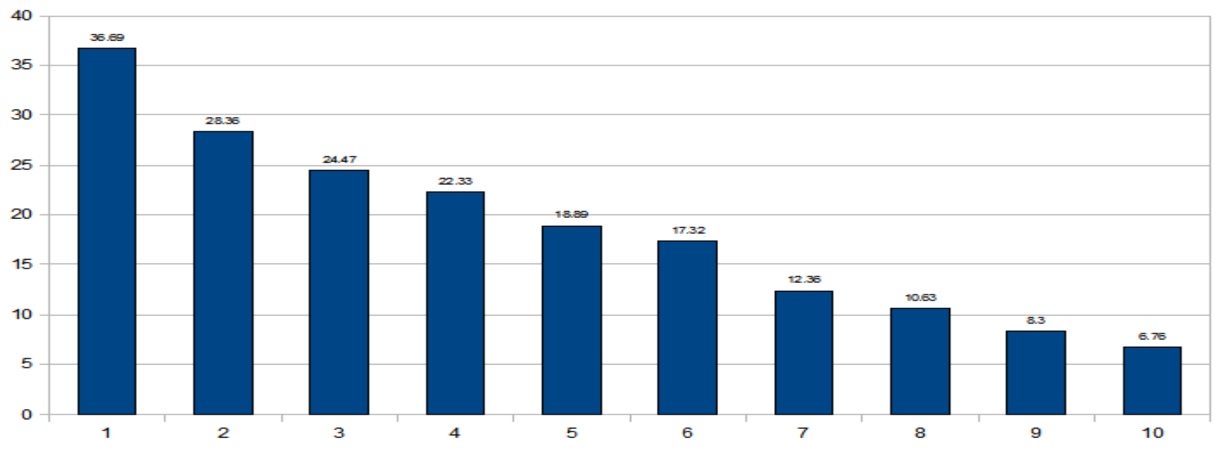


Figure 5.4: Content tweaked

classifier and one feature was simply awful, the classifier still performed better when using all the features. We believe there are two reasons behind this result. The first part is that this experiment uses a much larger sample size with 470 nodes compared to 135 nodes in the confusion experiment. This gives us more variety in the content nodes that we analyse. The second part is that the only nodes included in the confusion experiment were those which belonged to one of our classes. This experiment includes all the nodes in the DOM-Tree, which gives a lot of other nodes that can look like content nodes. Since our classifier works primarily on text, the parent node and child nodes of content node, will look very similar to the actual content node.

The overall scores for content are rather bad. A 6.78% accuracy means that we would have to look through almost 150 nodes to find 10 of the content class, while we would have to check close to 300 nodes if the page contains 20 nodes of content class. Our conclusion for the content classification is that we can only use the Naive Bayes as a helper for structure classification in our effort toward identifying the content.

Time Accuracy

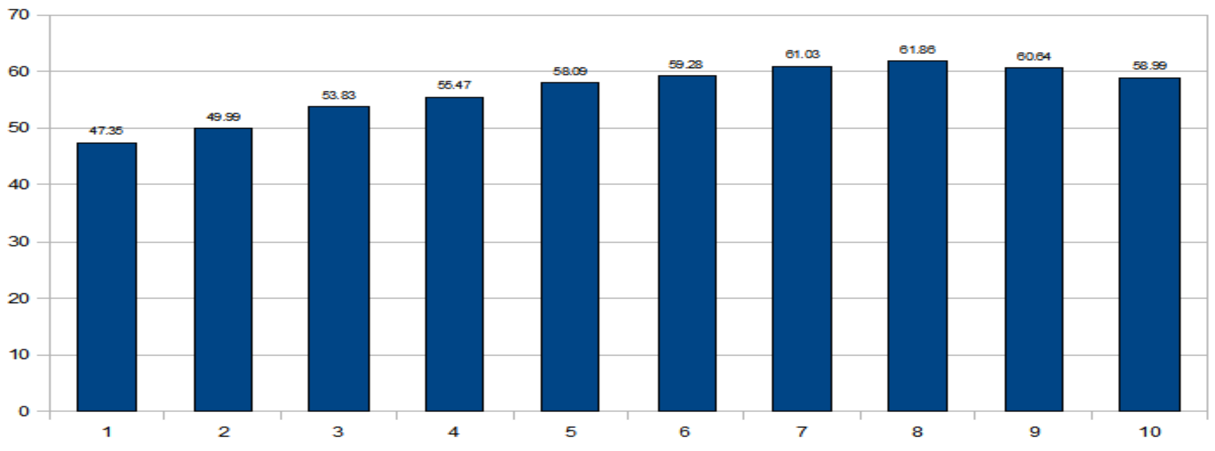


Figure 5.5: Time all features

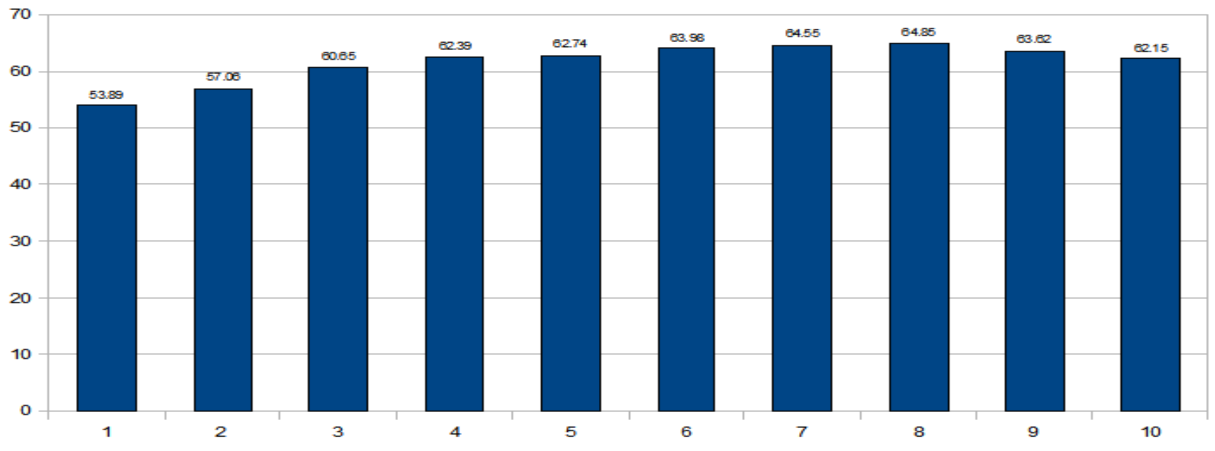


Figure 5.6: Time tweaked

On average our modified classifier is almost 5% better than the classifier which uses all the feature. Our ability to identify time nodes, by using the classifier, is quite good. If we have a discussion board with 10 posts and get our average accuracy for finding all the time nodes, then we would find all of them among the 17 highest scored nodes. With 30 nodes we would need to look through the 49 highest scored nodes for the class. When you add comparison of the nodes path in the DOM-Tree, we should be able to consistently find all the time nodes.

5.2 Structure Classification

The result of extracting data on discussion board was judged on the success rate of identifying the correct author, content and date+time. We also measured the success of identifying the three classes in the same post.

This section contains result on the influence of Naive Bayes, structure classification features and grammar. In order to keep the experiments as accurate as possible, we selected 31 pages with 456 posts from 16 discussion boards. We have a page set which contains 2 pages from 15 of the discussion boards and 1 page from 1 discussion board. The location of the pages can be found in Appendix A.4.

In the experiments we tried different combinations of Naive Bayes, grammar, structure features, date+time factor and content factor. This was to show how the different parts of our project influences the result. In author classification, we tested the pages with and without grammar and Naive Bayes score of author. In content classification, we tested the pages with and without using Naive Bayes score and equivalent sub-nodes counting of content. In date+time classification, we tested the pages with and without grammar Naive Bayes score of date+time.

A post classification is successful when the author, content and date+time under the post are all successful. It is different from post node success, which means just the path of the post node is correct. This result is heavily influenced by our success in finding the different classes since it requires all 3 to be correct.

5.2.1 Author

The author classification assumes that the post node has been classified. The author classification includes two steps. The first step is to select the possible authors, which are leaf nodes and have same relative path. In this step grammar of author is also used. The second step is to sort the possible authors. The sorting is based on Naive Bayes score. In addition, the author nodes which have the same text will be sorted will be sorted as the least likely to be author nodes.

Naive Bayes Score and Grammar

In order to test the impacts of Naive Bayes score and grammar in author, we did 4 experiments with and without Naive Bayes score and grammar. Table 5.26 is the author classification result of the four experiments.

	With Naive Bayes Score	Without Naive Bayes Score
With Grammar	94%	76%
Without Grammar	83%	48%

Table 5.26: Author Result with or without using Naive Bayes and Grammar

Figure 5.7 is the accuracy of the author results when using both Naive Bayes and grammar:

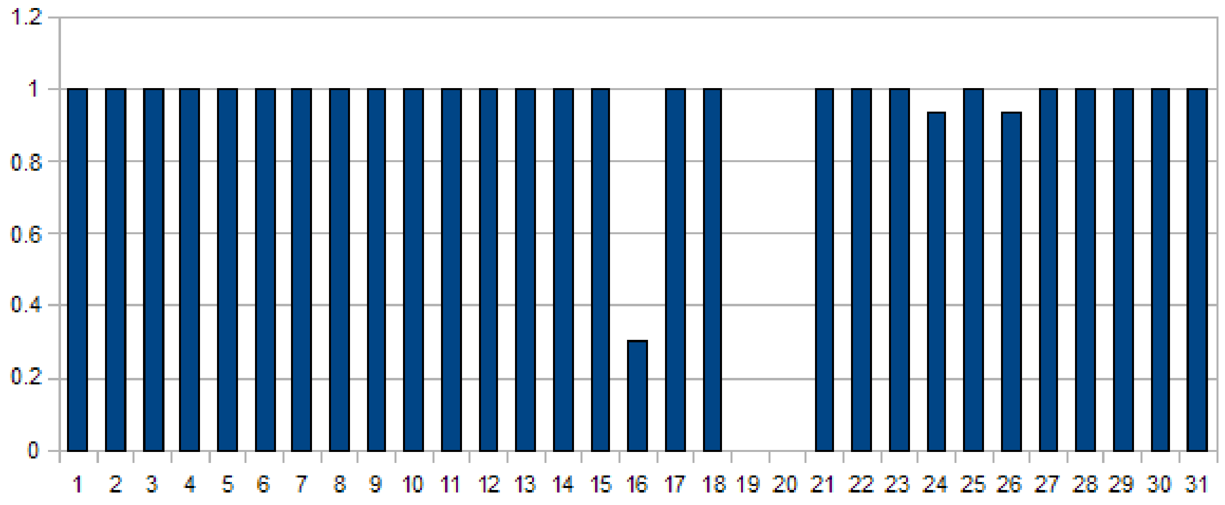


Figure 5.7: Author Success Accuracy with using Naive Bayes and Grammar

There are two typical cases where the author classification failed. The first case is when the the author's title is mistaken for being author, for instance 'moderate,' 'beginner' and similar is recognized as author. The second case is when the classification of the post nodes failed.

5.2.2 Content

Like in the author classification, we assume that the post node has been classified before classifying content. There are two steps of content classification. The first step is to select the content groups whose content node have the same relative path and few equivalent sub-nodes. The second step is to sort the possible content groups.

There are three parts in our algorithm which influences the final result. The first is the equivalent sub-nodes counting, the second is the Naive Bayes score and the third is content factor. The content factor derived from combining Naive Bayes score, size of equivalent sub-nodes and total sub-nodes.

We did several experiments, in order to observe the impact that each part has on the final result. First we had 4 experiments with and without using Naive Bayes score and equivalent sub-nodes counting. Then we did 12 experiments, in which we adjusted the value of the factor.

Naive Bayes Score and Equivalent Sub-Nodes Counting

If the experiment is done without using Naive Bayes score and equivalent sub-nodes counting, there is no sorting for the possible contents. Using either Naive Bayes score or equivalent sub-node counting will increase the result a lot. The combination of using Naive Bayes score and equivalent sub-nodes counting achieves an even higher success rate. The result will therefore be low in this case. The results are shown in Table 5.27.

	With Naive Bayes	Without Naive Bayes
With equivalent sub-nodes counting	76%	67%
Without equivalent sub-nodes counting	66%	10%

Table 5.27: Content Success Rate in 4 Types of Classification

The success rate when using both Naive Bayes and equivalent sub-nodes counting is shown in Figure 5.8.

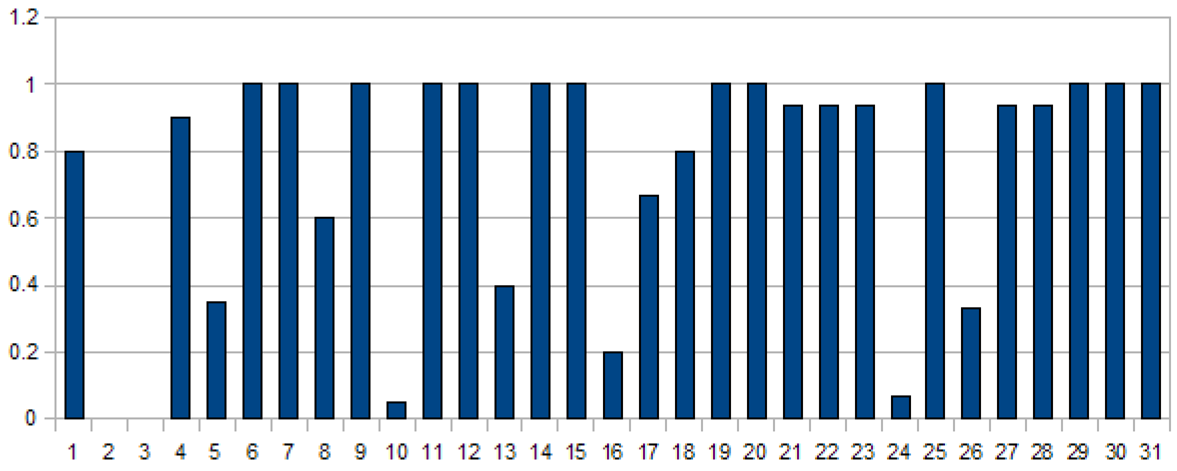


Figure 5.8: Content Success Accuracy with using Naive Bayes Score and Equivalent Sub-nodes Counting

There are two main reasons for the failed classification cases. The first reason is that the node contains both signature and real content. Because some people have signature while others do not, the correct content should not contain signature, but some of the nodes with the same relative path do contain signature. The second case is that when the date+time classification failed but post node is classified correctly, the date+time which should be classified successfully is in the content. For instance in the second case, there are two date+times, *"2007-10-3 12:13"* and *"register time 2005-10-2"*, and the content text is *"Hello,everyone!"* if the second date+time is classified as date+time, the content is the node which has the text *"2007-10-3 12:13 Hello,everyone"*.

Content Factor

In our algorithm, there is a calculation for the Naive Bayes score, equivalent sub-nodes and total sub-nodes:
*Content score=average Naive Bayes position+factor*equivalent sub-nodes/average total sub-nodes*

When the factor is small the Naive Bayes score has more influence on results. If the factor is 0, then only the Naive Bayes score is used. When the factor is large, the influence of equivalent sub-nodes counting is increased. If the factor is large enough, only the equivalent sub-nodes counting is used. Table 5.28 shows the experiment result.

Content Factors	Content Success Rate
0	61%
0.5	68%
0.7	71%
1	71%
1.5	73%
3	76%
5	76%
8	74%
50	67%
150	67%
300	67%

Table 5.28: Content Success Rate on Content Factors

The results for the factor experiment shows that a very small factor is less accurate. The accuracy rises until we have a factor of 5 and shows a decline in accuracy after that. The experiment shows that the optimal factor is between 3-5. We therefore set this factor to 4 in our other experiments.

5.2.3 Date+Time

There are two steps in classification process of date+time. First the nodes which are the best fit in the structure classification will be selected and placed in groups based on path. Then the groups of date+time will be sorted. The Naive Bayes score is helpful in sorting the date+time groups while the grammar helps to filter out nodes that are unlikely to be date+time.

We had 4 experiments for date+time classification to observe how the Naive Bayes influences the final result. Since we used an algorithm to combine Naive Bayes and date+time count, there is a factor which influences the final result. We did several experiments to determine the optimal factor. We changed the value of factor to a number between 0-15000 and observed the result in the experiment. We had to use Naive Bayes score and grammar, in order for the factor experiment to work.

Naive Bayes Score and Grammar

We did 4 experiments on finding date+time. It was performed with combinations of Naive Bayes score and grammar. Table 5.29 shows the average success rate of date+time with or without Naive Bayes score and grammar.

	With Naive Bayes Score	Without Naive Bayes Score
With Grammar	86%	53%
Without Grammar	57%	30%

Table 5.29: Date+time Result on 4 Types of Classification

Without using Naive Bayes score and grammar,the experiment is only based on structure classification. The success rate is 30%. In this case, the only feature for sorting is the count of date+time. It's easy to classify nodes inside or outside the post as date+time. In our results from this experiment, 8 out of 31 nodes are register date, 3 are authors and the other nodes contains part of content and author. Our result increases a bit, to 53% when using grammar to help in the structure classification. The wrongly classified nodes are the content, join date and node containing both author and join date. Using Naive Bayes as a helper for the structure analysis, increased our result to 57%. The false positives are nodes that the Naive Bayes finds to be similar to date+time. These nodes include nodes under content, author etc. We achieve our highest success rate when using both Naive Bayes score and grammar as helpers for the structure classification.

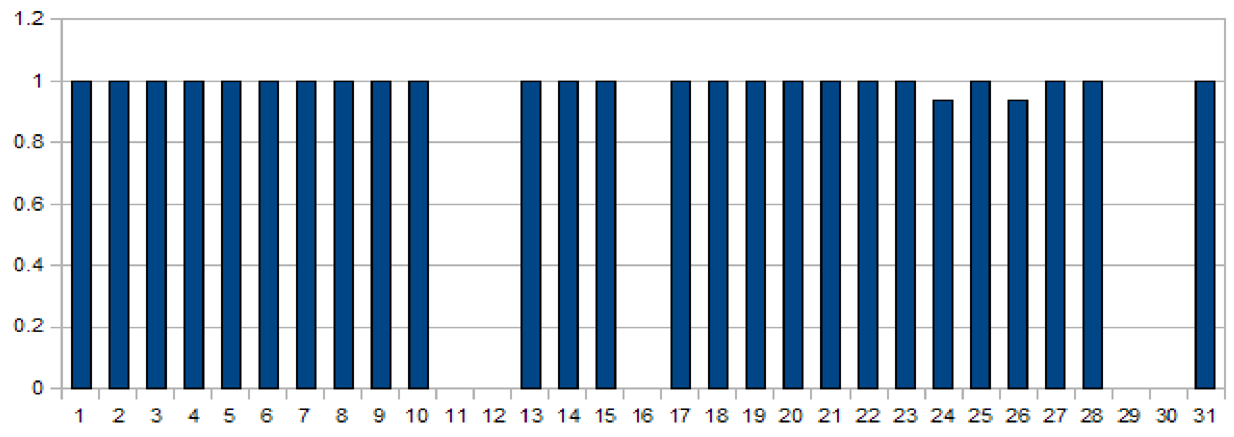


Figure 5.9: Date+time Success Accuracy with using Naive Bayes and Grammar

In Figure 5.9, our success rate is 100% for many of the discussion board pages, but there are some pages where we have 0% success. In our experiment, the main reason for failed predictions is the register date+time. For instance, *"01-05-2006 14:23"* and *"join date:01-02-2006"* can not be 100% distinguished in our approach.

Date+Time Factor

We thought of date+time count as a feature for classifying posts. We consider date+time groups with, which contain a higher number of date+time nodes as more likely to be the correct ones. Without Naive Bayes and grammar checking, the only sorting feature is date+time node count.

We explained in the research Chapter that the final score of a date+time group equals *"average position of date+times in the list * 'date+time factor' / posts count"*, and the date+time factor's value influences the sorting of the date+time groups. In our experiment, the date+time factor is set to 200, which has balanced the average position(2-10) and total posts count(10-20).

Table 5.30 shows the date+time experiment result on different date+time factors.

Date+time Factors	Success Rate
0-100	83%
150	86%
200	86%
400	84%
500	83%
600	81%
1300	81%
1800	78%
2000	68%
5000	58%
12000	58%
15000	53%
>15000	53%

Table 5.30: Date+time Success Rate on Date+time Factors

The success rate trend, that is based on the factor, increases, levels out and then decreases.

When the factor is 0, meaning post length features do not influence the result, the success rate is 83%. When the factor increases, the posts length will be more influential. If the factor is too large, then Naive Bayes is not used and the the success rate decreases to 53%. The top rate of factor in our experiment is about 150-200, which is the one we chose for the rest of our experiments.

5.2.4 Final results

Figure below shows the final result with our best classification combinaton, including the optimal date+time and content factors.

Figure 5.10 shows the correct prediction of the classes. The post's success rate is influenced by the author, content and date+time results. Content will be the main influence since it has the lowest accuracy. If the the date+time's success rate is 100%, author is 100% and content is 40%, then the post rate will be 40%.

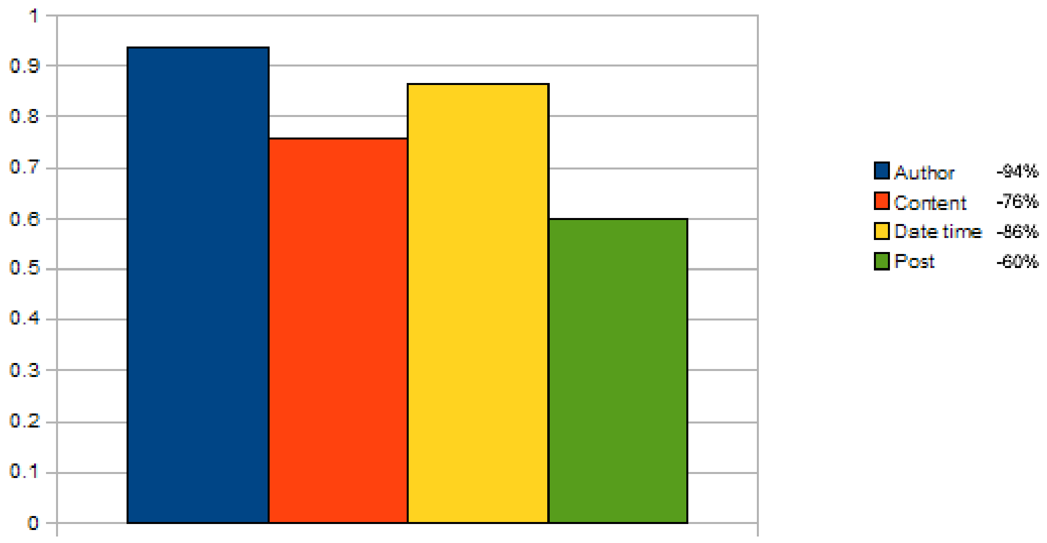


Figure 5.10: Success Rate of Author, Content, Date+time and Post

The results show that author has rather good accuracy at 94%. Our ability to predict date+time is decent with an 86% accuracy, which would be increased, if we were able to distinguish the join dates, which are written with both date and time in numbers, from the time of posting. The score for content is not very good, at 76%. Our low content accuracies influences our ability to find complete posts, quite a bit. The overall score for complete posts is 60% which is rather bad, but we do believe the scores for author and date+time shows that our approach could work with a some improvements.

Chapter 6

Construction of the Prototype

This is our engineering chapter where we give a quick overview of our prototype from an engineering perspective.

6.1 The prototype

During our thesis, we made a research platform which we later developed into a prototype for finding the nodes which contain the author, content and date+time. The prototype is used in the training of the Naive Bayes classifier. It also served as our platform for performing experiments and adjusting the algorithms to improve their performance.

Our prototype is divided into one large program and two smaller programs. One of the smaller programs is used to normalize and transform the Naive Bayes accuracy results into a more readable format. The other creates the distributions. This is after the feature values for all the nodes that belong to a class, has been written to .csv files. The data in the .csv files are then analysed and the distribution parameters written to new .csv files.

6.2 The Packages

Our goal with the prototype was to show how our approach to classifying discussion boards works. Configuration of the program therefore has to be done in the program code itself. Our prototype is written in Java [9] and consists of 6 different packages. The diagram in Figure 6.1 shows the different packages and the classes that they contain.

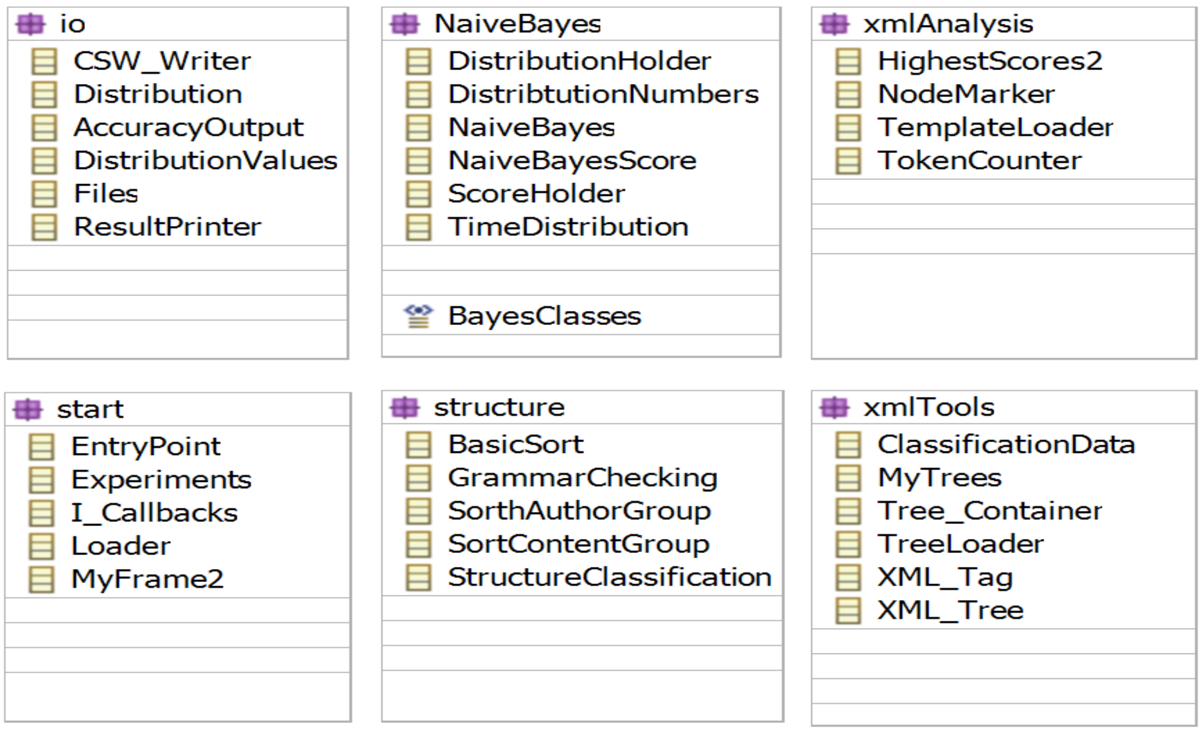


Figure 6.1: Prototype Package Diagram

- **start**
  The 'start' package contains the entry point of our main program and our graphical user interface. Initialization of the different parts of our program happens from here.

- **io**
  Read and write operations on files is done by the classes in the 'io' package. All of the methods in this package are static with the exception of those found in the 'Distribution' class.

- **NaiveBayes**
  The 'NaiveBayes' package contains data structures for holding the distribution parameters for the classification, the methods that do the classification and data structures for holding the classification scores.

- **structure**
  We placed the structure classification part of our program, in the package named 'structure.' This is where we identify the nodes that we want to extract data from. Verification of our node identification results are also done in this package.

- **xmlAnalysis**
  'xmlAnalysis' contains the methods and classes for marking nodes in the DOM-Trees, after they are loaded into memory. We also placed methods for calculating the feature values for each node and methods for sorting nodes based on Naive Bayes scores, in this package.

- **xmlTools**
  The final package is 'xmlTools,' where we retrieve the DOM-Trees from XHTML files and move them into our custom data structure. This package contains a data structure for keeping the feature values, prior to Naive Bayes scoring.

6.3 The Main Program

The main method of our primary program, is located in the 'Entry_point' class, which resides in the 'start' package. Here we set the parameters, that controls the operation of the 'Loader' class, which forms the backbone of our program. The class diagram for our prototype is shown in Figure 6.2.

The parameters and what they control:

- useExperiments: Naive Bayes confusion and accuracy experiments
- useGUI: Showing the GUI
- useNaiveBayes: Controls if the Naive Bayes is used

- useStructureClassification: Structure classification
- useMarkNodes: Marking of nodes, through use of templates
- useWriteCSV: Writing node features to .csv files
- dist: The path of the distribution files
- folder: The folder where the XHTML files and corresponding XML files with node paths are located
- outPutFile: The path and filename of where we write the results from the structure classification experiments

There are certain dependencies between our parameters. The 'useExperiments' and 'useStructureClassification' parameters, enables methods and algortihms that requires the Naive Bayes classification to run first. Since they print results that are related to our marked nodes, they also require the 'useMarkNodes' to be enabled. This applies to 'useWriteCSV' as well.

For our Naive Bayes to run, we require the distributions to be present. This means that when training the classifier, only 'useWriteCSV,' 'useMarkNodes' and 'useWriteCSV' should be enabled.

The folder paths should always be correct. However, in the cases where the Naive Bayes is not running, only the 'folder' path is needed. The 'useGui' has no dependencies, except for the path to the folder containing the XHTML files.

6.3.1 Loading the XHTML Files

When one of the constructors for the 'Loader' class is called, new instances of 'Tree_Container' and 'TreeLoader' are made. The 'TreeLoader' retrieves the DOM-Tree from a specified file by using parts of a Xerces [7] sample application. We then copy the data, in the XHTML elements, from the DOM-Tree into instances of 'XML_Tag.' The DOM-Tree that we get from the sample application, keeps the text in nodes that are separate from the XHTML element nodes. For our purposes it is easier to if the local text and the text contained in the entire sub-tree of the node is easily accessible. We therefore store this in the 'XML_Tag' as well. During this process we build a new tree that contains only the XHTML nodes and place it into an instance of 'XML_Tree.' References to the nodes are also kept in an flat list within the 'XML_Tree' class. This allows us to iterate over this list during analysis, instead of walking the tree, which makes it much easier to perform analysis of each node.

Once the copying of the original DOM-Tree is done, we place both the DOM-Tree from the Xerces [7] sample application and our copied tree into a 'MyTrees' object. This also contains the file paths for the XHTML page. This process is repeated for all the XHTML pages found in the folder and sub-folders that the 'folder' path parameter points to. In the end, we place all the 'MyTrees' into 'Tree_Container' which keeps all the trees in memory.

6.3.2 Marking the Nodes

The marking of the nodes is done in the 'loadXML' method in the 'Loader' class. It iterates over all the 'MyTrees' in 'Tree_Container' and sends a document, which contains the DOM-Tree paths of the nodes that we wish to mark, to the 'NodeMarker' class. We then walk our instantiation of the DOM-Tree and mark the nodes with Naive Bayes class.

6.3.3 Getting the Feature Values for Each Node

Our Naive Bayes requires the values of our features for every node in the DOM-Trees. The process of acquiring these values is done by sending the list of all our trees to the 'TokenCounter' class. We then iterate over all the nodes in all the trees and run algorithms on each node to get the feature values. When the 'useWriteCSV' parameter is enabled, we write the features to for all the nodes in the DOM-Tree that belong to a Naive Bayes class, to file. Initially we write one file for each XHTML file that is loaded and merge the results into one file once we have gotten the features from all the DOM-Trees.

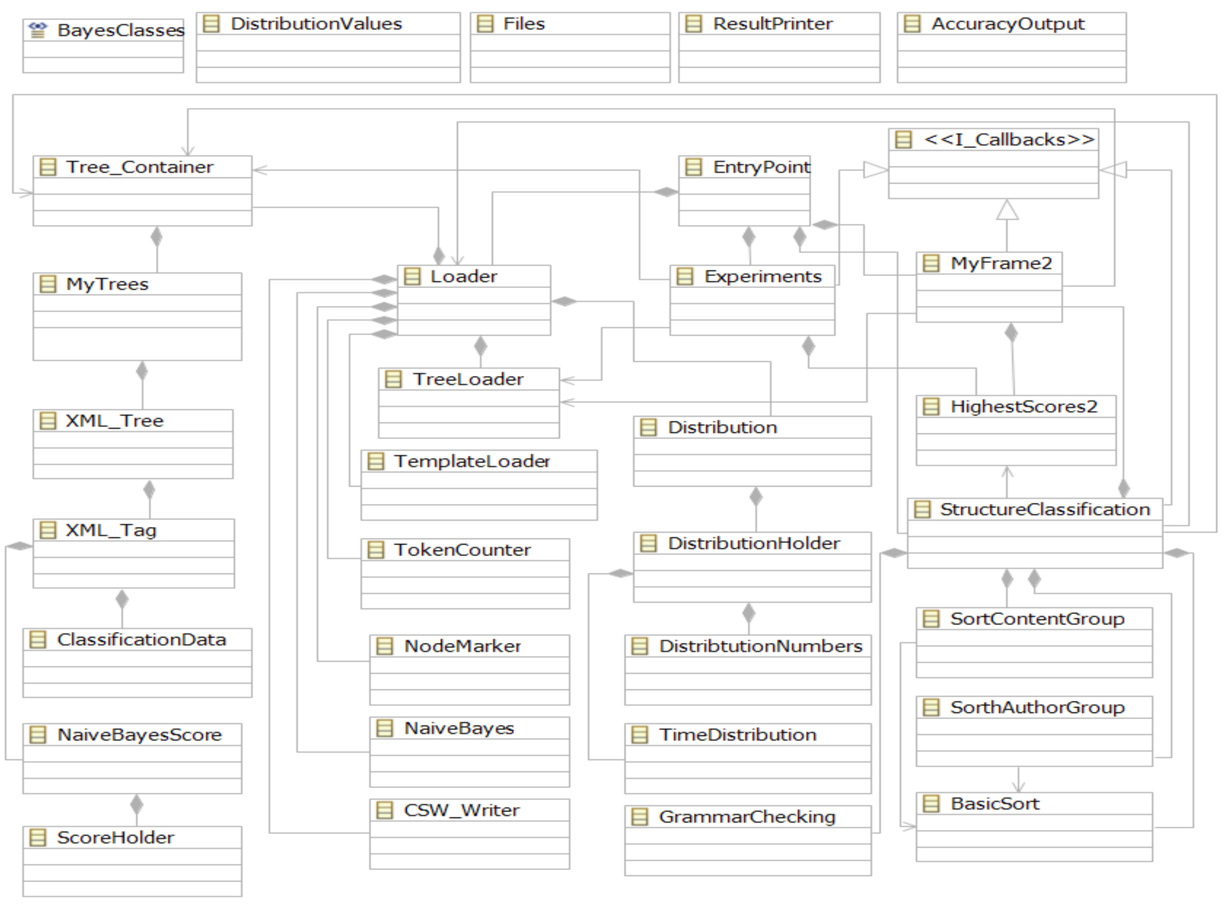


Figure 6.2: Prototype Class Diagram

6.3.4 Naive Bayes

There is three parts to our Naive Bayes. The first is loading the distributions parameters into memory. The second is scoring the nodes, using our distributions and the third is keeping the scores for each node that belongs to a DOM-Tree. We also run several experiments with the Naive Bayes, which is done in the 'Experiments' class.

The loading of the distribution parameters is done in the 'Distribution' class, which belongs to the 'io' package. This class keeps three distribution sets, one for each of our Naive Bayes classes. A distribution set consists of a 'DistributionHolder' which keeps one 'DistributionNumbers' for each feature in our Naive Bayes and one 'TimeDistribution.' The 'DistributionNumbers' class is a simple data structure with two doubles, one for the average value of the feature and one for the standard deviation. The ’TimeDistribution’ contains a special version of the percentage of numbers feature which only applies to date+time nodes.

The Naive Bayes scoring is done by iterating over all the nodes in the DOM-Trees and scoring the likelihood for each feature given a class. This process is done for all of our Naive Bayes classes. The scores for each feature when given a class is then multiplied to form our final score. We keep the scores in the class ’NaiveBayesScore,’ which keeps three instances of ’ScoreHolder,’ one for each Naive Bayes class. The ’ScoreHolder’ class stores the likelihoods for each feature and the final likelihood for a node when given a certain class.

We wrote the probability engines for the Normal distribution and our special time distribution ourselves, but chose to use a pre-made probability engine from JSC [15] for the Poisson distribution.

Our Naive Bayes experiments, which are done in ’Experiments,’ compare the feature and final score likelihoods between the classes for each node. It also calculates the accuracy for identifying nodes in the complete DOM-Tree. A more thorough description of these experiments can be found in Chapter 5.1.

6.3.5 Graphical User Interface

The graphical user interface is contained in the ’MyFrame2’ class and uses a tabbed pane to display multiple ’JTrees’. New trees can be registered through the ’setScoreTree’ method. The panel on the left side shows data for the selected node. The data consists of feature values, local and subtree text when an ’XML_Tag’ is selected. It can also show Naive Bayes values, for a given class. However this requires the Naive Bayes class to be set in the code, since we did not give an option to swap this class in the GUI itself.

6.3.6 Structure Classification

The structure classification classifies XHTML page files that can be found in specific set of folders. The root folder for the page files is set in the 'folder' property in 'Entry_Point' class. It verifies the structure classification result, and outputs them to .csv files. The classification process includes two parts. The first part is finding nodes that correspond to the features used in structure classification and the grammar rules. The second part is sorting the nodes according to Naive Bayes score and structure features. The process is used in date+time, author and content classification. The verification process is done in three stages. In the first stage it selects sorted 'MyTree' from classification results. Then it compares the top node of possible date+times, authors and contents with "MyTree" objects containing correct node from the marked templates. Finally it builds output strings and write them into csv files. The classification and verification is done by the 'StructureClassification' class in the 'structure' package. 'StructureClassification' invokes the classes 'BasicSort', 'SortPostGroup', 'SortAuthorGroup', 'SortContentGroup' for sorting part of the algorithm and class 'GrammarChecking' for the grammar checking.

The 'BasicSort' class is used to sort classified objects by a classification score value. The GrammarChecking is used for all the grammar checking in structure classification.

The date+time classification starts with path comparison which is done in 'compareTime' method in the 'StructrueClassification' class. Then the possible date+time groups are sorted based on Naive Bayes score in SortPostGroup. After the classification, it will build a 'MyTree' object in memory that contains date+time, where author and content will later be added as children of their corresponding date+time.

The author classification starts with getting same relative paths, to the author nodes, within each of the posts in the discussion board page. This is done in the 'AddAuthor' method. The possible author nodes are then added into the 'MyTree' object, which was generated in date+time classification. Finally the possible authors are sorted based Naive Bayes score in 'SortAuthorGroup' class.

The content classification starts by getting the nodes with the same relative paths. We then compare the size of the sub-trees and exclude trees which has the same amount of sub-nodes in all the discussion board postings from the same page. This is done in the 'getContent' method. Several possible content nodes are added into the same 'MyTree' as a child of it's related date+time. The possible content nodes are then sorted based on Naive Bayes score in 'SortContentGroup' class.

The verification of the nodes that has been marked for extraction are compared with the marked nodes from the discussion board page. We compare the content of the author nodes that has been marked through the use of templates, with the nodes that our algorithms have marked as the most likely authors. The comparison is done in the same order as the nodes appear in the DOM-Tree. This means that the first marked author node is compared to the first predicted author node. This process is done for content and date+time as well. If the content, that we have marked for extraction, matches the content of the marked nodes then we consider the classification process to be successful. After the comparison is finished, we write the results to two files. The file with the name 'TheOutput_result.csv,' contains detailed result for every classified node. The contents of the other file is the result statistics, and it's filename is 'TheOutput_statistic.csv'.

6.4 Getting the Distributions

We selected to make the creation of the distributions into a separate program. It requires that the main program has done a training run on a training set of XHTML files with XML node path templates. The next step is to run the 'DistributionValues' class, with the text string pointing to the location of the training set. It will then run a merge on the .csv files, that contains the feature values for each XHTML page and run algorithms that get the distribution parameters and output them to the folder where training set resides.

The files that it writes to are:

- merged_author_distribution.csv
- merged_contentData_distribution.csv
- merged_time_distribution.csv
- merged_time_special_distribution_smooth.csv

The special distribution for date+time is made by taking the frequencies of percentages with 2% gaps. Since several of these gaps have zero occurances in the frequency, we draw a linear graph between the non zero gaps. This does however leave us with large amplitudes between values that are close to each other. We therefore use a sliding window to make the graph smoother. The window size is 5 gaps, two before, two after and the gap that we are calculating the value for. We handle the edges as special cases so the window size is 1 in the beginning and increases by 1 until it reaches a size of 5. When we reach the end we decrease the window size by 1 until it leaves the graph. We then have our special distribution for date+time which gives the likelihood of a feature by checking which gap it belongs to, and retrieving the value directly.

6.5 Accuracy Output

The accuracy output is for normalizing the accuracy values retrieved in our Naive Bayes accuracy experiment 5.1.2. It takes the accuracies from each XHTML page and that has either 10, 15, 20 or 30 nodes and normalizes the values so we get 10% intervals for the nodes.

Chapter 7

Conclusion and Further Work

This chapter contains our conclusion and suggestions for further work on our approach to extracting data from on-line discussion boards.

7.1 Conclusion

The focus of our thesis was on automatically finding the locations of author, content and date+time in on-line discussion boards. We used a combination of structure classification, Naive Bayes and grammar in order to find these locations.

We did an analysis of our Naive Bayes, which showed that 9 out of our 10 features worked well for finding date+time. However, the same analysis showed that several of our features were less than optimal for author and content, so there should be room for improvement in that area. We also provided an analysis of how the different parts of our extraction algorithms influenced our ability to identify data. The combination of using Naive Bayes with structure classification, provides slightly better results than grammar and structure classification in our approach for author and date+time. It is however no where near the accuracy we get when we combine all three methods with a date+time accuracy difference of 29%. These results indicate that the structure classification works better with more guidelines for the node class predictions.

Our overall result for extracting complete posts is only 60% which is not very good. This is primarily due to our problems with identifying content and separating certain join dates from the posting date+time. Our author accuracy is 94% which is good, but not great and our date+time accuracy is decent at 86%. Our problem is with the content, which only achieves 76% accuracy. While we are very strict in our verification of content, which means that we will not accept it as content unless the text exactly matches what we marked in our templates, we still feel this accuracy leaves a lot to be desired. Overall we feel our solution needs improvement to be usable in real world data extraction, but the approach shows promise and is worth pursuing.

7.2 Further Work

Due to the limited amount of time that we have to perform our Master's Thesis, there are a lot of ideas that we did not have time to do research on. We believe these ideas and approaches can solve several of the ambiguities that in some cases prevented us from identifying the correct discussion board data that we were looking for. In this section we present these ideas in the hope that people who wish to expand our work, will research and provide results on the validity of these ideas.

7.2.1 Finding the Post Path Based on Sub-Tree Size

When we started researching the structure classification, prior to having a working Naive Bayes classifier, we experimented on using the size of sub-trees to find the path to the node that is the direct owner of all the discussion board postings. This method relies on the fact that in a flat structured discussion board, the node that is the direct parent to all the sub-trees where the postings reside will have a very large sub-tree compared to it's children.

In a page where there are 10 postings with an average of 40 nodes each, a header with 60 nodes and footer with 20 nodes in the table that shows the postings on screen, the parent will have atleast 480 nodes. If there are some additional formatting nodes with small sub-trees, the parent of the postings will have even more nodes. The largest sub-tree of the parent will be the header with 60 nodes, which means there is a large dropoff in sub-tree size. This dropoff is what this approach looks for, in order to find the path of the node which owns the discussion board postings.

In our brief experimentation with this method, we walked the tree from the root until we found the node we believed to be the parent of the discussion board postings. The algorithm for walking the tree was to follow the path of the largest sub-tree. It would check all the children of the root node of the document and select the child with the largest sub-tree, then repeat this process for this node until we found a large change in the sub-tree size. When the largest sub-tree amongst the children had a smaller size than a cut-off value we considered the node we were analysing to be the owner of all the discussion board postings.

By just using 50% of the parent's sub-tree size as the cutoff value, we were able to find the correct post owner in most of the pages that we analyzed. However we felt that we would need to use a pattern classifier to take full advantage of this method. Due to time constraints we decided to not pursue this method, as it would take too much time and resources to make a training set for this classifier. In our implementation we use our Naive Bayes to find nodes that resided inside discussion board postings' sub-trees, and find the post owners path with this instead. We do however believe that using the sub-tree size differences to find the same path does have merit and would like to see research done on this approach.

7.2.2 Grammar With Term Look up Tables to Decrease Ambiguity

In our efforts to find the nodes that contained the author name, we ran into problems where user level or other similar information was confused with author name. An example of this would be if administrator, user, moderator and vip were possible user levels and all reside in leaf nodes. To our classifier these all look like good author names. We believe that using grammar where terms like these would make the node unlikely to be an author, would be of great help in the structure classification. These terms should be placed in a database or similar which the grammar can use to lookup excluded terms.

In several cases, the time is written in a format similar to "yesterday" or "two days ago." While our classifier can deal with this to some extent, we believe a grammer where these terms can compared to terms in a database would give good results in combination with our classifier and increase the algorithms ability to identify time nodes. Ideally one would want to identify the language used in the discussion board in order to limit the amount of terms to look, which would give more accurate results.

7.2.3 More Custom Fitted Naive Bayes

After discussing Naive Bayes with our supervisor, it was decided that we would use the same features for all of our classes. This limited our selection since the features would have to be quantifiable for author, content and date+time. Our confusion experiments showed that there were several features that worked less than optimal for our classes and some that were outright bad. After optimizing our Naive Bayes, by removing the worst features, we increased our overall accuracy for both date+time and author by a considerable margin. We do believe that these accuracies would increase even more by adding features that are tailor made for each specific class. It would not be possible to do feature confusion experiments for these, but we feel the advantages of custom fitting each class outweighs the negatives. By the time we got our confusion results, there was not enough time to go back to the drawing board to add such features. But it should be done if people wish to further our research into using Naive Bayes to find parts of discussion board postings or similar.

7.2.4 Use Larger Sample Sizes for Identifying the Nodes

During this thesis, our focus has been on identyfing the the desired data in one discussion board page at a time. This gives a rather small sample size, which leads to more fluctuations in the results. Combining post nodes from multiple pages from the same discussion board should help reduce these fluctuations, since the relative paths for author, content and date+time should stay the same. We believe this could greatly increase the classification algorithm's ability to find the desired data.

Bibliography

[1] Richard O. Duda, Peter E. Hart and David G. Stork, *"Pattern classification (2nd edition)"*,Wiley-Interscience, October 2000, pp 56-62

[2] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo, *"RoadRunner: towards automatic data extraction from large web sites,"* Proceedings of the 27th VLDB Conference, Roma, Italy, 2001.

[3] Arvind Arasu and Hector Garcia-Molina, *"Extracting structured data from web pages,"* SIGMOD 2003, June 9-12, 2003 San Diego, CA.

[4] Bing Liu and Yanhong Zhai *"Web data extraction based on partial tree alignment,"* WWW 2005, May 10-14, 2005, Chiba, Japan.

[5] Fabrice Even and Chantal Enguehard, *"Specific domain model building for information extraction from poor quality corpus,"*

[6] *Tag Soup Home Page*, `http://home.ccil.org/∼cowan/XML/tagsoup/`

[7] *Xerces Home page*, `http://xerces.apache.org/xerces-j/`

[8] *Apache XML Project*, `http://xml.apache.org/`

[9] *Java Home Page*, `http://java.sun.com/`

[10] *Sun Microsystems Home Page*, `http://www.sun.com/`

[11] *AOL-Netscape*, `http://netscape.aol.com/`

[12] *Microsoft Internet Explorer*, `http://www.microsoft.com/windows/Internet-explore`

[13] *Mozilla Firefox Homepage*, `http://www.mozilla.com/en-US/firefox`

[14] *Firefox Add-On*, `http://www.mozilla.com/en-US/firefox`

[15] *JSC Home Page*, `http://www.jsc.nildram.co.uk/`

Appendix A

Page Sets

During our project we have used several different sets of pages in our discussion board analysis and experiemnts. We have delivered these pages on our attached CD-Rom. This appendix gives the paths to the page sets.

A.1 Page set 1

The page set used in our prelimenary research.

Can be found at the folder path: *Page Sets and experiemnts\Page_Set1*

A.2 Confusion experiment page set

The page set for the confusion experiement is in the same folder as the results from the confusion experiement. Both the training set and the verification set.

Can be found at the folder path: *Page Sets and experiemnts\Confusion Experiment \Pages*

A.3 Accuracy experiement page set

The page set used in the accuracy experiment. Both the training set and the verification set.

Can be found at the folder path: *Page Sets and experiemnts\Accuracy results\Pages*

A.4 Structure classification page set

The page set used for the structure classification and the final experiment. Both the training set and the verification set.

Can be found at the folder path: *Page Sets and experiemnts\Extraction Experiment*

A.5 Complete page set

The complete page set that we mad

Can be found at the folder path: *Page Sets and experiemnts\Full Page set*

A.6 Prototype source and assets

Our prototype source code

Can be found at the folder path: *Project source and assets*