Segmentation and Classification of HTML Documents for Display on Small Screen Devices

Peter Kelly, B.Info.Tech.

October 2003

School of Computer Science

The University of Adelaide,

South Australia

Supervisor: Dr. Barry Dwyer

Submitted in partial fulfillment of the requirement for the Honours Degree in Computer Science

Abstract

The increase in availability of hand-held devices capable of browsing the web, such as mobile phones and PDAs, has necessitated new approaches to the presentation of web content. Pages designed for the display capabilities of traditional desktop PCs are awkward to view on low-resolution displays because of the amount of content present. This project investigates a method for transforming the structure of a web page into a format more suitable for display on such devices, by identifying parts of the document that can be removed without altering the main content of the page.

Algorithms for performing segmentation of a page into its constituent parts, and classification of each of those segments are presented. The resulting classifications determine which types of segments should remain in the page and which should be removed. A prototype implementation is described and evaluated using a set of sample web pages, and shown to produce good results on certain types of pages.

Acknowledgments

I would like to thank my supervisor, Dr. Barry Dwyer, for all his helpful advice, and for setting such high standards for my work.

I would also like to thank Dr. Paul Coddington, for his assistance in providing useful feedback on my thesis.

Thanks also to my friends and family, for providing support and encouragement during what has been a rather "interesting" year.

Contents

1 Introduction **8**
- 1.1 Motivation . . . 8
- 1.2 Overview . . . 9
- 1.3 Contribution . . . 9

2 Background and Related Work **11**
- 2.1 Mobile Web Access . . . 11
- 2.2 Content Adaptation . . . 14
- 2.3 Document Classification . . . 16
- 2.4 Web Standards . . . 18
  - 2.4.1 HTML . . . 18
  - 2.4.2 XML . . . 19
  - 2.4.3 DOM . . . 20
  - 2.4.4 CSS . . . 21
  - 2.4.5 HTML Rendering . . . 22
  - 2.4.6 JavaScript . . . 23
- 2.5 Machine Learning . . . 23
  - 2.5.1 Probability Theory . . . 23
  - 2.5.2 Bayesian Networks . . . 25

3 Design **27**
- 3.1 Segmentation . . . 28
  - 3.1.1 Rendering . . . 29
  - 3.1.2 Region Detection . . . 30
  - 3.1.3 Region Merging and Segment Creation . . . 31
  - 3.1.4 Floats and Absolute Positioning . . . 33
  - 3.1.5 Segment Representation . . . 33
- 3.2 Classification . . . 35
- 3.3 Page Extraction . . . 37
  - 3.3.1 Spacing . . . 38
  - 3.3.2 Style Rules . . . 38
  - 3.3.3 Cell Alignment . . . 38
- 3.4 Training . . . 39

4 Implementation **40**
4.1 Training . . . 43
4.2 Classification . . . 43
4.3 Measurement . . . 44
4.4 Implementation Details . . . 44
4.4.1 Segmentation . . . 44
4.4.2 Segment Analysis . . . 48

5 Evaluation **50**
5.1 Classification Accuracy . . . 51
5.2 Class Merging . . . 54
5.3 Additional Feature Variables . . . 57

6 Discussion **58**
6.1 Limitations . . . 58
6.1.1 Segmentation . . . 58
6.1.2 Classification . . . 59
6.1.3 Feature Variables . . . 60
6.1.4 Support for HTML Features . . . 60
6.2 Implementation Considerations . . . 61
6.3 Suggestions for Future Work . . . 62
6.3.1 Segmentation . . . 62
6.3.2 Classification . . . 62
6.4 Other Useful Applications . . . 63

7 Summary and Conclusions **64**
7.1 Summary . . . 64
7.2 Conclusions . . . 66

A HTML Tags **67**

B Segmentation Examples **68**

C Segment Classes **70**

D Measurement Output **71**

List of Tables

2.1 Joint probability distribution 25
2.2 Conditional probability table 26

3.1 Segment classes 35

5.1 Classification accuracy with test set = training set 53
5.2 Classification accuracy with split data sets 53
5.3 Classification accuracy with test set $\neq$ training set 54
5.4 Segment mis-classifications 55
5.5 Classification accuracy after class merging 56
5.6 Classification accuracy after class merging with split data sets 56
5.7 Classification accuracy after inclusion of spatial information 57
5.8 Classification accuracy after inclusion of spatial information with split data sets 57

List of Figures

1.1 PDA and desktop web browsers 9

2.1 WEST Zoomed-out view [44] 12
2.2 RSVP browser [9] 13
2.3 Power Browser [3] 13
2.4 m-Links [41] 14
2.5 Example HTML document 19
2.6 Example XML document 20
2.7 Example DOM tree 21
2.8 Rendering tree 23
2.9 A simple Bayesian network 26

3.1 The HTML rendering process 29
3.2 Grid cells with possible region configurations 30
3.3 Grid cells and inline boxes, with the chosen region configuration 31
3.4 Distinct regions requiring merging 32
3.5 Multiple repeating instances of title/body region pairs 33
3.6 Segment nodes with an excluded float 34
3.7 Naive Bayes classifier 36

4.1 The prototype system 41
4.2 Available display modes 42
4.3 Segment identification 47

Chapter 1

Introduction

This project investigates an approach to displaying web pages on low-resolution displays. It includes a discussion of the problems associated with providing a good user experience in spite of limited hardware capabilities, and builds on previous efforts that have been taken in the area to present new ideas about how to solve the problem.

1.1 Motivation

Access to the web from portable, hand-held devices is becoming increasingly popular. Many users are taking advantage of technologies such as mobile phones and PDAs to view information when away from their offices and homes. Reading the latest news, catching up on sports results, chatting with friends, researching information, and other popular on-line activities are now commonly done while traveling, having lunch, or waiting at airports. However, since the devices used for mobile web access are relatively limited in terms of their screen size, computing power, and user interfaces, the user experience is substandard compared to that provided by desktop PCs.

Working around these limitations promises to be beneficial to many users by improving ease of use and thus reducing the amount of time taken to find and access information. One of the ways in which this can be done is by providing improved user interfaces which are more suited to low-resolution devices, and by presenting information in a way that is less cluttered and more focused toward user needs.

Web page design and web browsing technologies have traditionally been geared toward desktop PCs, which have powerful processors, fast network connections and high-resolution displays. As a result, the vast majority of web sites on the Internet are designed such that they are more suited to screens that can display large volumes of content. Attempting to view these pages on a low-resolution display is at best awkward, requiring a significant amount of of horizontal and vertical scrolling to view the content.

Figure 1.1 shows a typical web page displayed on both a PDA with a 240×240 screen and a desktop computer with a 1024×768 screen. The PDA displays only a small portion of the page, consisting mainly of the site logo and navigation menu. In order to view the actual content, the user needs to scroll both across and down. Even when the main body of the story is in view, the column is almost, but not quite narrow enough to fit on the screen. This means that the user not only has to scroll down the page as they read, but also across and back for each line, since the last few letters of the column are cut off. This is not helped by the fact that the page in question has been designed for a fixed screen width of 800 pixels, as evidenced by the desktop screenshot in which it remains at this width despite the window having more room for content. Fixed-width layouts are not uncommon, particularly on many larger, commercial sites. These layouts are sub-optimal for any screen resolution other than that for which they were designed.

Many web developers have responded to these problems by creating separate versions of their sites targeted specifically at small screen devices. So-called “wireless” or “mobile” editions are much easier to read and navigate, and present only the main content of the page without extra components such as sidebars and large navigation menus.

Figure 1.1: PDA and desktop web browsers

While these efforts are useful in themselves, since they greatly enhance usability for mobile users, they require significant manual effort on the part of the developer. In some cases, this extra effort is avoided by having the web server automatically convert documents based on requests from the client. Documents are stored in a standard format which is independent of presentation, and then transformed to an appropriate representation using technologies such as XSLT [48] or custom scripts specific to the website or content management system in use. However, this is still not a general solution to the problem, because it only works for the relatively small number of sites which have been built with this in mind.

It is clear that different approaches are needed that take into account access to the large base of existing content, and do not require extra effort on the part of page authors. The most realistic and scalable way of dealing with the problem is to build systems which either modify pages or present them in a manner which is better suited to low-resolution screens. This project is one of many that have attempted to tackle the problem, and presents some new ideas which, when combined with existing technologies, can be used to enhance the usability of the web for mobile users.

1.2 Overview

Chapter 2 provides an introduction to the concepts used in this project, and discusses the capabilities of currently available technologies. Chapter 3 describes the algorithms developed as part of this project and Chapter 4 details their implementation in a prototype system. In Chapter 5, an evaluation of the prototype is given. Chapter 6 gives a discussion of the results and how the outcomes fit within the context of other work. Finally, Chapter 7 summarises the work performed and what has been achieved.

1.3 Contribution

The aim of this project is to develop and evaluate a method of effectively displaying web pages on low-resolution displays. There has been a significant amount of effort in this field in the past, and many of the ideas investigated

in this project have been influenced by this work. While the specific algorithms introduced here are new, this past research has provided a good background from which to design new techniques.

The approach taken to the problem is to perform *segmentation* and *classification* of web pages, and use the results of this to alter the display. Segmentation is the process of splitting up a page into multiple distinct sections, each containing a separate piece of content. Many pages are structured such that they have not only a main body of text relating to the central purpose of the page, but other elements which serve a peripheral purpose. By examining each of these segments and working out which type of content each contains, decisions can be made about what to present to the user. The classification is based on information from a training process which has been performed in which sample web pages and segments are supplied to the system with classifications assigned by a human, and this information is used to decide whether or not to present certain segments to the user when viewing a pages.

This is a new approach which, while widely used in the field of OCR (Optical Character Recognition), has not been applied to the problem of web page display up until now. The intention of this project is to present a new segmentation algorithm and means of classification which are appropriate for use on the wide variety of pages available on the web.

A major part of the project is the implementation of these algorithms in a prototype system capable of loading and displaying web pages. This involves integrating the techniques introduced here with parts of an existing web browser, and fine-tuning the algorithms based on experiences with testing on various web pages. The aim of developing such a system is to determine if the ideas work when applied to real-world web pages, and to identify any problems that arise in practice.

Also central to the investigation is an evaluation of how accurate the classifications preformed by the system are. By providing a thorough investigation into the effectiveness of the algorithms, an attempt is made to demonstrate both the capabilities of the system, and to identify areas in which improvement is needed. By finding any shortcomings inherent in the techniques, further work necessary for effective implementation can be identified.

Chapter 2

Background and Related Work

2.1 Mobile Web Access

Many different approaches have been taken to the problem of providing efficient interfaces for accessing the web on small screen devices. These vary in many ways, including the user interface and presentation of content to the user, the sources of information relied on, and the methods by which web content is modified for display. This section summarises a number of approaches taken in the area, to give a background on some of the techniques that have found to be useful. Many of these ideas have served as a basis for the thinking behind various aspects of the design undertaken as part of this project.

A number of specialised markup languages have been developed for use on mobile phones. The most well-known of these is WML (Wireless Markup Language) [45], part of the WAP protocol suite. It operates based on the model of *decks* and *cards*. A deck is a single resource retrieved from a web server and is analogous to a HTML file in a normal web browser. Each deck may contain multiple cards, each of which represents a separate screen. Facilities are provided for navigating between cards and decks.

WML is primarily text-based but does include some support for images. However, due to the limited bandwidth and screen size of the target devices, these are not used extensively. Support is also provided for WMLScript, a scripting language similar to JavaScript, which allows for additional interactivity and extra processing such as form validation.

WAP has been the target of much criticism throughout the industry due to usability issues [37], security problems [30], lack of compatibility with other standard protocols and differences in implementations between vendors. More recently, efforts have been undertaken to bring the protocol suite more in line with common standards. WAP version 2.0 deprecates WML and instead uses XHTML Mobile Profile [46], which defines a restricted set of tags for documents that can be displayed by devices with limited capabilities. This is advantageous because it is closer to the format used for normal web pages, however it is still limited in that pages to be displayed on WAP 2.0 devices still cannot use the full range of tags, so access to the general web is still not possible. However, since the set of tags in XHTML Mobile Profile is a subset of those defined in the HTML specification, these documents can be viewed on normal desktop browsers, which is not possible with WML.

As mentioned previously, the use of specialised markup languages or subsets of HTML is not a general solution to the problem since documents must still be authored or converted to the target language. For this reason, a number of projects have looked at ways of using alternative interfaces to view web pages on small screen devices. All of these perform some sort of transformation of the page either on the client or an intermediate proxy server in order to get the content into an appropriate format or display, either by modifying the page to be displayed, or converting it into another structure more suited to the interface presented.

WEST (WEb browser for Small Terminals) [44] is web browser designed for low-resolution displays. It uses a

Figure 2.1: WEST Zoomed-out view [44]

focus+context visualisation technique called Flip Zooming [12] to allow easy access to different parts of the page. Upon loading the page, it is split up into a series of *cards*, each of which contains a portion of the content. The primary mode of the interface consists of a series of thumbnail images of cards. Once a card is selected, the display zooms in to show it at full size. This allows the user to get an overall view of the page, and then view the parts of the page they are most interested in.

In the zoomed out view, the user can choose to see different representations of each card, other than just thumbnails. Other views available are the keyword view, which shows keywords selected from the card, and a link view, which displays each of the links extracted from the card. Figure 2.1 depicts the zoomed-out view provided with keywords for each card shown.

User testing found this approach to be significantly easier to use than a traditional browser interface running in the same low resolution. The navigation mechanisms provided are more suited to the small screen due to the fact that they provide more efficient access to different parts of the page, and allow users to get an overall view of the page without having to scroll.

The RSVP browser [9] provides a novel approach to exploring links on a page. Instead of reading through the page and looking at the content of each link, the user is presented with a thumbnail image of the page pointed to by each link destination. These appear in sequence in a slide show fashion, with the intention of giving the user a brief look at each link destination so that they can make a decision about which page they want to visit. The slide show runs automatically but can be paused by the user if necessary. This method cuts down on the amount of interaction that the user has to perform to view the different links. Figure 2.2 shows a screenshot of the browser, in which the destination page of a link is shown in the middle of the window with buttons for each destination down the side.

Power Browser [3], shown in Figure 2.3, is a web browser developed for the Palm Pilot which focuses on providing efficient navigation of websites. Its interface model is based on the observation that most web browsing is split into two phases: *navigation* and *end-game* browsing.

In the navigation phase, the user is looking for a page containing particular information. Typically this involves traversing a series of links through different pages, at each stage getting one step closer to the target. During navigation, the user focuses on links rather than the content. In navigation mode, Power Browser displays a tree structure containing the links on each of the pages visited in the session, without displaying the content. This greatly reduces the screen space required to display the pages, and makes navigation among links in a page much easier.

Once the user has reached their destination, the browser goes into end-game mode. This is where the actual content of the page is displayed, since the focus is now on viewing information rather than finding it. A technique called *accordion summarisation* [4] is used to display the page in a manner suitable for the low resolution display. The page is divided up into a hierarchical structure of *semantic text units*, usually individual phrases or sentences, and the

Figure 2.2: RSVP browser [9]

Figure 2.3: Power Browser [3]

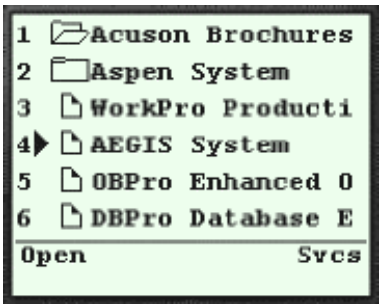


Figure 2.4: m-Links [41]

structure is shown as a tree on the display. Initially, only top-level nodes are shown, and the user can expand these to see more detailed parts of the page. This interface is very similar to that provided for navigation, and performs essentially the same function, except that content is displayed instead of just links, and the items displayed are within the context of a single page.

A similar navigation+usage model is used by the m-Links project [41]. This is designed specifically for mobile phones with text-based displays, where only a few lines of text can be shown at a time. Instead of displaying the page in the traditional manner, it inspects the page and extracts links and other relevant data items such as phone numbers and addresses, which it then presents in a menu. A user can navigate to a different page by selecting a link from the menu, or can view the various data items and perform actions on them such as calling calling a phone number or faxing a document. A screenshot is shown in Figure 2.4.

The extraction of data items and links is performed by an intermediate proxy server, which also implements functionality to support various user actions in relation to data items or links. For each menu item, there is a series of actions that can be performed, including emailing or faxing the contents of a link to a particular destination, or viewing the item on screen.

This user interface model is seen as more appropriate because it is very familiar to mobile phone users. The menu-based navigation is easier to use on such interfaces, and displaying the page as a series of small lines of text is also more appropriate for traditional mobile screens. However, this approach is becoming less relevant, as many new mobile phones available today have higher resolution screens and better navigational facilities, a trend which is likely to continue at a rapid pace.

One approach proposed early on in the evolution of the web was to use *dynamic documents* for adapting to mobile displays [17]. These are scripts written in TCL which are downloaded to the browser and then executed. Extra hooks are provided into the TCL interpreter to provide the scripts with information about the display resolution and other capabilities of the client, which can be taken into account by the scripts to generate a page with appropriate layout and content.

While this approach enables documents to be tailored specifically for certain devices, it has poor portability because the script authors must have prior knowledge of all devices on which the script will run. When new types of clients appear, pages designed in such a manner will either assume the wrong type of client or not work at all. Additionally, this approach is based around the model of a web page being a script that is executed on the client as opposed to the more conventional model of a static piece of content.

2.2 Content Adaptation

Rather than relying on content targeted specifically for mobile devices, or specialised interfaces for viewing pages, another approach that has received widespread attention is *content adaptation*, which is the presentation of content

to user in a modified form. The modification process, called *transcoding*, takes the original document and changes or removes certain parts of it to make it more suitable for display on the target device. Other applications of content adaptation include merging of contents from external sources, and presenting documents in a more readable manner for accessibility purposes. There is a wide range of types of transcoding, each of which serves different purposes. Some of the more common types are:

- *Page reformatting*

  Page reformatting changes the layout of the page in a way that makes it more appropriate for display on low-resolution displays, while retaining all of the original content. This includes the adjustment of font sizes, spacing, and the position of various objects on page.

  An example of page reformatting is the small screen rendering feature of the Opera web browser [33]. This reformats layout tables so that all cells are shown vertically rather than in multiple columns, reducing the width of the page in order to eliminate the need for horizontal scrolling.

- *Text summarisation*

  Some systems use *text summarisation* to reduce the total amount of text shown on screen. This involves replacing the original text with summarised versions which convey the same or similar meaning in fewer words. Some systems rely on simple techniques such as displaying the first sentence of every paragraph, or displaying section headings along with links to the content of each section. More advanced techniques have also been used, such as natural language processing which determines the semantics of each block of text, and generates a summary for each.

  A significant drawback of text summarisation techniques is that they usually retain text elements such as navigation information and copyright statements, which are not necessarily relevant to the main content of the page. They also leave the layout in place, which may mean that the page still does not fit on the screen, requiring horizontal scrolling.

- *Image reduction*

  Image reduction is the process of resizing images to take up less area on the page. This is a very simple and effective technique, since many pages contain a significant number of images, some of which can be fairly large. In addition, most images can be reduced in size without losing their meaning.

  In some cases, images can be removed from the document if they are deemed to be unimportant. This is appropriate in cases where a page can still be understood without images, or if the images are only present for decorative reasons. Image classification techniques can be used to decide whether or not certain images should be kept or removed. Previous research has shown good results in classifying web pages images based on various aspects such as file format, text presence, edge characteristics, colour distribution and size [34, 14].

  Another type of image reduction is using higher levels of compression when transmitting images to a client, when the images are requested through a proxy server. While this does not affect the amount of room taken up on screen, it reduces the transfer costs because the image files are smaller. Taking into account the characteristics of images in terms the savings that can be gained from such compression allows proxies to efficiently reduce the amount of traffic to clients [6].

- *External annotation*

  Instead of making modifications based on intrinsic parameters of the transcoding algorithm or decisions derived from the original documents themselves, external annotation [28] relies on information obtained from a source independent of the original document. This may come as separate file on the same web server, custom settings stored on the client, or annotations stored on another server [31]. Systems supporting this functionality present documents in a modified form based on these annotations, representing a value-added service to the user, since

they can view additional information beyond that supplied by the original author. This extra information can also be used to encode the semantic meaning of content, which can be of use when making decisions about how to transcode a document [32].

Document modifications based on such information have a range of uses beyond small screen browsing. They can be used to support collaborative research among groups, editorial corrections to documents, and additional discussion about the topics of documents [38].

One of the most common uses of annotations is to change the layout of the document to give a different visual appearance. This can be achieved using Cascading Style Sheets (CSS), which specify the appearance of elements in a document [51]. The HTML specification defines several ways of including or referencing a style sheet from a document, and often a web page will include the URLs of one or more other style sheets stored on the same site, which are to be retrieved by the browser and used to format the page. Some browsers also allow the user to specify their own custom style sheet to be used with all pages. This allows the page display to be tailored to the user's needs, such as by using large fonts and high contrast colours for the visually impaired.

- *Feature selection*

  Feature selection, the method used in this project, is the process of inspecting the page and deciding which parts to keep and which parts to remove. It enables the reduction of content by removing unnecessary or irrelevant sections of the page. Approaches to feature selection include the removal of images and selection of certain sections.

The part of the architecture which performs the transcoding process is an important consideration. Transcoding can be performed either by the client itself, or on an intermediate proxy server. In the latter case, all access to web servers goes through the proxy, which retrieves documents on behalf of the client, and returns modified versions of the documents.

Transcoding can also be performed by the web server itself, in situations where client browser detection is used to serve up a document in an appropriate format. However, this approach only solves the problem for the sites that implement it, and most content adaptation systems focus on solutions that allow access to the whole web and do not require any changes to the web server.

The proxy server approach is advantageous if the client only has a low bandwidth connection relative to that of the proxy, as is the case for most mobile connections, and if the amount of content sent from the proxy to the client is significantly less than that sent from the web server. The latter is true if significant parts of the page are removed, or images are sent back at a lower resolution or higher compression level. This is the most popular approach, because it can be implemented as a separate system and does not require any modification to the client's web browser.

Implementing transcoding directly on the client provides greater flexibility in terms of the user interface functionality provided. If the client allows the user to adjust the way in which modifications are performed, then the transcoding has to be re-done whenever those preferences change. If implemented on the client, this does not require any extra requests to the proxy, thus making the user interface more response to such changes. Additionally, other aspects of the user interface can reflect information obtained during transcoding, such as additional menus and navigation controls which are separate from the main page.

Client-side transcoding is a more secure option, since it allows the content to be transferred directly from the web server to the client using an encrypted connection, without any intermediate party having access to the content. Lack of security inherent in proxy-based architectures has been the subject of much criticism since it is inappropriate for the demands of e-commerce [8].

2.3 Document Classification

Document classification is the problem of assigning a *class* to a document based on its contents. The meaning of the class can take a variety of forms, such as category, purpose, topic, or some other semantic. The ways in which the

content is used in making the decision, and the mechanisms by which classes are assigned have been the subject of a large amount of research. This section provides a brief introduction to the area, and describes some of the techniques that can be used for classification.

The design detailed in Chapter 3 discusses the classification of individual parts of documents, called *segments*, rather than entire documents themselves. Much of the literature in the field, and the overview given here, considers the case of assigning one class per document. However, the concepts and algorithms are the same, and when considering the implementation of segment classification described later in this thesis it is helpful to think of each segment as being a separate "document" for classification purposes.

When performing classification, a number of different pieces of information about the document, called *feature variables*, are taken into account. The way in which these are derived from the document varies greatly between systems, and different approaches give varying levels of effectiveness.

Using word frequency measures, common in text classification, feature variables are based on the number of instances of each word in the document. Often, dimensionality reduction is applied, taking into account only a subset of the words in the document, and reducing the computational cost of training and classification [20]. Word counts are either measured in ranges, or as binary-valued variables indicating the presence or absence of each word [40]. Feature variables based on other information such as word semantics, derived from the text using natural language processing techniques, have also been investigated, although do not appear to be more effective than word frequencies [42].

The approach taken in this project is to base the classification mainly on geometric layout information obtained from the document. Previous research has investigated this approach and proved it to be useful in classifying documents [13]. While geometric analysis is useful for determining the purpose of a document, it is not appropriate for determining semantic meaning, since it does not take into account the content of the text. Since it is the purpose of document segments that is important for the content adaptation techniques implemented here, this is a useful way of classifying segments.

The most common approach to document classification is the use of machine learning techniques. A *classifier* is built from the analysis of a manually classified training set, allowing automatic classification to subsequently be performed. A wide range of machine learning techniques are applicable for this purpose; a good overview describing many of these is given in [43].

A *decision tree* classifier [1] consists of a series of nodes corresponding to feature variables. The branches between parent and child nodes are derived from the allowable values of the parent node. A document is classified by starting at the top of the tree and working downwards, at each point moving along the branch that represents the value of the current node's variable in the document being classified. Upon reaching a leaf node, the class of the document is chosen based on the class associated with the leaf. Decision tree variables are usually binary, in that the decisions are "yes or no" values based on whether or not a document has a particular value. In contrast to many other machine learning algorithms, decision trees produce classifiers that can be easily interpreted and understood by humans [26].

K-nearest-neighbor classifiers [25] operate by comparing the document to be classified with the members of the training set most similar to that document. The class picked is the most common class of the nearest documents. In some cases, the class assignments are weighted according to the distance between the target document and each training example; documents that are more similar are weighted higher in when deciding the class.

Support Vector Machines (SVMs) [16] represent documents as points in n-dimensional space, where n is the number of feature variables. Once constructed, a SVM contains a plane through the space which divides the points in the training set in two. The points on one side of the plane consist mostly of the training examples that match a particular class, while the points on the other side are mainly examples which are not in that particular class. The plane is chosen such that the maximum number of training examples are on the correct side. During classification, the location in n-dimensional space at which the document in question resides determines whether or not it is assigned the class. SVMs give yes or no decisions about whether a document is in a particular class; they do not support the selection out of a group of classes. They have the significant advantage that they do not require complete re-training as new documents are added to the training set, and do not rely heavily on dimensionality reduction [23].

The *Roccio method* [15] is similar to SVMs in that it uses the same n-dimensional representation of documents,

and operates on a yes/no decision about a class. During training, it calculates two points corresponding to the centres of the positive and negative samples. To classify a document, the distance between its point in space and each of the two centres is calculated, and the class is selected based on a comparison of the two distances. The Roccio method is often used as a baseline when evaluating the performance of other classifiers. A drawback of the Roccio method however is that it performs poorly when the negative training samples are widely dispersed within n-dimensional space; it is most efficient when all of the positive samples are closely related.

Bayesian classifiers [24], used in this project, consist of a network of nodes, each of which represents a feature variable. A series of causal connections exists between nodes, and probability information is recorded in the network, which stores the likelihood of variables having certain values given knowledge about the values of other variables. These have been found to be highly effective in many areas of document classification [29]. A more detailed explanation of Bayesian networks is given in Section 2.5.2.

2.4 Web Standards

This section gives an overview of the web standards and technologies used in this project. While a full description of each is beyond the scope of this work, the aim is to give a sufficient explanation necessary for understanding the way in which they are used in the design and implementation described here. The focus is mainly on how these technologies are implemented, rather than how they are used, since implementation details are most relevant in this context.

2.4.1 HTML

HTML, or *Hypertext Markup Language [47]*, is the standard language in which all web pages are written. Its simple text-based format is easily to learn, which has contributed to its popularity and widespread usage.

A HTML document consists of a body of text interspersed with markup in the form of *tags*. The text itself is displayed directly on screen, while the tags provide additional instructions on how to format the text, and where to insert special objects such as images and form controls.

A tag consists of a name and a series of name/value pairs called *attributes*, surrounded by a pair of brackets. For example, a paragraph in which the text is to be centre aligned is written as:

```
<P ALIGN=”center”>
```

Many tags require a corresponding close tag to indicate the point at which their content ends. In some cases, such as the `P` tag, the close tag is optional and is implied by the beginning of another tag. Close tags begin with a slash (/) and do not have attributes:

```
<P ALIGN=”center”>
This is the content of the paragraph
</P>
```

Other tags do not require a close tag, such those that represent an object being inserted into the page instead of surrounding a body of text.

```
<IMG SRC=”logo.gif”>
```

The HTML specification defines the set of allowable tags and attributes. A large number of these are provided, covering a range of functionality, including text formatting, tables, lists, hyperlinks, images and forms.

Each HTML document consists of two main parts: the head and the body. The head consists of metadata about the document, such as the title, base URL, style sheet references and other custom metadata. The body contains the actual text to be displayed on screen. An example HTML document is shown in Figure 2.5.

```
<HTML>
  <HEAD>
    <TITLE>HTML example</TITLE>
  </HEAD>
  <BODY>
    <P>
      This is an example of a HTML document.
    </P>
    <P>
      Please visit our <A HREF="http://website.com">home page</A><BR>
      <IMG SRC="logo.gif">
    </P>
  </BODY>
</HTML>
```

Figure 2.5: Example HTML document

A piece of software capable of loading and processing a HTML document is referred to as a *user agent*. Web browsers are the most common type of user agent. Other types also exist, such as search engine spiders and web page editors.

The process of rendering a HTML document on screen involves a number of steps, described in Section 3.1.1. These make use of information contained within the page itself, as well as from external sources such as style sheets, described in Section 2.4.4.

A HTML document may also have one or more scripts associated with it, either included within the document, or referenced as an external file. Scripts add interactivity to a page through various mechanisms such as changing status bar text, creating pop-up windows, or manipulating the document structure. The most widely supported scripting language is *JavaScript*, described further in Section 2.4.6.

2.4.2 XML

XML, or *Extensible Markup Language* [49], is similar to HTML in many respects, but is designed as a more general purpose language. It has a wider range of uses beyond that of document representation, and in recent years has become popular as a means of storing information in a portable manner for exchange between different applications.

While HTML defines a restricted set of tags and attributes with specific meanings, XML allows any names to be used. It can therefore be used to store any type of information, and the set of tags used in an XML document depends on the needs of a particular application. One common use is for representation of information stored in a database, as shown in Figure 2.6.

```
<xml version="1.0">
  <staff>
    <employee name="Fred">
      <position>Manager</position>
      <salary>60000</salary>
      <hiredate>19960403</hiredate>
    </employee>
    <employee name="Joe">
      <position>Programmer</position>
      <salary>40000</salary>
      <hiredate>19990201</hiredate>
    </employee>
  </staff>
</xml>
```

Figure 2.6: Example XML document

XML also has a number of syntactic differences with HTML. Tag and attribute names are case sensitive and usually specified in lower case. Each open tag must also have a corresponding close tag, or must include the slash at the end:

```
<story>Once upon a time...</story>
<goodnight />
```

Since XML is a generic language and not aimed specifically at document representation, there are no built-in rules dictating how to display a document. The above example could be displayed simply as a long run of text containing all of the words between tags, or as a table with rows corresponding to employees and columns for each of the details. The presentation used is application specific — if indeed, the data is to be displayed at all. It may instead be analysed or stored in a database rather than being made directly visible to a user.

When loaded by a web browser, XML documents can be formatted using style sheets. These are used in much the same way as for HTML documents. The use of style sheets is covered in Section 2.4.4.

2.4.3 DOM

The *Document Object Model* (DOM) specification [50] provides a number of classes and programming APIs for dealing with the contents of a HTML or XML document. These enable a program or script to manipulate a document in memory in various ways such as adding or removing content.

When loaded into memory, a document is represented structurally as a tree of nodes. A number of different types of nodes can be present in the tree; the two of interest for the purpose of this discussion are *Element* nodes and *Text* nodes. An Element node represents an open tag/close tag pair in the source file, or just the open tag if the latter is not present. Each element has a name and a set of attributes associated with it. A Text node represents a sequence of characters residing between tags in the source file.

Element nodes may have any number of child nodes in the tree, which can be Text nodes or other elements. The set of nodes under a particular element corresponds to the portion of the source file between the open and close tag of the element. Text nodes cannot have children — they are only ever leaf nodes in the tree. An example DOM tree, corresponding to the HTML file above, is shown in Figure 2.7. Grey boxes in the figure correspond to text nodes, and white boxes represent elements.

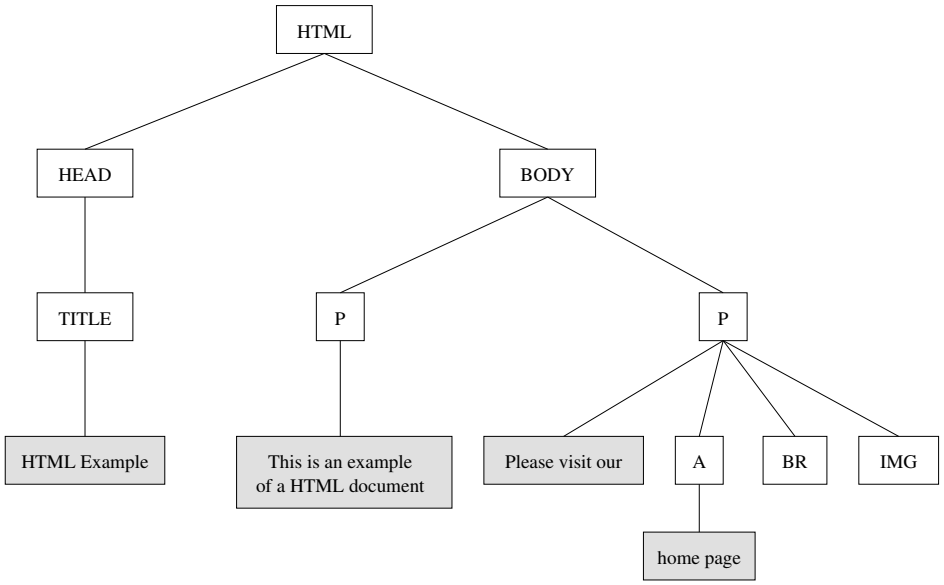


Figure 2.7: Example DOM tree

In order to build a DOM tree from a document, a user agent must first parse the source HTML or XML file. This involves reading in all of the tags and text and constructing the node structure based on the contents of each tag. Once the tree is in memory, it can be modified by the application or used for other purposes such as information extraction or rendering of the document.

Many web browsers allow scripts included in the document to access and manipulate the tree. Once loaded, a page can modify itself in response to user actions. Many websites use this ability to implement user interface features such as drop-down menus and image roll-overs.

2.4.4 CSS

Style Sheets are a mechanism for providing style information for a document. They are useful for separating content from presentation by allowing the appearance of various parts of the document to be specified separately from the actual content itself. The most common form of style sheets used on the web today is *Cascading Style Sheets*, or CSS [51].

CSS style sheets consist of a number of style rules which specify values for display properties that apply to certain elements in the document. There are many different properties that can be specified, including font size and colour, paragraph borders, margins, and other aspects of visual appearance. An example style rule is given below:

```
P { width: 80%; color: red }
```

This states that all `P` elements in the document should take up 80% of the width of the page, and that all text within the paragraphs should be drawn in red. Style rules can also match based on the `CLASS` attribute of an element. The following style rule specifies that all paragraphs with a class of `abstract` should be centre aligned:

```
.abstract { text-align: center }
```

Paragraphs can then be designated as abstracts, causing the properties in the style rule to be set:

```
<P CLASS="abstract">
```

There are numerous other ways of matching property assignments to elements such as the matching of other attributes, and particular types of relationships between elements.

In addition to using style rules, properties can also be set on individual elements. This follows a similar syntax to style rules with the exception that the element name and brackets are omitted, since the element to which the property assignments applies is already known from the context:

```
<DIV STYLE="font-size: 12pt; background-color: blue">
```

All style rules are written in a style sheet. This is either included directly within the head section of the document, or stored in an external file and referenced from the document. The latter usage allows a single style sheet to be used for multiple documents, for example when a consistent visual appearance is desired for all documents on a website.

The end result of applying a style sheet to a document is that for each node in the DOM tree, the value of every display property is known. For XML documents this information is derived solely from the style sheet. For HTML documents, the property assignments are also based on built-in rules included in the HTML specification, and the values of attributes which have special meaning in HTML. For example, consider the following paragraph tag:

```
<P ALIGN="center">
```

The resulting `P` node in the DOM tree would have a value of `1.33em` for the `margin` property, which is specified in the built-in HTML style rules. It would also have a value of `center` for the `text-align` property, since this corresponds to the `ALIGN` attribute.

2.4.5 HTML Rendering

The mechanisms by which pages are rendered differs greatly between web browsers. However, the nature of the process is determined to a certain extent by the structure and rules included in the CSS specification about how document layout occurs. The size, position and presentation of document elements is determined by the properties set on elements int he document, and the spatial relationships between them are affected by their relationships in the DOM tree and certain other properties. The HTML and CSS specifications only specify required behaviour, not internal data structures or algorithms. The rendering model described here is a combination of the requirements of the specifications and the way in which they are implemented in the rendering engine used by the prototype.

The central aspect of the rendering model is the box model. The rendered output consists of a hierarchy of boxes, each of which has its position and size calculated during the layout process. Each box has a set of properties associated with it, derived from those of the corresponding element, and can contain other boxes. The rendered output can thus be represented as a tree, with the highest node corresponding to the document element, and child boxes representing each node in the tree, with some possible minor structural differences.

When laying out a document, the renderer creates a set of rendering objects, one per element [1]. These relationships between these objects closely mirror those of nodes in the DOM tree. The type of rendering object created for a given node is determined by the `display` property. The other properties associated with the node are also set on the rendering object and used for layout calculations and drawing purposes.

There are two main types of rendering objects: *Block* and *Inline*. Block objects are displayed as a single box, and have a specific position and size. They always take up the full available width within the portion of the document within which they reside, unless a different width has been specified via a CSS property. Inline objects can be split over multiple lines, in which case they result in multiple boxes being drawn. The most common example of an inline object is a text node, which can be split at word boundaries to cover several lines. Block objects can contain both block and inline objects, but inline objects can contain only other inline objects.

[1] There are some exceptions to this, e.g. when `display: none` is used to prevent an element from appearing in the rendered output.

An example of a rendering object tree and the corresponding set of boxes is shown in Figure 2.8. Each object is labeled with its type: BC = Block container, IC = Inline container, T = Text, LB = Line break, I = Image. The boxes created during the layout process are shown to the right. Note that no rendering objects are created for the `HEAD` element or its descendants, because these do not form part of the document content, and have a `display` property of `none` implicitly assigned to them.

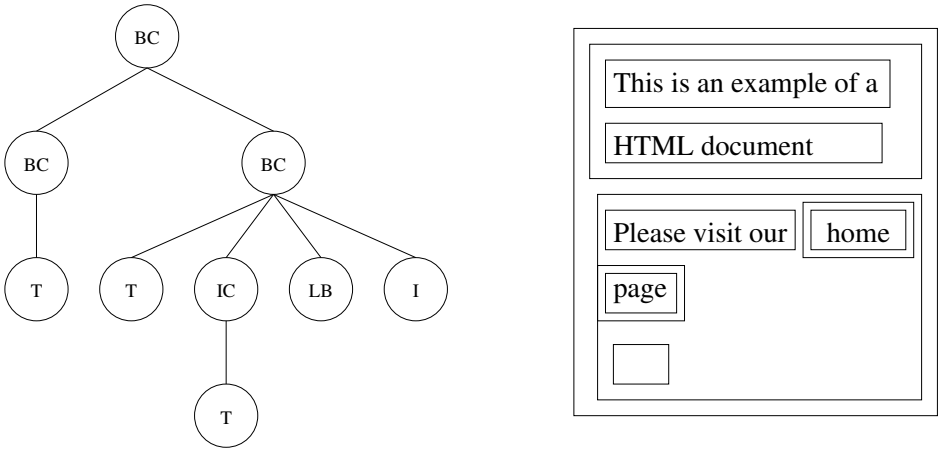


Figure 2.8: Rendering tree

2.4.6 JavaScript

JavaScript, also known as *ECMAScript* or *JScript*, is a scripting language that is commonly used in web pages. It provides access to a set of functions and objects which can be used to manipulate the contents of the page, and interact with the user in various ways.

The language itself is a fairly typical object-oriented style programming language. It provides a number of standard facilities such as expressions, functions, variables, literals as well as built-in routines for things like string manipulation and mathematical calculations. It has also been used widely outside the realm of client-side web scripting for other purposes, such as server-side scripting and application automation.

JavaScript code can either be included within a web page, or stored in an external file and referenced from within a page. The code is typically executed in response to user actions such as mouse movements or button presses. The actions performed by the code are determined by the page author, and may include things like bringing up new windows, changing values of form fields, displaying dialog boxes or more complex actions such as adding or changing the page contents via DOM tree manipulation.

2.5 Machine Learning

2.5.1 Probability Theory

In order to provide an understanding of the classification process described later in this document, it is necessary to first develop an understanding of the theories and concepts that underly the discussion. Central to this is probability theory, used widely in the area of machine learning and in particular Bayesian networks, which are used in this project. While more detailed introductions are available in other sources [39], the basic concepts are summarised here.

The probability of some expression A being true is expressed as:

$$P(A)$$

For example, a degree of belief that the weather will be sunny today can be written as:

$$P(Weather = Sunny)$$

This probability is a value between 0 and 1. 0 means that the expression is known for certain to be false, 1 means that the expression is known to be true, and values in between represent varying levels of certainty. If there is 50% certainty that the weather is sunny then the value of the above expression would be 0.5.

In many cases, this probability is expressed relative to some other information that has been observed. This is called a *conditional probability*, and expressed as:

$$P(A \mid E)$$

where E denotes the *evidence* obtained in relation to the situation. For example, to express the degree of belief that the weather will be sunny today given that the temperature is hot, the following would be written:

$$P(Weather = Sunny \mid Temperature = Hot)$$

There are a number of basic rules of probability. Chief among these are the 3 axioms of probability:

1. $0 \leq P(A) \leq 1$

   States that the all probabilities are measured as a real value between 0 and 1.

2. $P(True) = 1$ and $P(False) = 0$

   A probability of 1 indicates that the degree of belief is total, i.e. the expression is known for certain to be true. Likewise, a value of 0 indicates that the expression is known to be false.

3. $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$

   The probability of either A or B being true is the sum of the individual probabilities of A and B, minus the probability that both are true.

The product rule specifies a relationship in which two expressions are true. This is defined as the probability that one of the expressions is true, multiplied by the probability that the other expression is true given the first one.

$$P(A \wedge B) = P(B)\,P(A \mid B) = P(A)\,P(B \mid A)$$

Continuing the previous example, consider the case where the probability of it being hot on a particular day is 0.4. Based on evidence collected in the past, it is also believed that if it is a hot day then there is an 0.8 chance of it being sunny. Substituting A for *Weather* and B for *Temperature* in the above example, the likelihood of it being both hot *and* sunny on a particular day is given by:

$$\begin{aligned} P(Sunny \wedge Hot) &= P(Hot)\,P(Sunny \mid Hot) \\ &= 0.4 \times 0.8 \\ &= 0.32 \end{aligned}$$

Using axiom (3) from above, and additional knowledge about the likelihood of it being sunny on any given day (say 0.6), the probability of it being either sunny *or* hot (or both) can also be determined:

$$\begin{aligned} P(Sunny \vee Hot) &= P(Sunny) + P(Hot) - P(Sunny \wedge Hot) \\ &= 0.6 + 0.4 - 0.32 \\ &= 0.68 \end{aligned}$$

Another rule commonly used when determining probabilities is Bayes' rule, which can be derived from the product rule. This allows the probability of an observation A given certain evidence B to be calculated from the individual probabilities of the observation and evidence, combined with the probability of the evidence given the observation:

$$P(A \mid B) = \frac{P(A)\,P(B \mid A)}{P(B)}$$

2.5.2 Bayesian Networks

When constructing probabilistic models of real-world systems, there are generally a number of variables involved which represent different pieces of information. For example, when modeling weather conditions, the set of variables used might be *Humidity*, *WindSpeed*, *Temperature*, *Rainfall* and *Weather*. Each of these corresponds to data that can be observed from the natural environment. When attempting to calculate the probability of a particular variable assignment based on observations of other variables, it is necessary to decide which variables to take into account and how.

The simplest way of doing this is to use the *joint probability distribution*, which specifies the probability of each possible combination of values. Supposing that *Rainfall* had two allowable values and each of the other variables had three allowable values, this would mean that joint probability distribution would consist of $2 \times 3 \times 3 \times 3 \times 3 = 162$ different combinations. A table such as the one below representing all of these combinations would thus have 162 different rows, and a large number of samples would need to be taken from the environment to get an estimate of the probability of each different combination.

Humidity	*WindSpeed*	*Temperature*	*Rainfall*	*Weather*	Probability
High	High	Hot	No	Windy	0.004
High	Medium	Warm	No	Sunny	0.031
Medium	Medium	Cold	No	Rainy	0.002
Low	Medium	Hot	No	Sunny	0.152
Low	Low	Hot	Yes	Rainy	0.091
...	...	...	...	...	...

Table 2.1: Joint probability distribution

In many cases, not all of the variables have direct relationships with each other. Modeling systems based on the assumption that every variable is dependent upon every other variable is computationally expensive, because the number of different combinations of possible values makes training and inference very time consuming, and requires a lot of data to be collected to cover all possible cases. More commonly, each variable is assumed to be dependent upon only a subset of the other variables.

A *Bayesian network* is a structure which encodes these probabilistic dependencies. It consists of a series of nodes, one for each variable in the model. Each node has a number of parents, which correspond to variables upon which the node is dependent. The set of connections between the nodes forms a Directed Acyclic Graph (DAG).

An example Bayesian network is shown in Figure 2.9. In this network, the variable *Weather* is conditionally dependent upon *Temperature* and *Rainfall*, and *Temperature* is conditionally dependent on *Humidity* and *WindSpeed*.

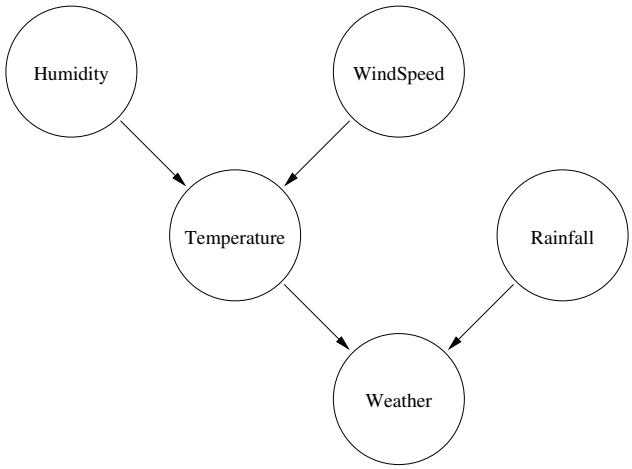


Figure 2.9: A simple Bayesian network

Associated with each node is a conditional probability table. This is similar to the joint probability table above, except that only the variables on which this node depends are included. The table shown below, associated with the node *Weather*, uses a slightly different representation for probability estimates based on each possible combination of values for the parents. Each row in the table contains the probabilities of each allowable value of the node, which add up to a total of 1. This information can be used when determining the value of a variable based on observations for each of the parents. For example, if *Temperature* has the value Hot and *Rainfall* has the value No, then there is a 0.8 probability that the weather will be sunny.

Temperature	*Rainfall*	P(*Weather*=Sunny)	P(*Weather*=Windy)	P(*Weather*=Rainy)
Hot	Yes	0.3	0.3	0.4
Hot	No	0.8	0.1	0.1
Warm	Yes	0.2	0.6	0.2
Warm	No	0.5	0.2	0.3
Cold	Yes	0.1	0.4	0.5
Cold	No	0.2	0.2	0.6

Table 2.2: Conditional probability table

Chapter 3

Design

This chapter details the overall process and specific algorithms developed as part of this project and implemented in the prototype system described in Chapter 4. While many of the ideas have been inspired by previous work in related areas, the specifics of the processing performed are new to this project.

The purpose of the system is to reduce the amount of information displayed to the user at any one time. The information displayed should consist as much as possible of content that is of most interest to the user. This means that extraneous information such advertising, navigation and decorative page elements should be hidden from view, and only the main content of the page and other relevant portions displayed.

The design is driven by four major goals:

1. *Present the most relevant information*

   By analysing the document and determining which parts are likely to be of most interest to the user, the amount of time required for the user to identify and navigate to the parts of the document they want read is reduced. This is particularly important for low-resolution displays where such navigation may involve a significant amount of scrolling.

2. *Avoid information loss*

   Although parts of the page are hidden from view, they should still be accessible to the user. Facilities should be provided which allow the user to select the parts of the page they want to view. By default, only the most relevant portions should be displayed, with additional interaction required to view the other parts.

3. *Take a best effort approach*

   Due to the wide variety of pages available on the web and the subjective nature of the graphic design process, it is not possible to handle every conceivable type of page layout flawlessly. There will inevitably be cases where a page has an odd layout or is significantly different from the types of pages that the system has been trained to handle. The most realistic approach is to aim for a good level accuracy in as many cases as possible, while recognising that some may be handled sub-optimally.

4. *Provide fall-back options*

   In cases where it is found to be particularly difficult to split up a page or classify different parts of it, the user should still be able to view the page in a usable manner. This can be achieved by providing a default display which shows the entire page, so that all of the information can still be accessed in a reasonable, albeit less convenient manner.

The process undertaken when preparing a page for presentation consists of three stages: *Segmentation*, *Classification* and *Page extraction*. Segmentation involves analysis of the page layout to identify areas containing sets of related elements, and constructing segments corresponding to these areas and the elements contained within them. Classification is the process of determining which type of content is present in each segment, so that decisions can be made about which of these should be displayed to the user. In page extraction, different views of the document are constructed, each representing one or more segments based on the classification information. Additionally, a training process, performed separately from these stages, uses sample data and manually specified classifications to build up the data structures used in the classification process.

3.1 Segmentation

The segmentation process divides up the page into multiple, distinct segments. Each of these contains a group of elements that reside near each other and serve a related purpose on the page. As a result of the process, each node in the DOM tree is either associated with a particular segment, or with no segment at all. The latter set consists of nodes containing only whitespace or decorative elements, or high-level nodes that cross the boundaries of multiple segments. Those that are assigned to segments represent parts of the tree that can be excluded from a modified version of the document, to hide the segments they are associated with.

Two main approaches can be taken to the problem of identifying segments within a HTML document. One is to examine the DOM tree and pick out groups of nodes based on the names of elements and relationships between them, or the amount of content present within a node [7]. This produces segments based on criteria such as relationships between parent, child, and sibling elements, and the presence or absence of particular types of nodes such as tables or lists. A common technique for page layout is the use of tables with each cell corresponding to a separate segment of the document, so these can be identified by picking the contents of certain cells to be marked as segments.

The other approach is to consider the visual representation of the document instead of the DOM tree itself, and perform geometric analysis to determine the segments. Characteristics such as position, size, colour, and borders of elements are considered, as well as the spatial relationships between them. In this context, elements are grouped based on their spatial proximity instead of locations within the tree. Extra data obtained from the rendering process, which would not otherwise be available, can also be taken into account. Visual analysis is used extensively in the field of OCR [5], where the only representation of a document available to the segmentation process is a scanned image.

Geometric analysis was chosen because it appears to be the most effective approach, given that it is more closely aligned with the way in which a person would divide up the page when viewing it on screen. Segments correspond to rectangular areas on the rendered page, rather than sets of nodes adjacent in the DOM tree which may reside at visually distant locations within the graphical representation. Since a given visual appearance can be achieved using different element structures with the appropriate use of style rules, it is better for segmentation to depend on the appearance of the document rather than the way in which it structured internally. A page layout built using tables will be segmented in the same manner as one constructed without using tables, as long as they look the same when rendered.

Most of the approaches taken to page segmentation for OCR purposes have focused on the identification of blocks of content surrounded by whitespace on the page. This is based on the observation that the majority of printed materials have a white background with black or coloured text and images overlaid on top. Segments are thus identified as portions of the page separated by whitespace.

Web pages tend to differ from this model. Instead of segments being separated by whitespace, each usually has a different background colour. Many sites use a white or lightly coloured background for the main area of the page, with sidebars and the top or bottom of the page in a different colour. Instead of attempting to identify blocks of content among a predominantly white page, the approach described here operates by detecting areas with different background colours. Appendix B includes examples of pages designed in this manner, and the results of applying the segmentation algorithm to them. Not all pages follow this design model; some do in fact use an all-white background or adjacent segments with the same background colour, separated by borders. The segmentation process presented here does

not take these into account, as they are less common than pages divided up using different background colours. The challenge of dealing with such pages presents an opportunity for further research.

Another difference between the segmentation of HTML documents and printed material is that instead of analysing the image pixel-by-pixel, it is possible to make use of already available information about which areas of the page contain blocks of text. The extra step required in OCR of identifying these areas from on the image data is not necessary here, because this information can be obtained directly from the HTML renderer.

3.1.1 Rendering

Before any segmentation can be performed, it is first necessary to load and render the document, so that information about the visual representation of page objects can be obtained. This information is used to identify areas of the page which are to be combined into segments. The process described here is based on the way in which the rendering engine used in the prototype operates, and is separate from the segmentation algorithm introduced in the next section. Other rendering engines operate differently, but produce essentially the same results.

The rendering process, depicted in Figure 3.1, includes a number of steps. While conceptually these can be considered separate processes, in practice they effectively run in parallel to support incremental loading of pages. When loading a page from a remote server, the portion of the page received by the client is parsed and rendered while more data is arriving. The user typically sees the rendered page display updated several times as the page loads. This is a benefit of the rendering engine's architecture, which enhances usability when run as part of a standard web browser. However, the segmentation process described here can only be run once the page has been completely loaded, since it relies on information about the whole page. While this means that incremental loading provides no benefit in the usage context of the system, it is useful to understand the concept when examining the how the rendering engine operates.

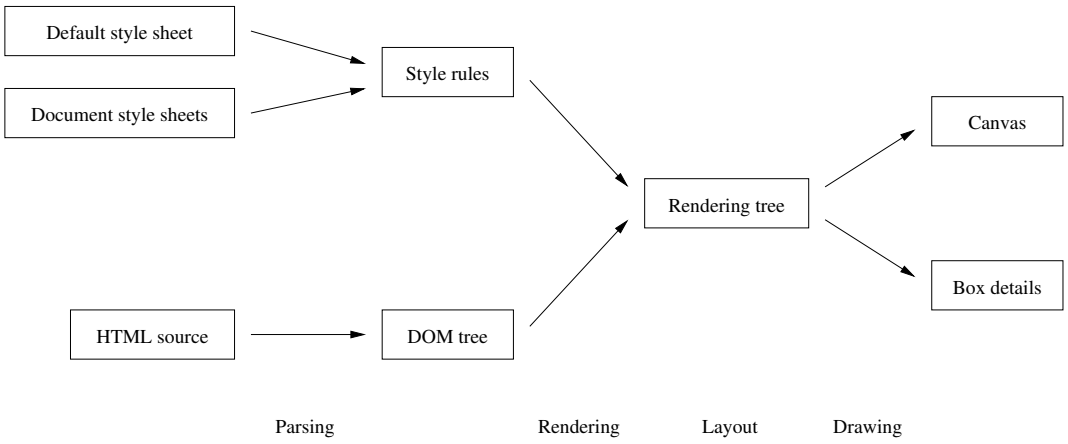


Figure 3.1: The HTML rendering process

The first step in the process is to initiate the retrieval of the HTML file from its source location, typically a remote web server or the local file system. The HTML parser then begins to build up a DOM tree based on the contents of the file. For every tag that is found, an *Element* node is created with the appropriate tag name and attributes. Characters residing between tags are placed into *Text* nodes which are added as children of the element corresponding to the enclosing tag. Element nesting occurs according to rules defined in the HTML specification, which specify restrictions on what types of elements can be contained within others. In practice, many web pages do not conform to these rules, so the parser must be very tolerant of errors. Invalid syntax is also common on a lot of sites; this is also taken into account and corrected for during parsing.

As the DOM tree is constructed, any style sheets included in or referenced from the document are loaded and parsed to create a list of style rules. In addition to these, a built-in style sheet specifying default display attributes for each standard HTML element is also loaded. The rule sets obtained from these sheets are combined together into a data structure which represents the rules in a compact manner, and allows the set of properties for a given element to be efficiently computed. The order in which the style sheets are parsed determines which rules have priority; rules specified in later style sheets override those set earlier.

Once a set of style rules has been obtained, the CSS properties associated with each node in the tree are determined. These are computed by examining the rule set and determining which rules match which elements, based on the element name, attributes and relationships with other nodes. A style object representing the properties assigned to an element is created based on the properties associated with all matching rules. These are subsequently used to determine how to display the elements.

A tree of rendering objects is then constructed based on the nodes in the DOM tree. The `display` property of each node determines which type of rendering object is created to represent the node, such as a table, list, image, form control, text or container. Each rendering object has a different method for performing layout and display calculations, and appears differently when drawn on screen. Normally, the value of the `display` property is determined by the default rules in the HTML specification; this results in the appearance of the elements being dependent on their name.

The layout process goes through the rendering tree and calculates the size and position of every rendering object on the page. These calculations take into account many factors, including the available page width, the amount of content present, style rules, relationships with other objects, and the value of CSS properties set for the object. The specific rules for performing this process are detailed in the CSS specification [51].

Finally, the set of rendering objects and associated layout information is used to draw the page onto a *canvas*. The most common type of canvas is the screen, where the page is displayed directly to the user. The canvas may correspond to a printer, or an off-screen buffer used for generation of image files representing the contents of the page. It is at this point that the information required by the segmentation process is obtained. For the purposes of this project, extra code was added to the drawing process to obtain information about the size, position and background colour for each of the boxes. Before segmentation is performed, the page is completely loaded and the drawing process is executed to record this information.

3.1.2 Region Detection

Based on the set of box details obtained from the rendering process, the page area is divided up into a grid. Divisions are placed at intervals along the x and y axes corresponding to the top, bottom, left and right sides of each of the boxes. Each grid cell is marked with a colour corresponding to the background colour of the block box in which the cell resides. This allows the segmentation algorithm to operate based on grid cells instead of pixels, reducing the amount of time required to analyse the rendered output.

The grid is then analysed to identify rectangular regions that have the same background colour. Depending on the page layout, there may be sets of adjacent grid cells with the same colour that do not form a rectangular shape. In this case it is necessary to choose an appropriate division of the area into rectangles.

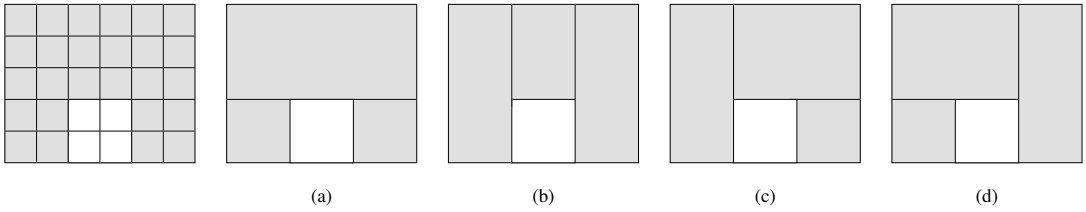


Figure 3.2: Grid cells with possible region configurations

Consider the grid cells shown in Figure 3.2. The set of adjacent grey cells is in a non-rectangular shape, so it is necessary to divide the area up into multiple regions. There are four ways of doing this — each resulting in three regions. The question then arises as to which of the configurations to choose. There are two criteria used in deciding this:

- *No region may intersect an inline bounding box.*

  An inline bounding box is the box that contains a series of consecutive inline objects, such as a sequence of text nodes and images in a paragraph. The purpose of this constraint is to prevent the undesirable situation of having a segment containing only part of a sentence or paragraph.

- *Out of all valid rectangle choices for a given top-left position, the tallest is chosen*

  Most web pages use a column-based layout rather than a row-based layout. This gives preference to identifying columns as regions.

Figure 3.3 shows the same set of cells with inline bounding boxes included, and the chosen region configuration. Since the bounding boxes are in columns, configuration (b) is the only valid choice.

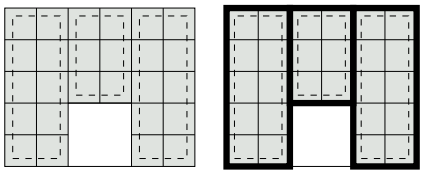


Figure 3.3: Grid cells and inline boxes, with the chosen region configuration

3.1.3 Region Merging and Segment Creation

Each segment on the page is made up of one or more regions. While most regions each go into a separate segment of their own, in some cases it is desirable to combine multiple regions into a single segment. Some types of page layouts contain logically associated adjacent regions with different background colours, such as those shown in Figure 3.4. The segmentation process looks for some common spatial relationships between regions to find cases where merging can be performed. These relationships are based on observation of common page design techniques and are intended to cover the majority of cases.

Figure 3.4: Distinct regions requiring merging

Each segment consists of one or more adjacent rectangular regions. In the general case, each region becomes a separate segment. However, there are some cases where two or more regions can be merged together to form a single segment:

- If two segments are vertically adjacent, and the top segment contains a single line of text and the bottom region contains multiple lines of text, the two regions are merged. This is a way of dealing with the common technique of having a block on the page with a body of text, and a title directly above associated with the block but having a different background colour. This is known as a *title/body pair.*

- Some pages have multiple regions with the same background colour that are vertically adjacent. An example of where this can occur is a repeating set of table cells in a navigation menu with borders dividing the cells. These are not initially identified as a single region due to the border divisions, however because no actual content is present in the space between the regions, it is useful to merge them together. This helps avoid having a higher than necessary number of segments on the page by collecting related regions in a sequence together.

  This merging occurs after the merging of title/body pairs, to cater for a sequence of these pairs. An example of a page containing this type of layout is shown in Figure 3.5. Region pairs $\{2, 3\}$, $\{4, 5\}$ and $\{6, 7\}$ are first merged together according to the title/body rule. The 3 resulting regions are then merged to create a segment containing $2 - 7$, labeled section b. Regions 1 and 8 are each put into separate segments, a and c.

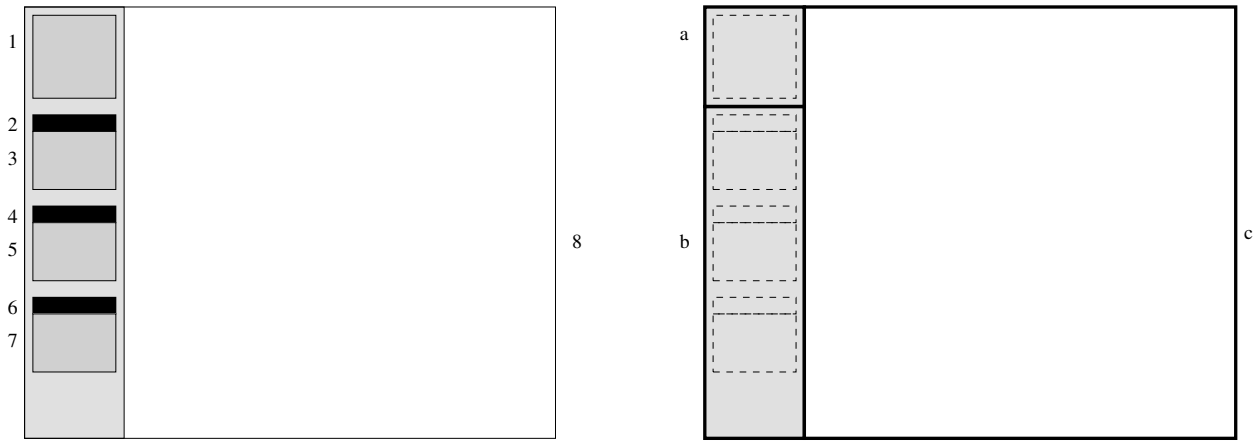


Figure 3.5: Multiple repeating instances of title/body region pairs

3.1.4 Floats and Absolute Positioning

CSS provides two properties which allow content to be placed outside the main flow of text: `float` and `position`. If an element has the `float` property set to either `left` or `right`, then all content within the element will reside in a box that is placed against the side of the page, and text outside of the element will flow around it. This is commonly done with images, but can also be applied to elements containing blocks of text and other objects. Similarly, if the `position` property is set to either `fixed` or `absolute`, and coordinates are also specified for the element, then the contents will be displayed at that location on the page, and superimposed on top of other content.

These cases present a special problem for the segmentation algorithm. Since region detection is based on combining adjacent grid cells and merging them into rectangles, elements which interrupt the flow or sit on top of a piece of content intersect the segment. Without taking extra steps to deal with this, a segment containing a float or absolutely positioned element would actually be split up into multiple segments. Additionally, the segment boundaries could potentially cross those of inline boxes, which is an undesirable situation.

To deal with this problem, these elements are treated specially when identifying segments. At the start of the process, before the page is divided up into a grid, all elements in the DOM tree which have `float` or `position` set are marked as *float roots*, indicating that they effectively start a new sub-page within the current one. During subsequent analysis of the page, these nodes are treated as if they didn't exist in the document. The assignment of background colours to grid cells only considers elements which do not have any ancestors that are float roots. Areas of the page that contain left or right floats in the rendered output are treated as blank areas, and content that normally is obscured by a positioned element placed on top is assumed to be completely visible.

Once segmentation has been performed on the main parts of the page, a new segment is created for each float root in the document. Note that it is possible for these to be inside other segments, or even other float roots, so this may result in segments that contain other segments. For the purposes of identifying which parts of the document belong to which segments, content inside float roots is only considered to be in the float root's segment, not in any segment corresponding to a higher-level node in the tree.

3.1.5 Segment Representation

Once the physical dimensions of the segments on the page have been identified, it is necessary to determine the set of DOM nodes associated with each. This is used for calculating information about the segment for classification purposes, and to perform document modifications, as discussed in section 3.3.

Consider the set of all nodes in the DOM tree, denoted N. Each node N_i has associated with it a set of rendering objects O_i which appear on the visual representation of the page. From the processes described above, the information about the size and position of each segment is already known. By comparing the coordinates of each of the rendering objects with those of the segment, it is possible to determine which of these reside in the segment. A rendering object O_{ij} is defined as being contained within a segment S_k if all of the following conditions are met:

$$\begin{aligned} \text{contains}\,(S_k, O_{ij}) \quad = \quad & \text{top}\,(O_{ij}) \geq \text{top}\,(S_k) \\ \wedge \quad & \text{bottom}\,(O_{ij}) \leq \text{bottom}\,(S_k) \\ \wedge \quad & \text{left}\,(O_{ij}) \geq \text{left}\,(S_k) \\ \wedge \quad & \text{right}\,(O_{ij}) \leq \text{right}\,(S_k) \end{aligned}$$

Not all nodes end up being associated with a segment — such as those that contain only whitespace, or have only descendants containing whitespace. When a segment is hidden from view, some of these nodes are removed from the tree depending on where they lie. This is explained further in Section 3.3.

By maintaining a reverse mapping between rendering objects and nodes, the set of of nodes associated with each segment can thus be obtained.

$$\text{nodes}\,(S_k) \quad = \quad \{N_i \mid \exists O_{ij} \mid \text{contains}(S_k, O_{ij})\}$$

For efficiency purposes, each segment does not store the complete list of nodes associated with it. Instead, each segment has a list of root nodes and a list of exclusions. The root nodes are those that have all of their descendants assigned to the segment, with the exception of the exclusion nodes and their descendants. To illustrate this, consider the DOM tree shown in Figure 3.6, with segment nodes shaded. Nodes 1 and 2 have all of their descendants in the segment, while node 3 contains a descendant float root, node 4. Thus, the list of root nodes for the segment is $\{1, 2, 3\}$ and the exclusion list is $\{4\}$. The segment associated with the float will have a root list of $\{4\}$ and an empty exclusion list.

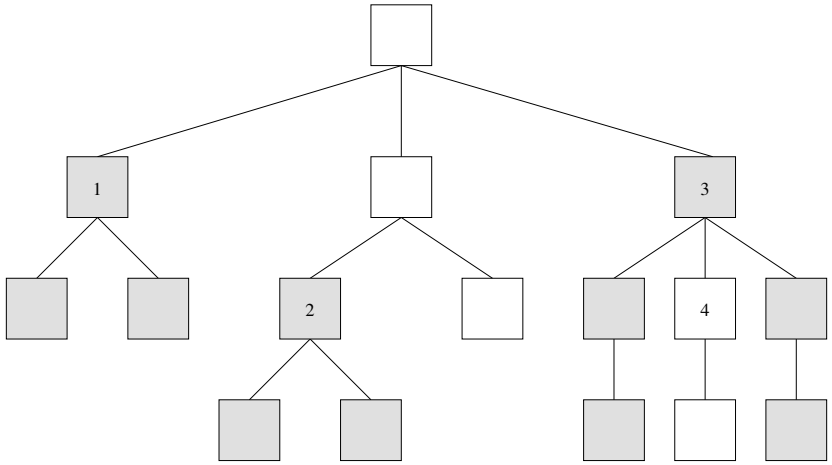


Figure 3.6: Segment nodes with an excluded float

3.2 Classification

The classification process assigns a *class* to each segment in the document indicating the type of information it contains. It uses information about the segment as well as a Bayesian network constructed from previously supplied training data to decide which class to use. The classifier is built during the training process, described in Section 3.4. The set of available classes, denoted C, is listed in Table 3.1. Examples of each type of class are shown in Appendix C.

Class	Description
Content	Main body content that is of primary interest to the user
Title	The title of a document or a section within the document
Navigation	Page elements related to site navigation
Advertising	Banner ads and other advertising material
Logo	Images and other decorative information prominent on the site
Sidebar	Information provided on the page as "extra" information that may be of interest, but is not part of the main body of text
ExtraInfo	Extra pieces of information such as copyrights, disclaimers, help text etc.
Form	Form controls for purposes such as login, registration and surveys
Decoration	Page elements that are purely for decorative purposes and do not provide any actual information

Table 3.1: Segment classes

The class chosen for a given segment depends on the values of the *feature variables* associated with the segment. Each feature variable represents a piece of statistical information relating to the segment's appearance, and is derived from both the DOM structure and visual rendering of all of the nodes in the segment. These variables are chosen to give a reasonable indication of various aspects of the appearance, so that visually similar segments will be close to having the same set of values for all variables. The set of variables is denoted V, with allowed (V_i) giving the set of allowed values for a variable $V_i \in V$.

For each segment, 273 different feature variables are calculated. Each of these represents one of three statistics relating to one of the 91 element types defined in the HTML 4 specification [47], listed in Appendix A. For each element type, the following is calculated:

- *element*`-text`

  The proportion of characters in the text of the segment that reside in text nodes that are descendants of this type of element. This variable ranges from 0–1.

- *element*`-rendered`

  The proportion of the total rendered area of the segment corresponding to elements of this type or their descendants. This variable ranges from 0–1. It is calculated for a fixed screen width of 800 pixels, since for many pages the area taken up by certain elements depends on the width of the page.

- *element*`-count`

  The number of elements of this type present in the segment

In order to use these variables with the classifier, it is necessary to represent them as a discrete number of values. *element*`-text` and *element*`-rendered` are real-valued variables ranging from 0–1. These are quantised over a set of 5 values indicating both extremes of the scale, and ranges in between. While more advanced discretisation methods are possible [52], the fixed ranges are chosen here for simplicity.

$$\texttt{element-text} = \begin{cases} 0 & v = 0 \\ 1 & 0 < v \leq \frac{1}{3} \\ 2 & \frac{1}{3} < v \leq \frac{2}{3} \\ 3 & \frac{2}{3} < v < 1 \\ 4 & v = 1 \end{cases}$$

where v is is the value obtained from the proportion. The same quantitisation is used for *element*-rendered.

element-count is measured differently, as it comes from an integer value with no finite limit. Its value is assigned as follows:

$$\texttt{element-count} = \begin{cases} 0 & v = 0 \\ 1 & v = 1 \\ 2 & 2 < v \leq 5 \\ 3 & 5 < v \leq 20 \\ 4 & v > 20 \end{cases}$$

Again, v represents the original value, i.e. the number of such elements in the segment.

The type of classifier used here is the *naive Bayes* classifier [27]. It is a Bayesian network consisting of one node corresponding to the class, and other nodes representing each of the feature variables. As shown in Figure 3.7, each feature variable node is causally dependent upon the class node, and no others. While this may seem an oversimplifying assumption, the naive Bayes classifier has been found to produce results comparable to those of more advanced Bayesian classifiers [11]. Additionally, its simplicity provides a significant advantage over other methods in terms of ease of implementation.

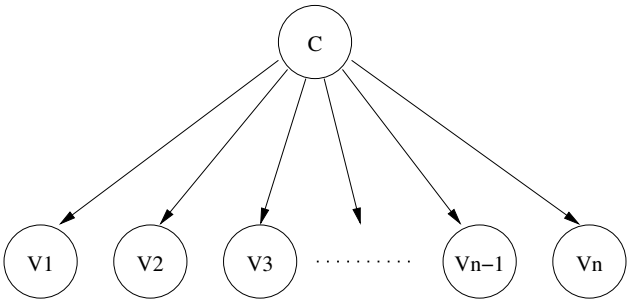


Figure 3.7: Naive Bayes classifier

In order to classify a segment, it is necessary to know all values of the feature variables, as well as information about the classifier constructed during the training process. Here, S is the set of all segments, v_{ij} is the value assigned to variable V_i in the segment S_i, and class (S_i) indicates the class assigned to the segment during training. The information built in to the classifier is as follows:

- $\|S\|$

  The total number of segments in the training set

- classcount $(C_j) = \|S_i \in S \mid \text{class}\,(S_i) = C_j\| \, \forall C_j \in C$

  The total number of segments associated with each class

- valcount $(V_j, m) = \|v_{ij} = m \mid 1 \leq i \leq \|S\|\| \, \forall V_j \in V, m \in \text{allowed}\,(V_j)$

  The number of assignments of each allowable value to each variable for the whole training set.

- classvalcount $(C_k, V_j, m) = \|\text{class}\,(S_i) = c_k \wedge v_{ij} = m \mid 1 \leq i \leq \|S\|\| \,\forall C_k \in C, V_j \in V, m \in \text{allowed}\,(V_j)$

  For each class, the number of assignments of each allowable value to each variable for segments in that class.

The class chosen is based on the most likely classification of the segment given the values of each of the feature variables. That is, for each class $C_k \in C$ it is necessary to calculate $P\,(\text{class}\,(S_i) = C_k \mid v_{i1} \ldots v_{in})$. The class with the highest probability is chosen.

The probability of the segment being of a particular class C_k can be determined from the individual variable probabilities using Bayes' rule:

$$P\,(C_k \mid v_{i1}, \ldots, v_{in}) \quad = \quad \frac{P\,(v_{i1}, \ldots, v_{in} \mid C_k)\,P\,(C_k)}{P\,(v_{i1}, \ldots, v_{in})}$$

Since all variables v_{ij} are conditionally independent, it is possible to calculate the probability based on the product of the prior probabilities of each variable, as well as the probabilities of each variable given the classification C:

$$\begin{aligned} P\,(v_{i1}, \ldots, v_{in}) &= P\,(v_{i1}) \ldots P\,(v_{in}) \\ P\,(v_{i1}, \ldots, v_{in} \mid C_k) &= P\,(v_{i1} \mid C_k) \ldots P\,(v_{in} \mid C_k) \end{aligned}$$

The prior probability of each variable is the proportion of segments in the training set which have that particular variable assignment:

$$P\,(v_{ij}) \quad = \quad \frac{\text{valcount}\,(V_j, v_{ij})}{\|S\|}$$

And similarly for the probability of each variable assignment given the class:

$$P\,(v_{ij} \mid C_k) \quad = \quad \frac{\text{classvalcount}\,(C_k, V_j, v_{ij})}{\text{classcount}\,(C_k)}$$

The prior probability of a class is the proportion of segments in the training set that have that class assigned to them:

$$P\,(C_k) \quad = \quad \frac{\text{classcount}\,(C_k)}{\|S\|}$$

3.3 Page Extraction

A page extract is a partial view of a document which in which only certain segments are visible. When viewing a page using this system on a small screen device, a series of page extracts is constructed containing different selections of segments from the original document. Each extract contains all segments of a particular class. After loading a page, one extract for each class is generated, and the extract containing *Content* segments is initially displayed. This means that when a user first views a page, they see only the main body of text on the page, and not peripheral information such as navigation and advertising. The user interface provides mechanisms to switch to a different extract in order to view the segments contained within it, so that all of the information on the original page is still accessible.

In implementation terms, a page extract is a separate document which contains copies of a subset of nodes from the original document. The nodes included are those assigned to the relevant segments, plus certain additional nodes

necessary for maintaining the layout and presentational aspects of the relevant parts of the page that are visible in the original. While the selection of the former set is straightforward, determining which additional nodes must be included requires extra analysis of the document. There are three reasons why this must be done: to maintain spacing between segments, to ensure the same set of style rules are applied, and to obtain correct cell alignment in tables.

3.3.1 Spacing

Whitespace nodes are those that do not have any content, such as empty table cells or paragraphs. This also includes spacer images, defined as having either a 1×1 size, or all pixels as a single colour. Because whitespace nodes do not provide any actual information, they can can be removed without affecting the semantic meaning of the document.

However, in some cases it is desirable to keep these nodes, since they play a role in the layout of the document and provide a visual separation between different parts of the page. In order to determine whether a whitespace node should be included in a page extract, its position and size is compared with that of the segments that are to be displayed. A node is kept if it resides either wholly or partially between an included segment and the edge of the page, or between two included segments. Nodes which do not meet these constraints are not copied into the extract.

3.3.2 Style Rules

One of the factors determining the CSS properties set for an element are the rules matching its ancestors in the tree. Certain properties in CSS are said to *cascade*, which means that if they are set on a particular node then they are implicitly set on all descendant nodes. It is common practice to assign a set of style properties on an element high up in the tree, so that all nodes under it have the same appearance. This allows a consistent appearance to be given to portions of the document, without having to specify properties for each individual element.

When building a page extract, it is important to maintain these rules so that the appearance of the included segments remains the same. If a node was included in the page extract by itself rather than inside its previous ancestor nodes, its style information could be lost. Therefore, when selecting a node for inclusion it is also important to also select all of its parent nodes. However, not all siblings are necessarily included — only those that meet the other criteria with regards to segment visibility or whitespace inclusion are selected.

3.3.3 Cell Alignment

When constructing a page extract consisting of a subset of the cells in a table, it is necessary to include some of the other cells in order to maintain the correct layout. The HTML table model is *row primary*, which means that tables are organised first by rows, and secondly by columns. That is, a table consists of one or more rows, each of which contains one or more cells. When the layout of a table is calculated, columns are formed by cells that are in the same index in each row. Removing a cell at the start of a row will offset all of the other cells in the row, changing the assignment of cells to columns.

When reconstructing the table in the new document based on a subset of the cells, it is important to ensure that cells that were in the same column in the original document stay in the same column in the page extract. Thus, if some cells are not to be included, then depending on their position in the table it may be necessary to actually add them as empty cells in the new table. This maintains the correct relationship between cells and columns.

Another complication is that HTML tables support cells with a row span or column span greater than 1. That is, a given cell may take up multiple rows or columns. This also needs to be taken into account when deciding which cells to copy over.

3.4 Training

In order to perform automatic classification, it is necessary to first construct a naive Bayes classifier which is able to classify segments based on information supplied during the training process. To build the classifier, a set of pages, called the *training set*, is manually classified by a human in order to get some initial indication of what each of the segment types look like. Each segment of each page in the training set is then analysed to calculate all of the feature variables, which are then paired up with the manual classifications and used to construct a classifier.

The information collection from this is straightforward, since all that is required is to count totals for segments, classes and variable assignments. The four sets of information listed in Section 3.2 are derived as follows:

- $\|S\|$

  A count of the number of segments. Not all segments in the pages in the training set need to be classified — only those that have classification information provided are taken into account when building the classifier.

- classcount (C_j)

  For each class, a separate count is kept which indicates the number of segments with that class assigned to them. The result is an array of integers storing the counts for each class.

- valcount (V_j, m)

  For every possible feature value, an array of counts is stored corresponding to the allowed values of each variable. Every time training information for a segment is processed, each array has one of its elements incremented. The result is a two dimensional array of integers representing the number of assignments to each allowed value of each feature variable.

- classvalcount (C_k, V_j, m)

  This is essentially the same as valcount, except the result is a three dimensional array. Each element of the first dimension corresponds to the variable assignment counts for the appropriate class.

The training information is calculated based on two sets of information from the training set: the values of all feature variables for all trained segments, and the classes manually assigned to each segment. The actual process of constructing the classifier does not need access to any structural or visual information about the pages; in fact, it operates on segments only and has no concept of pages. These variables are still calculated as part of the training process, but this is done in an earlier stage, which is separate from classifier construction. For specific details of how this is achieved in the implementation, see Section 4.1.

Chapter 4

Implementation

In addition to the design, the other major part of the project was the construction of a prototype system to test out the algorithms presented above. This was useful for both experimenting with and fine-tuning the algorithms, as well as measuring their effectiveness by training against a set of sample data.

The system was implemented in C++ using the Qt widget set [35] and KDE libraries [21]. An existing rendering engine, KHTML [22], was used to provide all of the facilities related to HTML parsing, DOM tree construction and page rendering. Some minor modifications were made to the rendering engine to provide the segmentation and analysis algorithms with additional information about the position and display properties of elements on the page. The analysis and training files generated as part of the training process are stored in XML format. The code developed as part of this project consists of around 4,000 lines. Figure 4.1 shows the user interface provided by the prototype.

The prototype operates by providing a list of pages from which the user can select, along with an area displaying the page in either original or modified form. Upon selection of a page from the list, the corresponding page is loaded and segmentation performed. The list segments is also displayed in the window.

There are three different methods of displaying the page, as shown in Figure 4.2. The user can switch between these three methods at any time by selecting from a menu. These are:

- *HTML display*

  Shows the HTML file in unmodified form, in the same manner as a normal web browser.

- *Segmentation display*

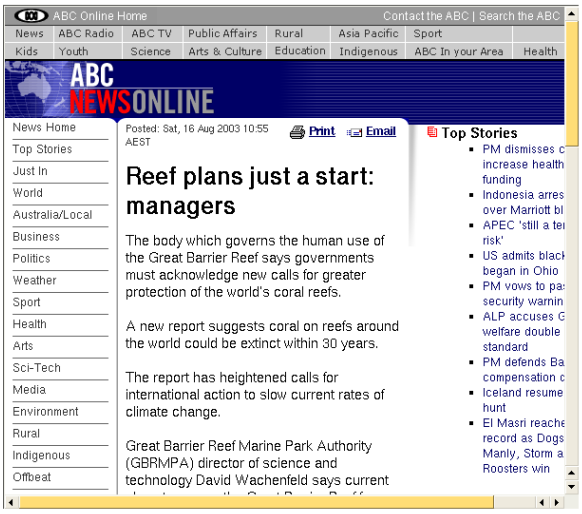
  Shows a set of boxes that indicate divisions between segments that have been identified by the program, as well as inline and container boxes corresponding to elements in the document. Options are provided to hide or show certain box types, change the zoom level, and overlay the boxes on top of the rendered page.

- *Modified display*

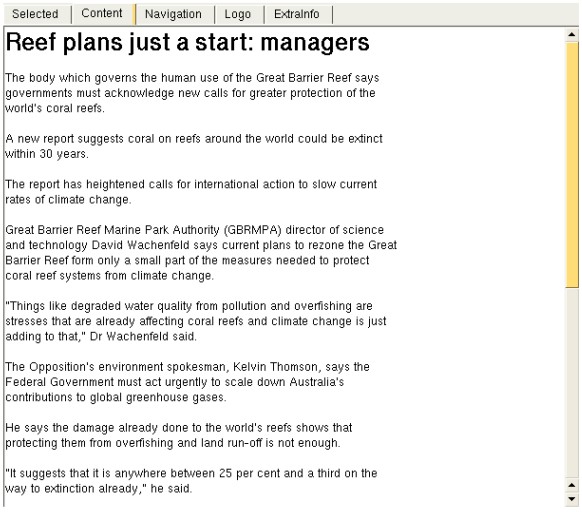
  Shows extracts from the documents based on the content assigned to different segment classes. For each class, a tab is shown at the top, which, when clicked on, results in the segments of that class being shown. An additional tab shows segments based on whether or not they have been selected for display by the user. Segments can be hidden or shown by double-clicking on them in the segment list in the left-hand side of the window.

The features provided by the prototype are intended for providing assistance with debugging and testing the segmentation, classification, training and page extraction algorithms. They also provide a means of measuring the effectiveness of the classifications made by the system on sets of test data.

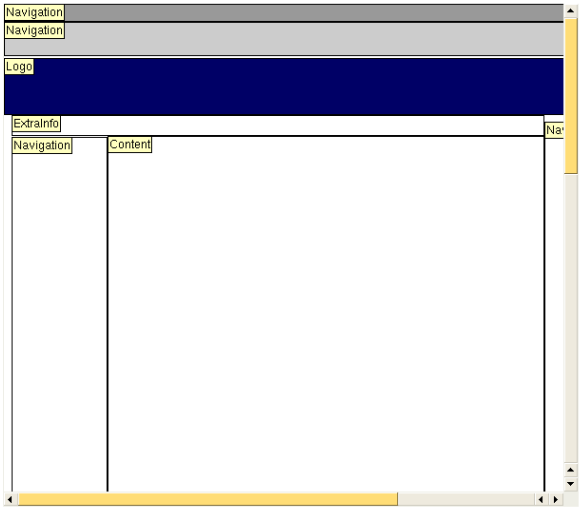
Figure 4.1: The prototype system



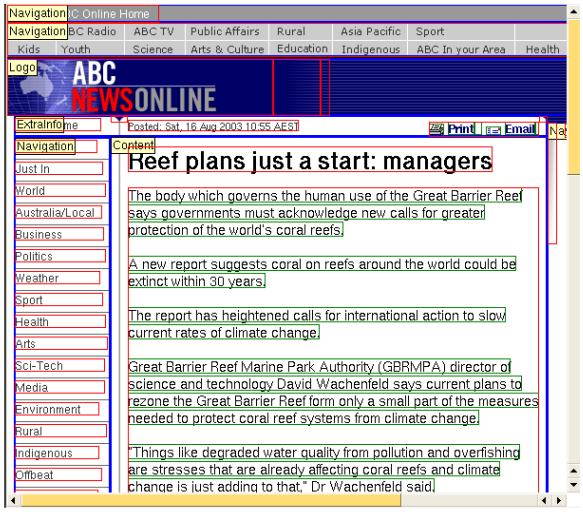
HTML display



Modified display



Segmentation display



Segmentation display with inline & container boxes

Figure 4.2: Available display modes

4.1 Training

In order to perform automatic classification of segments, it is first necessary to construct a classifier which is able to make decisions about which class should be assigned to a given segment. This classifier is built using information derived from a set of pages known as the *training set*, as well as training information provided by a user. Before any of this can occur, the training set must first be obtained from various sites around the web, and should contain a representative sample of the types of pages that are likely to be encountered in normal usage of the system. Details of the sets of pages used for evaluating the prototype are given in Chapter 5.

The process of building a classifier is split up into three stages: *Manual classification*, *Document analysis* and *Classifier construction*. The first two stages each generate a data file based on the information obtained during that stage, and the third stage combines information from the two files and performs the final calculations necessary to build the classifier. The training process is divided up in this manner due to the time consuming nature of the first two stages, making it desirable to be able to perform them separately instead of each time automatic classification is performed.

- *Stage 1: Manual classification*

  In this stage, a user is required to go through each page in the training set and classify all of the segments. A page is loaded by selecting it from the list; then, in the segmentation display, segments are assigned to a particular class by right-clicking on the corresponding area of the page and selecting the class from the pop-up menu that appears. Not all segments need to be classified; those that are omitted will not be taken into account in the classifier construction process. The set of classifications is then saved to a *classification file* which provides input to the processing performed in the third stage.

  Because manual classification can be a time consuming task for a large number of pages, it is not necessary to complete it all in one session. The classification file can be saved at any time, and loaded again later on. Thus, classification can be done in several sessions, and also updated later if any new pages are added to the training set.

- *Stage 2: Document analysis*

  The system iterates through all pages in the training set, loading each one in turn and calculating the values of the feature variables for all segments. Once these have been obtained for all pages, the information is saved to an *analysis file*, which is used as input into the third stage of the training process.

  The analysis stage can be performed on an automated basis, and requires no user interaction other than selecting the appropriate menu item to start the process. Since it is independent of the manual classification stage, it can be run again if the analysis algorithms are changed. This can be done before or after manual classification.

- *Stage 3: Classifier construction*

  Once an analysis and training file have been created, they can be used to build a classifier. The details recorded in the files about the feature variables and classifications of each segment are used to construct a naive Bayes classifier, using the algorithm described in Section 3.4. The resulting classifier can then be used to automatically assign classes to segments on subsequently loaded pages.

  Since this process takes only a short amount of time, the classifier is kept in memory and not stored on disk. It can be easily be rebuilt from the training and analysis files during a later invocation of the program.

4.2 Classification

In addition to the training facilities provided, the prototype can also run in classification mode, which simulates the way in which a browser using this system would operate. The interface operates in much the same way as in training

mode, except that when a page is loaded, automatic classification is performed and segment labels are assigned. By switching the display area to modified mode, the user can see the page as it would be split up for display on a small screen device, and switch between the different page extracts.

When switching to classification mode, the user must specify classification and analysis files from which the classifier is to be constructed. These files do not have to correspond to the set of pages being used. This makes it possible to examine one set of pages with a classifier built from another set. This is a useful way of investigating how well the classification algorithm performs for unseen data.

Automated classifications are represented in the user interface in the same way as manual classifications. When viewing the page with the segmentation display, labels are shown against each segment indicating the class chosen for that segment. This information can be saved to a classification file in the same format as during the training process. The resulting files can then be supplied to a file comparison tool such as the UNIX `diff` command to manually inspect differences between the automatic and manual classifications.

4.3 Measurement

While classification mode provides an interactive way of inspecting the results of automatic classification, measurement mode allows a set of pages to be classified in batch. This allows quantitative results to be obtained to measure the accuracy of classification.

Measurement mode operates with two sets of pages: a training set and a test set, both of which must have manual classifications supplied for them. The training set is used to build a classifier which is then used to classify all of the pages in the test set. The resulting classifications are then compared with the corresponding manual classifications for the test set, to determine the accuracy of the classifier. If only a training set is supplied, this is used as the test set, so that the classifier is used with data that has been trained with.

The output produced by this mode includes counts for all possible combinations of classes indicating the number of times that segments trained against each class had each other class assigned to them during automated classification. These can be then post-processed to determine information such as the precision and recall of the classifier, the two measures of classification effectiveness described in Section 5.1. An example of the output produced by measurement mode is given in Appendix D.

Running the program in measurement mode is very efficient, because it does not need to load any of the actual pages in the test or training sets. All information necessary for training and measurement is contained within the specified classification and analysis files.

4.4 Implementation Details

4.4.1 Segmentation

The description of the segmentation process given in Section 3.1 is a general overview of the tasks performed. It is useful, however, to look at the process in finer detail to get a picture of how it is implemented in the prototype, and how to perform segmentation in an efficient manner. A significant amount of effort was put into optimising the code to get good performance, which was necessary due to the large number of pages being processed as part of the test sets used for evaluation. For the majority of pages, the optimised segmentation process takes less than half a second to run.

The particular implementation used here is specific to the classes and APIs provided by KHTML; if the algorithm was to be implemented using another rendering engine, the data structures available for use would be different. Additionally, the prototype relies on certain modifications made to KHTML which provide access to internal information not normally available via the public APIs. The implementation is thus dependent on a customised version of the

library. Most of the ideas, however, could be implemented for other engines, as the main difference is the way in which information about the rendered objects is obtained.

The main change was the addition of a new class, `RenderInfo`. This contains a single abstract method, `box()`, which is called by the renderer whenever it draws certain types of rendering objects, including images, text and container boxes. The method takes a number of parameters specifying various properties of the rendering box, including size, position and background colour. An additional method was added to the main view class of the rendering engine to set the current `RenderInfo` object associated with a document, and when the page is rendered, `box()` is called for each relevant rendering object.

The central class used for the segmentation process is `DocInfo`, which represents all relevant information about the document and set of segments. It includes methods corresponding to each of the different stages of the segmentation process, which are executed in sequence when a page is loaded. These are as follows:

1. `getRenderBoxes()`

   Before the page can be divided up into segments, it is necessary to obtain information from the rendering engine about objects on the page. This method gets the rendering engine to draw the page, and passes the `DocInfo` object in as a parameter to collect the information. `DocInfo` is a subclass of `RenderInfo`, so whenever an object is found during the drawing process, the `DocInfo::box()` method is called which records the relevant details. After drawing is complete, the result is an array of `RenderBox` objects, each of which contains the following information:

`rect`	The coordinates and size in pixels of the object on the page
`colour`	The background colour of the object
`renderer`	The corresponding rendering object
`type`	The type of box: one of `container`, `text`, `image` or `form`
`floatRoot`	Whether or not this object has the `float` property set
`inFloat`	Whether or not this object or one of its ancestors has the `float` property set

   Note that it is possible to have more than one `RenderBox` object per rendering object, for example when multiple line boxes are generated for a text node.

2. `determineGridIntervals()`

   This function iterates through all of the `RenderBox` objects and constructs two arrays of integers, `xPoints` and `yPoints`, which contain a unique set of coordinates corresponding to the left, right, top and bottom of each of the boxes. These determine which intervals the grid is divided into for use in later stages of the process.

   As part of this process, four two-dimensional arrays are created which correspond to the grid cells: `regionColours` contains the background colour of each cell, `inRegion` stores boolean values indicating whether a cell has been assigned to a segment, and `joinRight` and `joinDown` store boolean values indicating whether or not there is an inline box partially covering the cell which continues on to another cell to the right or below. The last three arrays are all initialised to `false`, and `regionColours` is initialised to white.

   The use of a grid rather than pixels is mainly for optimisation purposes. Inspecting every pixel to find segments would be computationally expensive, and since the page is known to contain objects in rectangular areas with the same background colour, it makes sense to operate on a cell-by-cell basis instead.

3. `colourCells()`

   The `regionColours` array constructed in the previous step is populated by this function, by iterating over all of the `RenderBox` objects and marking the appropriate grid cells with the colour of each box. The contents of the `RenderBox` array is ordered by the sequence in which the objects were drawn on the page, so boxes present later in the list that cover the same area as those earlier in the list will replace the colours set in the relevant cells.

Boxes which have either the `inFloat` or `floatRoot` properties set are ignored for these purposes, since only objects which are not floating or absolutely positioned are taken into account at this stage of the process.

4. `findBoundingBoxes()`

   To avoid creating segments that intersect text and images, it is necessary to determine the bounds of each collection of adjacent inline objects in the tree. As described in Section 2.4.5, the rendering tree is structured with block and inline objects. Block objects can contain both inline and block children, while inline objects can only contain other inline children. The subtree corresponding to each top-level inline box constitutes a set of objects which must be kept together and all placed in the same segment.

   The bounds of each inline group are calculated by taking the smallest top and left coordinates out of all of the boxes, and the largest right and bottom coordinates. This results in an array of `BoundingBox` objects, each of which has the following properties:

`rect`	The position and size of the bounding box, measured in pixels
`gridRect`	The position and size of the bounding box, measured in grid cell coordinates
`renderers`	An array of rendering objects that reside within the box
`inFloat`	Whether or not this bounding box is inside an object with the `float` property set

   As each bounding box is added to the list, the `joinRight` and `joinDown` arrays are updated. Each element in these arrays corresponding to an area within the box is set to true, with the exception of the right-most column of the box's cells in `joinRight`, and the bottom row of the box's cells in `joinDown`. These two arrays are subsequently used to make decisions about which segments to choose, such that all of the grid cells assigned to a segment must have all of their corresponding `joinRight` and `joinDown` values equal to `false`, ensuring that no segment intersects the edge of a bounding box of inline objects.

5. `identifyNextSegment()`

   Once all the information from the previous steps has been obtained, the process of assigning grid cells to segments can begin. The algorithm works by iterating through the grid in a left-to-right, top-to-bottom fashion. A cursor is initially placed in the top-left cell, and works its way across and down the page in a left-to-right fashion identifying sections as it goes. The position of the cursor is stored in the variables `regionStartX` and `regionStartY`.

   Each segment initially starts off as being a single cell in the grid, at the current location of the cursor. The bottom edge of the segment is moved down until it reaches a point where the background colour changes. If this edge intersects a bounding box, which can be detected by inspecting the appropriate entries in `joinRight` and `joinDown`, then it is moved up to the top of the bounding box.

   Next, the section is widened as far as possible. The right-hand edge is moved to the right until it reaches a column containing a cell with a different background colour, or the left edge of a bounding box which extends beyond the bottom of the section. As with the bottom edge, if the right edge intersects a bounding box, then it is moved back as necessary until it no longer intersects.

   An example of this process is shown in Figure 4.3. The bottom edge moves down to a colour boundary (1), the right edge across to another colour boundary (2), again on the next row up (3), across to the end of the page (4), and finally back to the the left edge of the bounding box that extends beyond the section bottom (5).

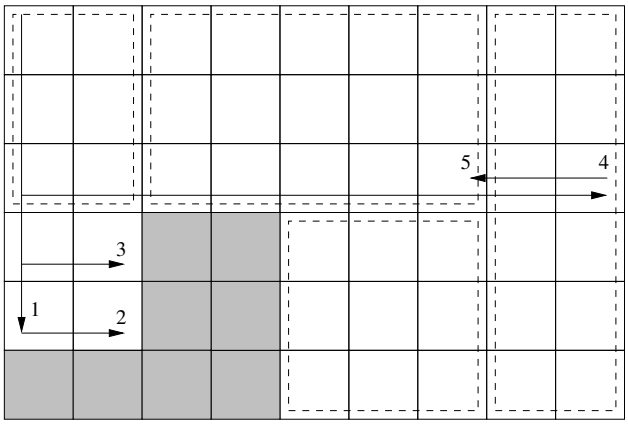


Figure 4.3: Segment identification

In the ideal case, the set of grid cells identified is then marked as a section. However, it is possible that the section still intersects an inline bounding box because it could not be extended far enough to the right. If this happens, the section bottom is moved up by one cell and the process of moving right beings again. In extreme cases, this may result in the segment only taking up the original 1×1 cell that it started with.

Once a segment has been found that begins from the cursor position and does not intersect any inline bounding boxes, the cursor is moved to the next available position in the grid. It first moves past the right edge of the current segment, and upon reaching the end of the page goes to the next line and then right again until it finds a cell that has not yet been assigned to a segment.

When a segment is identified, the set of rendering objects is examined to determine the list of nodes within the segment. Each of these is then inspected to determine if there is any content actually in the segment. This decision is made based on whether or not the segment contains any form controls, non-empty text nodes, or images greater than 10×10 pixels in size. The segment is only added to the list if meets these criteria.

6. `addFloatSegments()`

   Once all of the segments have been identified as described above, extra segments are added for any nodes in the tree that have the `float` or `position` property set. Each of these has their own subtree which resides outside the normal flow of rendering objects in the document, and is automatically assigned its own segment. This means that segments can potentially overlap each other on the page, but the set of nodes associated with overlapping segments is kept separate.

7. `mergeAdjacent()`

   Finally, all of the segments created are inspected to determine the spatial relationships between them. There are three criteria under which segments may be merged. Each of these is performed separately for all segments before moving onto the next criteria.

   The first case is when one segment is wholly contained within another, based on their relative sizes and positions on the rendered page. This is an additional criteria added to deal with floats that are inside another segment which do not cross the boundaries of that segment, and is used to deal with cases where there are a lot of floating images present in a segment. The second criteria is title/body pairs, where one segment is directly above another. For this merging to occur, the top segment must have only a single line box or a single text node, and the bottom segment must contain more that one line box. The third criteria is the merging of vertically adjacent segments with the same background colour, as described in Section 3.1.3.

The result of this process is an array of `Segment` objects, each of which has the following properties:

`nodes`	The set of nodes associated with the segment. In cases where whole or partial subtrees of nodes are present in the segment, only the top-level nodes are stored.
`exclusions`	Top-level nodes of subtrees that do *not* reside within the segment, but whose ancestors are present in the `nodes` list. These are added for floats that have their own segment.
`boxes`	All of the RenderBox objects associated with the segment
`rect`	The position and size of the segment, measured in pixels
`gridRect`	The position and size of the segment, measured in grid cell coordinates
`colour`	The background colour of the segment
`floatRoot`	Whether or not this segment is associated with a node that has either the `float` or `position` properties set

4.4.2 Segment Analysis

Segment analysis is the process of calculating the values of all feature variables for a segment. This is performed during the training process when generating the analysis file for each page, and when a page is loaded in classification mode. The set of values is stored in an array of `VariableAssignment` objects, each of which contains the corresponding value and a method for converting the proportional or count value into a quantised value that can be used with the naive Bayes classifier.

The initial version of the analysis algorithm used a separate tree iteration for every element type, but this was found to be inefficient due to the costs of traversal. Instead, the algorithm was modified to perform a single iteration over the tree, calculating all of the values for the feature variables as it goes along.

For every element type, a `TagSegmentInfo` structure is maintained which stores variables that are used for the calculation of feature variables, and are updated as the algorithm iterates through the tree. This structure has the following properties:

`matchingChars`	The number of characters found inside text nodes that are descendants of the element type
`matchingRegion`	A region object, which is a set of rectangles corresponding to the areas on the page that descendant elements take up
`count`	The number of elements of the particular type

Two additional variables are used for storing information general to the segment

`totalChars`	The total number of characters inside text nodes in the segment
`segmentBounds`	The bounding box of all rendering objects in the segment

A depth-first traversal of the tree is performed, and at each node, the global variables and the `TagSegmentInfo` structures corresponding to the names of the current node and all ancestor nodes are updated. By the keeping the names of ancestor nodes in a stack, the algorithm only has to consider a small number of element types at each stage, instead of the whole available set. Additionally, the `TagSegmentInfo` structures are stored in a hash table, which makes accessed to them very efficient.

For matching element types, the `matchingChars` variable is incremented by the number of characters if the current node is a text node. `matchingRegion` is updated by adding the rectangle bounding all of the rendering boxes associated with the current node. `count` is incremented only for the element type corresponding to the current node. The global `totalChars` variable is updated in the same way as `matchingChars`, and the `segmentBounds` box is extended to incorporate the area taken up by all rendering boxes associated with the node.

At the end of the traversal, the values of the feature variables for each element type can be calculated based on the corresponding `TagSegmentInfo` structures. The proportional value *`element`*`-text` is simply the number of

matching characters divided by the total characters. *element*-rendered is the bounding box of all the rectangles in matchingRegion divided by the area of segmentBounds. count is already at the correct value. Each of these values is stored in the appropriate element of the VariableAssignment array, which is subsequently used for training or classification.

Chapter 5

Evaluation

Evaluation of the algorithms presented here was performed using the prototype implementation with several sets of web pages. These pages were obtained from real-world websites, and were chosen in such a manner as to reflect the types of pages that would commonly be viewed by mobile web users. The selection of pages was undertaken with the aim of obtaining some pages that worked well with the segmentation algorithm, and others which did not work well, in order to get a balanced view of the system's effectiveness. The selection of pages was taken from 100 of the top news sites, which are a popular destination for many web users, and also a more general collection of pages selected on less specific criteria. Each page was manually classified, and then used for training and measurement.

Three sets of sample pages were used:

- **news_articles**

  This set consists of pages containing articles from news sites. These are typically structured with a large body of text in the centre corresponding to a *Content* segment, with other peripheral information above, below and to the sides. These pages typically have either *Sidebar* or *Navigation* sections to the left and/or right of the content, with *ExtraInfo* sections at the bottom of the page, and *Logo* or *Advertising* at the top or bottom.

  These pages were specifically chosen for use as the primary set of test data because they are handled well by the segmentation algorithm, and have well-defined segments that can be classified with little ambiguity. It is important to have such a set of pages in order to evaluate the classification and training algorithms without needing to be overly concerned about effects of poor segmentation or confusion about types of content. A total of 100 pages with 1066 segments is included in the set.

  Only one page per site was chosen, because the use of multiple pages with very similar sets of feature variables has little impact on the input to the training process. This was confirmed with separate experiments measuring the classification accuracy when training and testing on many pages with the same layout, which resulted in almost perfect accuracy.

- **news_frontpage**

  This contains pages taken mostly from the same set of sites as **news_articles**, except using the front page of each site instead of an actual article. The layouts of these pages are similar those of the articles, and in general are structured in a reasonably logical manner, with well defined segment divisions and contents. The main difference is that the segments of the page manually classified as *Content* typically contain different structures. While *Content* segments in article pages consist mainly of text and a small number of images, front pages of sites often contain more pictures and links, and in some respects can be considered navigational elements as well as body text. Despite this, they are marked as *Content*, because in this context they form the main part of the page.

This set was chosen to be more diverse in terms of the structures presented in each segment, while still retaining a reasonable level of clarity as to which classes should be assigned to segments during training. Again, 100 pages, containing 1576 segments, were chosen from different sites.

- **general**

  While the first two sets contain pages of a similar, well defined structure, the **general** set contains a wider variety of pages. These were obtained by downloading the first 100 results of a search performed on the keyword "technology". The set consists of a wide variety of pages, including company home pages, research projects, university websites, articles, and product information, with a total of 713 segments. The layouts of these pages are as diverse as the types of sites encountered, with some being information-rich, and others making heavy use of graphics with little actual content.

  While a few of the pages were well structured and came from complex, professionally designed sites such as those used in the previous two sets, many had structures that were not handled well by the segmentation algorithm. This resulted in pages that had segments containing several different types of content, and in some cases a single segment that covered the whole page. Manual classification was also more difficult, because in many cases there was significant ambiguity about which of the available classes should be chosen for a segment. This can be partially attributed to the limited range of classes allowed by the system, but also because much of the content on these pages did not fit into a well-defined category.

  The purpose of including this set was to see how well the system performed on the wide variety of page layouts found on the web, not all of which work well with the segmentation algorithm. These were expected to give poorer results than well-structured pages, and were thus used to determine the impact that such pages would have on the effectiveness of the system.

5.1 Classification Accuracy

The first task in evaluating the performance of the system is to determine how well the classifier determines classes for segments that it has already been trained on. In this context, the same data set is used as both the training set and the test set. Since the data is already known, this should theoretically give better results than when the two sets contain different data. This is because of the way in which the naive Bayes classifier works; for a given class, certain configurations of variable assignments have higher probabilities than others. The configurations which result in the highest probabilities are based on those observed during training, so when classifying a page that has been used for training the system, these configurations will be observed again. When testing with data separate from the training set, as is done in subsequent experiments, the results are more dependent on how similar pages in the training and test sets are. Thus, the focus on selecting the training set is to pick data that is likely to be as similar as possible to data encountered by the classifier in real-world usage.

Two measures of classifier performance were used: *recall* and *precision* [36]. Recall, denoted ρ, is the probability that if a particular class has been manually assigned to a segment during training, then it will be chosen by the classifier. Precision, denoted π, is the probability that if a particular class is chosen for a segment by the classifier, then it is correct according to the class manually assigned during training. The difference between the two is that recall refers to the degree to which the manual classifications are correctly repeated by the automated classifier, and precision indicates how often the decisions made by the automated classifier are correct. A low precision indicates the occurrence of a lot of false positives, because much of the time a given class is chosen for a segment, it has not been manually classified as such. A low value for recall represents a large number of false negatives, where segments that should be given a certain classification are not. Recall is the measure of most interest in this context because it represents how good the classifier is at emulating the behaviour of a person performing manual classification.

Using ϕ to represent the classifier, the recall of a class C_j can be expressed in probabilistic terms as:

$$\rho_j \quad = \quad P\left(\text{classified}\left(S_i\right) = C_j \mid \phi, \text{trained}\left(S_i\right) = C_j\right)$$

The calculation of recall for a given data set is based on the number of segments that had the class manually assigned, and of those, the number that also had the class chosen automatically by the classifier. By using the notation $\|X\|$ to represent the number of elements in a set, the computation of recall can be expressed as follows:

$$\rho_j \quad = \quad \frac{\|\text{trained}\left(S_i\right) = C_j \wedge \text{classified}\left(S_i\right) = C_j\|}{\|\text{trained}\left(S_i\right) = C_j\|}, \qquad i = 1 \ldots \|S\|$$

Similarly, precision is specified as the following probability measure:

$$\pi_j \quad = \quad P\left(\text{trained}\left(S_i\right) = C_j \mid \phi, \text{classified}\left(S_i\right) = C_j\right)$$

The precision of a particular class is obtained from the number of segments with the class chosen for them by the classifier, and out of those, how many were also assigned the same class during training:

$$\pi_j \quad = \quad \frac{\|\text{classified}\left(S_i\right) = C_j \wedge \text{trained}\left(S_i\right) = C_j\|}{\|\text{classified}\left(S_i\right) = C_j\|}, \qquad i = 1 \ldots \|S\|$$

The overall accuracy of the classifier is obtained from a weighed average of all of the precision or recall values, which is the same as:

$$\text{Accuracy} \quad = \quad \frac{\|\text{classified}\left(S_i\right) = C_j \wedge \text{trained}\left(S_i\right) = C_j\|}{\|S\|}, \qquad i = 1 \ldots \|S\|$$

The precision and recall obtained for each of the data sets is shown in Table 5.1. This also includes the results from performing measurement on a fourth data set, **combined**, which contains all of the pages from the previous three sets.

Although the overall accuracy is not particularly high, and only moderate precision and recall were obtained for most classes, some did give fairly good results. *Content* and *Decoration* in particular had high recall values, which can be attributed to the fact that all segments of those classes have reasonably similar values for their feature variables. For some of the other classes, the appearance of segments can vary widely, and in a lot of cases the difference between classes is hard to determine just based on the limited aspects of structure and appearance encoded in the feature variables. The lowest accuracy was obtained for the combined data set, which had a much wider variety of segments to classify than the others. The diversity in structure and appearance between pages in the three data sets makes it harder to take into account all of the different factors which contribute to the classification of a segment. As expected, the highest accuracy was obtained for the **news_articles** data set, due to the fact that all of the pages it contains use similar types of layouts.

	news_articles		**news_frontpage**		**general**		**combined**	
	π	ρ	π	ρ	π	ρ	π	ρ
Content	83%	90%	62%	54%	84%	62%	64%	56%
Title	26%	56%	23%	50%	54%	70%	18%	36%
Navigation	88%	65%	74%	63%	55%	54%	71%	56%
Advertising	51%	46%	37%	35%	58%	67%	37%	39%
Logo	46%	56%	39%	41%	38%	45%	38%	34%
Sidebar	40%	65%	46%	49%	54%	69%	35%	39%
ExtraInfo	50%	60%	44%	48%	54%	58%	33%	40%
Form	79%	71%	68%	81%	74%	83%	73%	71%
Decoration	35%	81%	47%	85%	67%	83%	37%	77%
Accuracy	65%		56%		62%		52%	

Table 5.1: Classification accuracy with test set = training set

The results shown above were observed when using exactly the same set of documents in the test and training set. To provide a more robust measure of classifier accuracy, another experiment was performed using *split data sets*, in which the data set is divided up into two separate portions — one for training, and one for testing. The training portions consisted of 90% of the original data, with the other 10% used for testing. 10 runs were performed for each data set, each one using a different 10% for the test pages. This means that every page was tested once, as with with the previous set, but the classifier used in each case was built using data from all of the other documents except those in the group of pages being tested. This resulted in significantly lower accuracy for the classifiers, particularly with the **general** set which went from 62% accuracy to 39%. The results are shown in Table 5.2.

	news_articles		**news_frontpage**		**general**		**combined**	
	π	ρ	π	ρ	π	ρ	π	ρ
Content	55%	86%	48%	52%	53%	63%	55%	56%
Title	0%	0%	0%	0%	0%	0%	11%	3%
Navigation	78%	58%	62%	49%	37%	37%	64%	50%
Advertising	31%	27%	24%	20%	21%	22%	28%	34%
Logo	41%	38%	30%	30%	22%	21%	28%	24%
Sidebar	24%	34%	34%	39%	21%	18%	26%	29%
ExtraInfo	40%	47%	25%	25%	30%	28%	27%	32%
Form	72%	51%	63%	59%	44%	30%	68%	66%
Decoration	22%	47%	39%	66%	69%	76%	36%	67%
Accuracy	52%		43%		39%		45%	

Table 5.2: Classification accuracy with split data sets

A number of interesting changes occurred in the results obtained for certain classes. The precision and recall of *Title* segments went down to 0 in all but the **combined** data set. This can be explained by the fact that there were only a very small number of *Title* segments in the data sets, and the correct classification of a portion of them in the previous experiment was because the classifier had already been trained to recognise segments with those particular sets of feature variables as *Title* segments. The nature of these tended to vary, with a number of different font styles being used for titles, and some consisting only of a single image. Therefore, classification of unseen title segments became much more difficult because the few samples that were provided varied too much.

Data splitting had less of an effect on other types of segments. In particular, the percentage of *Content* segments in **news_articles** that were correctly classified remained high, since there is a lot of structural and visual similarity between these segments in the data set. The *Content* segments in the test sets were therefore very similar to those on which the system had been trained, and thus had the correct classification assigned to them in the vast majority of cases. However, precision went down a lot for the class, indicating lot of false positives where other segments had been incorrectly classified as *Content*.

Results obtained from split data sets are more representative of those likely to be obtained during normal usage, because in practice, a user would be unlikely to encounter the same pages that the system had been trained on. When performing training, the samples used should ideally be representative of those that would be viewed in practice, and therefore training on pages with common layouts is a good strategy for obtaining reasonable classifications. Training the system on pages from many popular websites increases the chances that the training set will correspond closely to the pages actually encountered during normal usage.

To determine how well the classifiers worked against different data sets, each possible combination of data sets was tested. The results obtained are shown in 5.3, in which the following abbreviations are used: NA for **news_articles**, NF for **news_frontpage** and G for **general**. As can be seen, most of the precision and recall values are less than 50%, indicating that cross-usage of classifiers is not very effective under the conditions present here.

	NA/NF		NA/G		NF/NA		NF/G		G/NA		G/NF	
	π	ρ	π	ρ	π	ρ	π	ρ	π	ρ	π	ρ
Content	50%	26%	60%	29%	40%	78%	63%	56%	25%	89%	30%	42%
Title	0%	0%	4%	10%	0%	0%	0%	0%	0%	0%	11%	14%
Navigation	61%	65%	28%	40%	87%	57%	31%	35%	56%	24%	53%	31%
Advertising	32%	31%	18%	33%	30%	32%	13%	29%	33%	9%	35%	16%
Logo	38%	34%	23%	11%	30%	32%	22%	11%	27%	35%	20%	33%
Sidebar	26%	32%	12%	16%	21%	34%	24%	23%	5%	12%	15%	23%
ExtraInfo	21%	26%	20%	32%	37%	29%	25%	28%	35%	26%	21%	20%
Form	63%	64%	50%	57%	71%	67%	55%	53%	50%	28%	44%	30%
Decoration	47%	65%	53%	53%	22%	55%	48%	58%	20%	39%	41%	48%
Accuracy	45%		32%		51%		40%		30%		31%	

Table 5.3: Classification accuracy with test set $\neq$ training set

5.2 Class Merging

One of the main limitations to the set of feature variables used for classification is that it is based only on the structural and visual characteristics of segments. Often when performing manual classification during the training process the decision about which class to assign to a particular segment was based on additional factors such as the position of the segment on the page, the textual content or nature of images, or the relationship to other segments. The latter occurred quite often in situations where parts of the document that should ideally reside in the one segment were split up into multiple segments due to the fact that they had different background colours, such as a navigation menu split into multiple parts.

Because of these limitations, certain types of segments often got confused with each other. The results obtained during the experiments described above showed a large number of classification errors where segments of one type were classified as another. Certain segment classes have a lot of similarity between them in terms of how they look on screen. For example, the difference between a logo and an advertisement is something that normally can only be determined by looking at the image itself, and the distinction can usually only be made by a human.

The segments most often classified as other types in the **news_articles** data set are shown in Table 5.4. For most classes, the mis-classifications that occurred gave no clear indication as to what type of segment each is most similar to. A few, however, did have other classes that they were commonly confused with. The highest misclassification rate is between *Title* and *Decoration* — this is unsurprising given that many titles are constructed as images containing words, which are difficult to tell apart from purely decorative images. However, this figure is based on only a small number of sample segments, so it is difficult to draw conclusions from. Similarly, confusion between *Logo*, *Advertising* and *Decoration* is evident, which is to be expected, since these types of segments usually consist only of images, and the feature variables represent only the proportion of the segment taken up by images, not the content of the images themselves.

Segment	Misclassified as	
Content	*Navigation* 4%	*Sidebar* 4%
	Advertising 2%	
Title	*Advertising* 11%	*Decoration* 33%
Navigation	*Content* 3%	*Sidebar* 9%
	Title 1%	*ExtraInfo* 10%
	Advertising 3%	*Form* 1%
	Logo 6%	*Decoration* 3%
Advertising	*Content* 1%	*Sidebar* 7%
	Title 2%	*ExtraInfo* 7%
	Navigation 13%	*Form* 1%
	Logo 17%	*Decoration* 4%
Logo	*Title* 1.00%	*ExtraInfo* 1.00%
	Navigation 8.00%	*Form* 6.00%
	Advertising 12.00%	*Decoration* 12.00%
	Sidebar 4.00%	
Sidebar	*Content* 4.00%	*ExtraInfo* 9.00%
	Navigation 5.00%	*Form* 3.00%
	Advertising 4.00%	*Decoration* 4.00%
	Logo 5.00%	
ExtraInfo	*Content* 1.00%	*Logo* 2.00%
	Title 3.00%	*Sidebar* 7.00%
	Navigation 14.00%	*Form* 2.00%
	Advertising 4.00%	*Decoration* 8.00%
Form	*Content* 1.00%	*Sidebar* 12.00%
	Navigation 1.00%	*ExtraInfo* 1.00%
	Advertising 0.00%	*Decoration* 9.00%
	Logo 4.00%	
Decoration	*Title* 3.00%	*ExtraInfo* 6.00%
	Advertising 10.00%	

Table 5.4: Segment mis-classifications

The *Content* class was misclassified in only a small number of cases, with non-content segments rarely classified as *Content*. For this reason, it was decided to perform another set of experiments in which the set of classes was restricted to *Content* and *Other*. Based on the high division between classifications of the two, and the simplification gained by placing all segment types that are not part of the main page content into a single category, this was expected

to be an effective approach.

The classifiers for each data set were re-trained after modifying the prototype to accept only the two classes. This experiment made use of the existing manual classification files by treating any segment not classified as *Content* as having the class *Other*. The measurement process was run on each, and the weighted averages of precision and recall were better than for the previous experiments. As can be seen from the results in Table 5.5, very high recall was obtained for the **news_articles** set. The recall was also improved slightly for both **news_frontpage** and **general**. The weighted averages were significantly higher due to the much improved classification accuracy for the *Other* class, which is to be expected, since this class represents the combination of different classes that were often misclassified as each other. Table 5.6 shows the results of this strategy when using split data sets, as described previously. Only a small decrease in accuracy occurred in this case.

The most relevant measure is the recall obtained for *Content* segments, since this determines how effectively the desired end result is obtained. That is, it determines how often a user will see the main content of a page when visited in their web browser. Page extracts containing the other segments can be switched to after loading, and since these will be accessed less frequently, the accuracy of their classification is less important. As long as the main page content is identified correctly, this means that the overall aims are achieved. The high classification accuracy for **news_articles** provides a promising view of this, however it must be considered that in reality the classifier must be based on a wide variety of pages, rather than a specifically selected set of well-structured pages. For normal usage, based on the results obtained for the wider variety of pages in the **general** set, a user can expect to see the main page content two out of three times they visit a page. In other cases, they will have to switch to a different page extract to see the main body of the page.

Grouping the other classes together results in only two page extracts being generated, rather than one for each of the different non-content classes listed previously. Given that the user is normally going to be interested primarily in the *Content* segments, having all other parts displayed in the one extract is not a major problem. The only visible difference is that instead of being able to jump to specific parts such as *Navigation* or *ExtraInfo*, the user will switch to *Other* and have to scroll through all of the page elements contained there. Based on the relatively low expected frequency of such actions, and the fact that the size of the extract is still smaller than the whole page, this is seen as only a minor inconvenience.

	news_articles		**news_frontpage**		**general**		**combined**	
	π	ρ	π	ρ	π	ρ	π	ρ
Content	76%	92%	52%	63%	71%	67%	50%	64%
Other	99%	97%	89%	84%	84%	86%	90%	84%
Accuracy	97%		79%		80%		80%	

Table 5.5: Classification accuracy after class merging

	news_articles		**news_frontpage**		**general**		**combined**	
	π	ρ	π	ρ	π	ρ	π	ρ
Content	60%	88%	44%	62%	52%	67%	46%	64%
Other	99%	93%	87%	79%	83%	79%	89%	83%
Accuracy	92%		74%		73%		78%	

Table 5.6: Classification accuracy after class merging with split data sets

5.3 Additional Feature Variables

Observations made while manually classifying segments during the training process suggested that taking into account spatial information related to each segment would increase the accuracy of classifications. The feature variables chosen for use with the classifier, however, ignored these properties of segments. To determine the gains that could be made from the incorporation of this information into the classification process, modifications were made to the prototype to add extra feature variables representing this data. While the initial approach considered for representing the location of a segment on the page was the position and size in pixels quantised over a fixed number of values, the spatial relationships between segments were seen as preferable, since they are more closely aligned with the abstract structural representation of the page. Four extra variables were added, each of which has a true or false value for each segment S_i:

- `segment-above`$= \exists S_j \in S,\ \text{top}\,(S_j) < \text{top}\,(S_i)$
  Marks the presence of another segment in the document that is wholly or partially above the given segment
- `segment-below`$= \exists S_j \in S,\ \text{bottom}\,(S_j) > \text{bottom}\,(S_i)$
  Marks the presence of another segment in the document that is wholly or partially below the given segment
- `segment-left`$= \exists S_j \in S,\ \text{left}\,(S_j) < \text{left}\,(S_i)$
  Marks the presence of another segment in the document that is wholly or partially to the left of the given segment
- `segment-right`$= \exists S_j \in S,\ \text{right}\,(S_j) > \text{right}\,(S_i)$
  Marks the presence of another segment in the document that is wholly or partially to the right of the given segment

The resulting classification accuracy measures obtained after implementing this change are shown in Table 5.7 for the same test and training sets, and Table 5.8 for split data sets as used in the previous two experiments. The results show slight improvements in most cases, with the largest being a 15% increase in the recall of *Content* segments for the **general** set. However, this increase was not seen when run with split data sets. The outcome from using these additional variables suggests that other factors are limiting the accuracy obtained for the sample pages used. These are discussed further in the next chapter.

	news_articles		**news_frontpage**		**general**		**combined**	
	π	ρ	π	ρ	π	ρ	π	ρ
Content	77%	92%	54%	65%	75%	82%	51%	65%
Other	99%	97%	90%	85%	91%	87%	91%	84%
Accuracy	97%		81%		85%		81%	

Table 5.7: Classification accuracy after inclusion of spatial information

	news_articles		**news_frontpage**		**general**		**combined**	
	π	ρ	π	ρ	π	ρ	π	ρ
Content	60%	88%	47%	63%	54%	68%	49%	66%
Other	99%	93%	88%	81%	83%	80%	89%	83%
Accuracy	92%		76%		74%		79%	

Table 5.8: Classification accuracy after inclusion of spatial information with split data sets

Chapter 6

Discussion

6.1 Limitations

The poor results found when classifying segments based on the original set of 9 classes can be attributed to three broad types of problems: those related to segmentation, to classification ambiguity, and to the set of feature variables. Poor segmentation results and the unclear purpose of certain sections of some pages cause the classifications to be incorrect or ambiguous, and the limited set of feature variables used to encode various properties of each segment means that certain information that would normally be taken into account by a human is ignored when performing automated classification. While the merging of all non-content classes into one provides an effective way of working around these problems, it would still be desirable to improve the classification accuracy for the larger set of classes so that more page extracts can be produced, enabling users to more easily access the parts of the page they want to view.

6.1.1 Segmentation

The segmentation process is designed for certain types of pages which are well structured and have clear divisions between different parts of the page based on background colours. While this is appropriate for some of the more advanced page designs found on a lot of major portal and news sites, other sites use layouts that divide up the page using different mechanisms. When segmentation is performed on these pages, the resulting set of segments is not always optimal for classification purposes.

When performing training and classification, the assumption is made that the page is divided up into segments which each contain only one type of information. Assigning a single class to each segment is therefore considered a reasonable thing to do. However, the results of the segmentation process on some pages invalidate this assumption by either selecting segments containing more than one type of information, or producing multiple segments that ideally should be joined together. While some steps have been taken to deal with the latter problem, such as the merging of title/body pairs, only the common cases are catered for; other situations exist where merging should occur but doesn't because of the layout or background colours of segments.

If multiple types of information are present in the one segment, it does not make sense to assign a single class to the segment. This causes two problems. One is that during the training process, only one segment type can be selected, and the system is trained to believe that all of the information present in that segment is of the specified class. The classifier thus becomes polluted with partially incorrect training data, resulting in lower accuracy when the subsequent classification results are measured. The second problem is that when automatically classifying a segment, only one class will be chosen. For example, a segment containing both content and navigation elements may be classified as *Navigation*, resulting in the segment being included in a secondary page extract instead of the primary extract. As a result, the user will not see this content initially, and either miss it or be required to switch to the other extract in order

to view it.

Having multiple segments that should ideally be merged together is less of a problem, provided that each individual segment is assigned the correct classification. Since they will all be displayed together in the one extract, the distinction will not be noticeable to the user. However, if they are classified as different types, their contents will be displayed in separate extracts.

Another complication is that the segmentation algorithm assumes that divisions are based solely on differences in background colour. Some pages use a single background for the whole page, with different sections being separated by whitespace or images depicting lines. These divisions are not detected, resulting in another case of multiple types of information being present in the segment. An entire page could thus be identified as one segment, even though it may contain several different types of information.

These problems account for a large proportion of the classification errors found when measuring the **general** set, because a lot of the pages in the set have layouts that do not cleanly separate different parts of the page in the manner expected by the segmentation algorithm. There were a number of pages that had only one segment which covered the entire page, even though there were several different types of information present on the actual page. This made manual classification difficult, and in many cases it was necessary to pick a class based on the type of content most prominent in the segment, while recognising that other classifications would have also been appropriate. Since the actual classification algorithm itself has been proven to be very accurate on the high-quality training data in the **news_articles** set, poor results obtained when measuring classification accuracy can be attributed to factors such as these rather than problems with the classifier itself.

6.1.2 Classification

When performing manual classification, the decision about which class to assign to a segment is sometimes difficult to make. In addition to some of the cases mentioned above, there are other situations where it is not always obvious which class should be assigned to a segment, even if it does contain only one type of information. For example, a set of links and images pointing to other sites could be considered to serve a navigational purpose, but if they are for product promotion purposes they could be also considered advertising. If there are other types of elements such as form controls, the segment could also be considered to be a sidebar. Whenever a confusing situation like this arises during manual classification, the training that the system receives is based on information that is not necessarily 100% correct.

One solution to the problem would be to extend the range of available classes. New classes could be added for special cases where a segment is present for a particular purpose within the context of that site, such as displaying the current date and time, or stock quotes. While being able to classify segments at this level of granularity would be useful, the higher number of classes and the fact that the training and classification occurs based purely on structural and visual properties of the segment would result in an even lower classification accuracy. This would defeat the purpose of having the extra classes if they are rarely identified correctly in practice. Because of this, and also due to the limited number of documents in the data sets used for evaluation, the set of classes was kept small. This allowed each class to be trained on a reasonable number of sample segments.

As shown earlier, reducing the set of classes to just *Content* and *Other* produces much better results than having a larger set of classes. This demonstrates that it is necessary to make a trade-off between the number of classes and accuracy of classification. For the capabilities of the current algorithm and set of feature variables, it seems that reducing the set of classes to two is the only way to get decent levels of accuracy. With additional improvements to the segmentation algorithm and set of feature variables, it is likely that more classes could be supported while still achieving good results.

It would be useful to determine the degree of ambiguity encountered when making manual classification decisions. This could be done by having several different people perform the manual classification process, and comparing the results. Segments which had different classes assigned by different people would thus be considered ambiguous, and the classes chosen for them and the number of such segments would give a good indication of the percentage of errors

in the training data. This approach has been used previously in the area of image classification on web pages [14].

6.1.3 Feature Variables

The set of feature variables chosen to represent the properties of a segment has a significant impact on the classification accuracy. It is important to choose variables which represent the sorts of information that is most likely to be unique to each type of class. The variables used in the initial set of experiments were based solely on structural and visual information about each segment, and considered segments as separate entities without taking into account the relationships between them. The addition of the four spatial properties described in Section 5.3 made a step in this direction, but still only provided a relatively simple view of how each segment fits in with the overall context of the page.

During manual classification, the observation was made that many of the decisions about which class to assign to a segment were based not just on its appearance, but on the semantics of the content and the relationship between the segment and the rest of the page. The text and images included in a segment were often the deciding factor in which class to assign, such as choosing *Navigation* if the segment contained links to other pages on the site, or *Advertising* if the links were to information on a sponsor's site. Segments that were located next to or above other segments of a particular class were often assigned the same class if it looked like they were related. For example, a title placed above an *ExtraInfo* segment would also be labeled as *ExtraInfo*, whereas a title with the same appearance placed above a *Navigation* segment would have that class assigned to it instead.

By adding extra feature variables which take into account more relevant properties of the segments, the classification accuracy could be improved. Determining what these properties are, and finding out which of them have the most impact on accuracy is an endeavor that could improve results. Text classification is probably one of the most useful areas to explore, because classifying the textual content of a segment can provide an indication of its meaning. This would result in feature variables representing the most common words in the document, an approach taken by many document classification systems [40].

Most of the classification errors in the **news_frontpage** data set can be accounted for by the similarity in appearance between different segment types. Many of the segments in this set that were manually classified as *Content* had a high degree of similarity to *Navigation* segments, since much of the main body of news site front pages is a mixture between content and navigation. The distinction was primarily based on the position of the segment on the page, with segments in the middle of the page being classified mostly as *Content*.

6.1.4 Support for HTML Features

Some of the more advanced features of HTML complicate the task of performing segmentation and classification of a page. For simplicity, these were not taken into account in this project, and support for the relevant features was disabled in the rendering engine. Other features, specifically floats and positioned elements, have been dealt with in the segmentation process, as described in Section 3.1.4.

Pages that use frames break the paradigm of a single document per page. Instead of having one DOM tree and rendering area that can be analysed, a page containing a frame set consists of multiple documents. While segmentation and classification could be performed on each of the frames individually, the question arises as to how to present the segments as page extracts. One option would be to combine segments from multiple frames into one document. Another possibility would be to display each of the frames in their original size and position, but with the documents in each individual frame modified to only show the appropriate segments. While both of these are options that could possibly be made to work effectively, for simplicity they were not considered here.

Inline frames, specified by the `IFRAME` tag, present a similar problem. They allow another document to be embedded in a scrolling window inside the main document. A document inside an inline frame may contain multiple segments, and a decision would need to be made as to whether these are to be incorporated into the main document for the purposes of segmentation, or to be treated separately. Again, these were not taken into account in this project.

The use of JavaScript also raises questions about how page extracts should be generated and displayed. The system described here performs segmentation and classification at document load time. It is possible for a script embedded in the document to modify the structure after loading, potentially in ways which would alter the results of this processing were it to be performed again. This means that after such modifications, the page extracts into which the document has been split up may no longer correspond to the optimal choice provided by the initial processing. An example of this is a page displaying a pop-up layer containing some information in response to a user hovering the mouse pointer over a hot spot on the page. This extra layer would most likely be desirable to see on the main extract, however, if the content has been newly added to the document, or was not previously visible, then it would not appear there.

Re-segmentation could potentially be performed after each modification, provided the algorithm can be optimised to the extent that it does not significantly inhibit script execution times. However, the implications to the user interface may be undesirable. Depending on the modifications, it may result in whole segments being added or removed, causing the displayed portion of the document to change in non-intuitive ways. In addition, such effects are commonly intended with a very specific visual appearance in mind, which may not fit in with the way that the layout has been tailored for the small screen.

6.2 Implementation Considerations

There are a number of additional issues that would need to be taken into account when implementing the ideas presented here in a production system. The software written as part of this project was designed for use only with local copies of pages, and operation on a desktop computer with a high speed network connection and powerful display and processing capabilities. Implementation on a low-powered, hand-held device would need to work within the limitations of the hardware and embedded operating system.

A common approach to dealing with the limited processing capabilities and slow network connections of such devices is to use an intermediate proxy server that sits between the client and web servers. Transcoding is performed at the proxy, and the client receives the modified pages instead of having to do the processing itself. Using this approach for performing segmentation and classification would require significant functionality to be added to the proxy server. The proxy would need to actually render the page itself so that segmentation can be performed, which would require the incorporation of a HTML rendering engine into the proxy. The client would then receive only the chosen page extract, and would make additional requests to the proxy if the user chose to view a different extract. In this situation, the simplest way to implement navigation between page extracts would be to include links at the bottom of the HTML code of the modified document; this would allow the standard HTTP communication mechanisms to be used. However, if the client was to have the selection of page extracts built-in to the user interface in another manner such as a menu or separate tabs, it may be necessary to send the extract list separately rather than as part of the main HTML code, which would require either a custom protocol or an extension to HTTP such as additional response headers.

This user interface decision is also an important consideration. Incorporating the ability to switch between different extracts into the interface would provide a more seamless and intuitive way of viewing different parts of the page than just including a list of links. For devices such as mobile phones with very limited controls, this could be implemented as scrolling through a list, allowing the user to move back and forth through the list of extracts. Devices with better capabilities could present the extracts as a series of tabs at the top of the screen, or as a pop-up menu that appears on screen when the user wants to visit a different part of the page. Displaying thumbnails of each extract in a similar manner to WEST [44] would be another potentially effective approach, and would also give a good way of viewing the overall page structure before drilling down to a specific part.

While the approach of performing segmentation and classification on the proxy adds extra overhead to the processing performed on the proxy server, it is still likely to be a practical solution. The cost of rendering a HTML file and performing segmentation and classification is significantly higher than that of just transferring the page, and it is difficult to make quantitative estimates of the performance impact. However, it is not likely to cause a major slowdown, given the capabilities of modern hardware, and the fact that the results of segmentation and classification can

be cached in the same manner as the pages themselves. Additionally, the performance of CPUs included in mobile devices is rapidly increasing, and many products introduced within the last year or two are already capable of handling this type of processing quite easily by themselves. The proxy approach is therefore mainly an advantage for older, less capable devices.

6.3 Suggestions for Future Work

6.3.1 Segmentation

The segmentation process should ideally be able to recognise divisions in the page other than those based on changes in background colour. Some pages use an all-white background with text placed in blocks representing different parts of the page. Identifying long horizontal or vertical blank areas between text would provide a means of distinguishing one block of text from the other, and depending on the amount of spacing between blocks, these could be divided up into different segments. This is more in line with the techniques used in OCR, where scanned pages of text are analysed to find paragraphs and diagrams [2].

Other pages use different methods of separating out sections. Some pages use a horizontal rule, specified by the `HR` tag, to indicate a break in the page. Others use long, thin, vertical images between columns to indicate divisions. It would be beneficial to conduct additional research on common page designs, by examining materials on web design such as books and articles, and through observation of pages found on the web. It is likely that most of these cases could be dealt with in a similar manner to the segmentation algorithm introduced here, with the addition of extra rules for detecting divisions on the page and for merging segments together.

6.3.2 Classification

The decisions made during manual classification took into account a number of factors other than just the visual appearance of the segments. This included the different components present in the segment, the meaning of the text, the purpose of links, the relationship to other segments, and other pieces of information. By representing these details as additional feature variables, the classifier would be able to make more informed decisions. If these variables are carefully selected to include information that effectively encodes unique properties common to segments of a particular class, then the accuracy of classification is likely to increase.

Hard-coded rules could also play a part in the appropriate choice of class. For example, a rule specifying that all segments containing a single image located at the top of the page were to be classified as *Logo* could be used in cases where the choice based on the training data is unclear. If the classification probabilities of two or more classes were roughly equal, then the classifier could fall back to these rules to find an appropriate class.

Allowing multiple classifications per segment is another approach that may be useful in dealing with the fact that some segments contain multiple types of content. Some of the pages inspected during the evaluation phase, particularly in the **general** set, contained segments with combined logos, navigation bars and advertising. It was only possible to select a single class for these and often a decision had to be made arbitrarily. If these were marked with each of the classes that the segment did contain, then the resulting decisions made by the automated classifier would likely include these types. The page extraction algorithm would then include the segment in the extract for each class that was selected for the segment. The classifier would operate based on a separate Bayesian network for each class, and the result of each would be a boolean value indicating whether or not the class matches based on the supplied feature variables.

While the current approach attempts to identify the main body of content within a document based on segment classification, it may be possible to determine this information in other ways. Many sites use the same layout for all pages, or at least a large subset of pages. If, by retrieving several pages from the same site, common sets of elements are found to be present in each, these could be marked as non-content elements. Examples of these include navigation

menus, logos and copyright statements. The main content of interest on each page will not be the same on all pages, therefore page elements that are unique in the set can be marked as such.

This information could also be fed back into the training process. If the set of available segment types was restricted to $\{Content, Other\}$, this would allow for automatic training of the system, where parts of the pages that are different are marked as *Content*, and parts that are the same between pages are marked as *Other*. Such a capability could be harnessed to perform training on much larger collections of documents than would be possible manually. Whether or not this increases the resulting classification accuracy would be an interesting question to pursue.

6.4 Other Useful Applications

Finally, it is worth considering the ways in which data obtained from the classification process can be used for other purposes. There are a number of areas where it can be of benefit to know the importance of one piece of content relative to another. An example is during the indexing process performed by a search engine. Many search engines consider content in certain parts of the document, such as title and meta tags, to be of higher importance than others. Having a classification label for each segment would allow words in some types of segments to be considered more relevant than others, improving the results by ranking documents containing the query terms in important segments above others.

The classification data could also be of use in text-to-speech web browsers, commonly used by the visually impaired. Many existing systems read the page out loud in a top-to-bottom fashion, which in some cases requires the user to listen to long lists of navigation links or other information before getting to the main body of content on the page. These systems could use the classification label of each segment to determine which parts of the document to read first, so that the less relevant content is only read later, or not at all.

Chapter 7

Summary and Conclusions

7.1 Summary

This project has achieved a number of objectives, centred around the overall goal of improving the user experience on hand-held web browsing devices. While the algorithms and software developed are a long way from being at the level necessary for implementation in a real-world system, significant progress has been made in the area which, along with further developments, has the potential to provide an eventual benefit to end-users. The specific outcomes achieved are as follows:

- *Introduction of segmentation and classification algorithms*

  A new algorithm was proposed for splitting up a web page into different segments based on geometric information from the rendered document. This differs significantly from many past approaches which have chosen page divisions based solely on structural information from the HTML source code or DOM tree. The visual representation of a document provides a better source of information for segmentation purposes because it is more closely aligned with what the user sees. This algorithm has been shown to be very effective on well-structured pages, although it does not perform as well on pages with less inherent structure.

  The classification process leverages existing work in the area of machine learning, specifically Bayesian classification. It applies these techniques to the problem of classifying segments, based on feature variables derived from both structural and visual information relating to the segments. While this has been used with other types of documents in the past, the context in which the classification algorithms are applied and the way in which the feature variables are calculated are specific to this project.

- *Means of extracting sections of documents*

  After segmentation and classification, the next stage in the processing of pages is the transformation of documents into a manner suitable for display. Mechanisms were presented for extracting subsets of the document based on portions of the DOM tree associated with each segment, and maintaining formatting attributes and other related elements necessary for preserving as much of the original layout as possible. This is a necessary step in viewing documents on the target devices and essentially provides a bridge between the document analysis procedures and what the user actually sees on screen.

- *Implementation of a prototype system*

  The segmentation, classification, and page extraction algorithms were implemented to provide a means of testing and evaluating their effectiveness. The implementation used existing libraries to handle the parsing and

rendering of HTML files, and made use of information provided from these to obtain structural and visual representations of the document. Through testing with pages obtained from various parts of the web, the algorithms were fine-tuned and optimised to deal with cases not predicted in the original design, such as certain layout techniques. The resulting system also provided mechanisms for performing training and classification of test data, and obtaining measurements about classification accuracy which were used for evaluation.

- *Evaluation of algorithms against sample data*

  A study was carried out to determine the effectiveness of the prototype system on a set of sample pages taken from the web. These were chosen to be representative of the sorts of pages that were likely to be encountered by users of the system, and were used to obtain a general impression of how well the system works in practice.

  Results from some of the tests lead to the investigation of additional ways to improve the classification accuracy, specifically class merging and the inclusion of additional feature variables derived from spatial relationships between segments. These were found to give improvements in classification, resulting in an overall accuracy of 79% for the combined data sets.

- *Identification of limitations and areas for further improvement*

  While the results obtained after class merging may seem promising, it is important to note that the classification of segments into a larger set of classes achieved a significantly lower level of accuracy. The use of only two classes is an acceptable way of dealing with this problem, but being able to correctly classify a larger number of segment types would result in better usability. Based on observations made when performing manual classification, a number of factors were identified which limited the ability to distinguish between different types of segments and choose an appropriate class. Some of the assumptions made by the segmentation algorithm do not hold on all pages, and improvements in this area could also aid in providing appropriate classification.

One of the biggest aspects of the page analysis process needing improvement is the segmentation algorithm. Progress has been made in identifying a method which obtains good results on many pages, but it is clear that further investigation is needed into the types of page layouts in common use around the web. Layouts that need to be dealt with include those using a predominantly white background with borders and other divisions between segments, and pages which are not of a content-centric nature, such as those used as front pages of sites that serve mainly navigational purposes. The work done here serves as a useful basis for developing further techniques to deal with these cases. Such techniques can subsequently be integrated with the overall model of page analysis and modification presented here.

The other main area where further research is needed is the identification of appropriate feature variables for encoding information about individual segments. While performing manual classification, it became evident that the definitions given for each class need to be clearer so that classification decisions can be made more easily, and possibly extra classes added to cater for other types of segments. It was also noted that the set of feature variables calculated during segment analysis needs to cover a wider range of properties of the segment. The fact that the segments can be identified effectively by a human suggests that the addition of extra feature variables could produce better results. Semantic information about the meaning of each segment could play a role here, and the integration of text classification techniques with the existing processes is likely to improve the effectiveness of training and classification.

Structural analysis of web pages is a complex area in which many factors play a part. The variety of page designs throughout the web, and the purposes for which they are used suggest that much further work needs to be done in this area before a full understanding can be gained about how to divide up web pages into distinct parts. This project has made some small steps toward this which hopefully provide some insight into useful approaches to the problem. Additionally, it has demonstrated applications for segmentation and classification in the area of web browsing on small screen devices.

7.2 Conclusions

The problem of finding an effective way of displaying web pages that takes into account the hardware limitations of hand-held devices is a challenging one. The development of the web through the 1990s saw a major focus on desktop usage, resulting in the vast majority of sites being targeted at powerful computers with high-resolution displays. Advances made by mobile phone and PDA manufacturers, combined with increasing demand by many people for access to the web anytime, anywhere means that there is a significant need for improved usability in these devices.

This project has investigated one approach to displaying web pages which involves the modification of document structure to present a subset of the information normally included in a page. While the approach has found to be reasonably effective, it is clear that much more work is necessary before it is ready for end-users. Many issues need to be addressed concerning the analysis and classification of web pages, as well as dealing with a wider variety of page layouts. Additionally, support for some of the more advanced functionality provided by modern web browsers needs to be integrated with the way in which pages are analysed, modified and presented.

The ideas introduced here have the potential to be used as a basis for future research in the area. It is hoped that these will lead to improvements in the way that web pages are analysed, and the way in which structure is used to provide new functionality in web browsers. In particular, analysis based on visual aspects of a rendered document has achieved relatively little attention in the past, and the exploration of this concept here provides insights which may be useful in a range of areas in addition to that of small screen rendering. Further research incorporating these ideas and others will hopefully result in useful advances in the field.

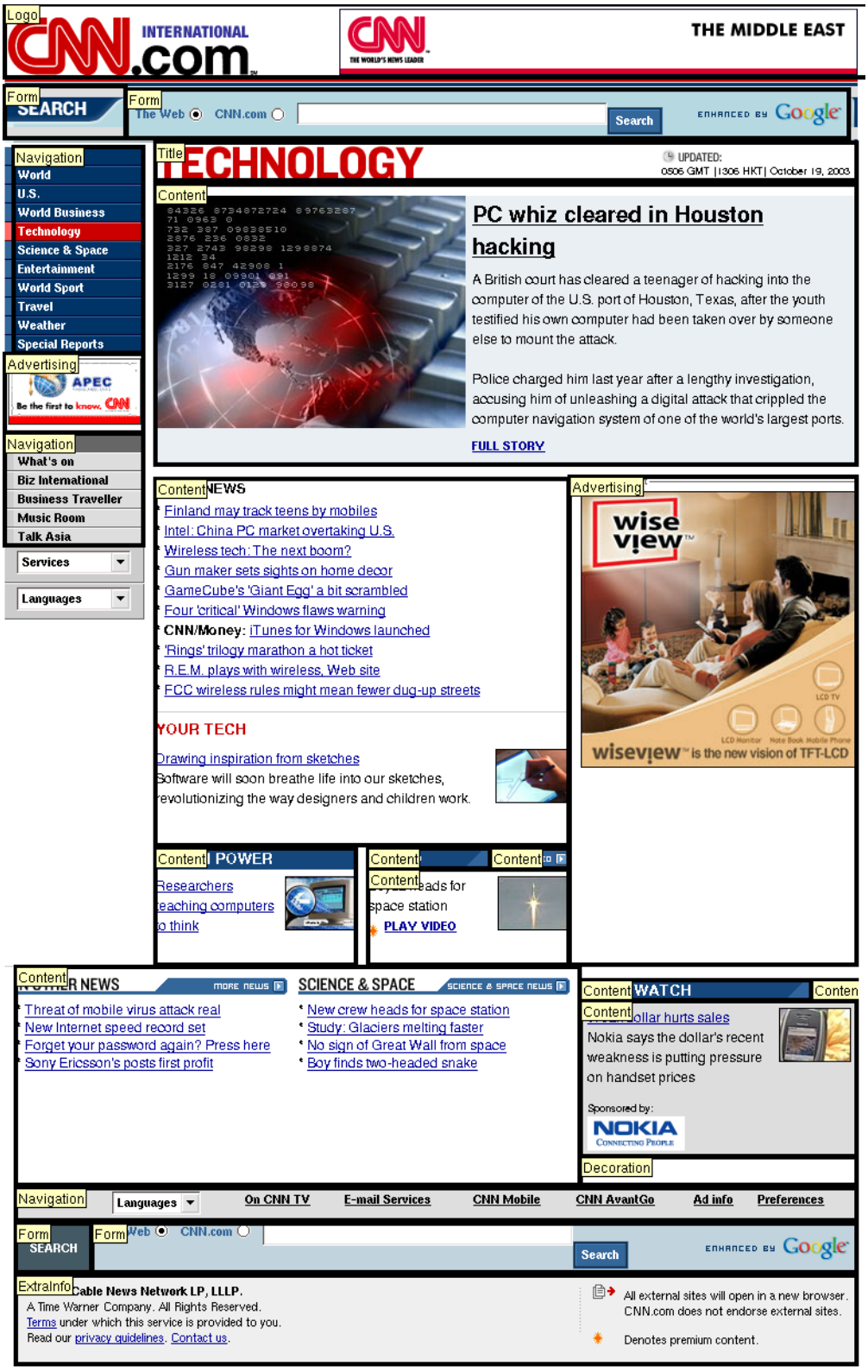
Appendix A

HTML Tags

A	ABBR	ACRONYM	ADDRESS
APPLET	AREA	B	BASE
BASEFONT	BDO	BIG	BLOCKQUOTE
BODY	BR	BUTTON	CAPTION
CENTER	CITE	CODE	COL
COLGROUP	DD	DEL	DFN
DIR	DIV	DL	DT
EM	FIELDSET	FONT	FORM
FRAME	FRAMESET	H1	H2
H3	H4	H5	H6
HEAD	HR	HTML	I
IFRAME	IMG	INPUT	INS
ISINDEX	KBD	LABEL	LEGEND
LI	LINK	MAP	MENU
META	NOFRAMES	NOSCRIPT	OBJECT
OL	OPTGROUP	OPTION	P
PARAM	PRE	Q	S
SAMP	SCRIPT	SELECT	SMALL
SPAN	STRIKE	STRONG	STYLE
SUB	SUP	TABLE	TBODY
TD	TEXTAREA	TFOOT	TH
THEAD	TITLE	TR	TT
U	UL	VAR	

Appendix B

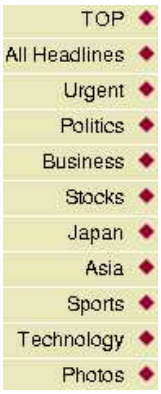
Segmentation Examples


Logo
CNN INTERNATIONAL .com
THE MIDDLE EAST
Form
SEARCH
Form
The Web
CNN.com
Search
ENHANCED BY Google
Navigation
World
U.S.
World Business
Technology
Science & Space
Entertainment
World Sport
Travel
Weather
Special Reports
Advertising
APEC
Be the first to know. CNN
Navigation
What's on
Biz International
Business Traveller
Music Room
Talk Asia
Services
Languages
Title
TECHNOLOGY
UPDATED:
0506 GMT | 1306 HKT | October 19, 2003
Content
PC whiz cleared in Houston hacking
A British court has cleared a teenager of hacking into the computer of the U.S. port of Houston, Texas, after the youth testified his own computer had been taken over by someone else to mount the attack.
Police charged him last year after a lengthy investigation, accusing him of unleashing a digital attack that crippled the computer navigation system of one of the world's largest ports.
FULL STORY
Content
NEWS
Finland may track teens by mobiles
Intel: China PC market overtaking U.S.
Wireless tech: The next boom?
Gun maker sets sights on home decor
GameCube's 'Giant Egg' a bit scrambled
Four 'critical' Windows flaws warning
CNN/Money: iTunes for Windows launched
'Rings' trilogy marathon a hot ticket
R.E.M. plays with wireless, Web site
FCC wireless rules might mean fewer dug-up streets
YOUR TECH
Drawing inspiration from sketches
Software will soon breathe life into our sketches, revolutionizing the way designers and children work.
Advertising
wise view
wiseview is the new vision of TFT-LCD
Content
POWER
Researchers teaching computers to think
Content
Content
Content
space station
PLAY VIDEO
Content
MORE NEWS
Threat of mobile virus attack real
New Internet speed record set
Forget your password again? Press here
Sony Ericsson's posts first profit
SCIENCE & SPACE
SCIENCE & SPACE NEWS
New crew heads for space station
Study: Glaciers melting faster
No sign of Great Wall from space
Boy finds two-headed snake
Content
WATCH
Content
Nokia says the dollar's recent weakness is putting pressure on handset prices
Sponsored by:
NOKIA
Decoration
Navigation
Languages
On CNN TV
E-mail Services
CNN Mobile
CNN AvantGo
Ad info
Preferences
Form
SEARCH
Form
Web
CNN.com
Search
ENHANCED BY Google
ExtraInfo
Cable News Network LP, LLLP.
A Time Warner Company. All Rights Reserved.
Terms under which this service is provided to you.
Read our privacy guidelines. Contact us.
All external sites will open in a new browser.
CNN.com does not endorse external sites.
Denotes premium content.

Appendix C

Segment Classes

Content



Navigation

Sidebar

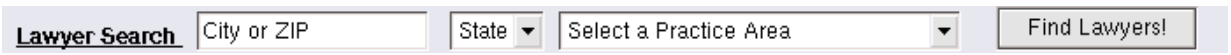
Decoration

Title

Logo

ExtraInfo

Advertising



Form

Appendix D

Measurement Output

```
# ./cabrowser -m -a news_articles/an.xml -c news_articles/training.xml
Running in measurement mode
Training set analysis file: news_articles/an.xml
Training set classification file: news_articles/training.xml
Test set analysis file: news_articles/an.xml
Test set classification file: news_articles/training.xml
www.arabicnews.com_ansub_Daily_Day_031017_2003101704.html.war
  Segment 0: trained=Logo classified=Logo
  Segment 1: trained=Content classified=Content
  Segment 2: trained=Navigation classified=Navigation
  Segment 3: trained=Navigation classified=Navigation
  Segment 4: trained=Form classified=Form
  Segment 5: trained=Navigation classified=Navigation
  Segment 6: trained=Advertising classified=Advertising
  Segment 7: trained=Advertising classified=ExtraInfo
  Segment 8: trained=Advertising classified=Advertising
greenvilleonline.com_news_2003_10_17_2003101717107.htm.war
  Segment 0: trained=Logo classified=Logo
  Segment 1: trained=Navigation classified=Navigation
  Segment 2: trained=Navigation classified=Sidebar
  Segment 3: trained=Content classified=Content
www.philly.com_mld_inquirer_news_nation_7023679.htm.war
  Segment 0: trained=Logo classified=Logo
  Segment 1: trained=Form classified=Form
  Segment 2: trained=Advertising classified=Logo
  Segment 3: trained=Navigation classified=Navigation
  Segment 4: trained=Content classified=Content
  Segment 5: trained=Navigation classified=Navigation
  Segment 6: trained=Navigation classified=Navigation
  Segment 7: trained=Navigation classified=Navigation
  Segment 8: trained=Navigation classified=Navigation
  Segment 9: trained=Navigation classified=Navigation
  Segment 10: trained=Advertising classified=Advertising
  Segment 11: trained=Navigation classified=Navigation
  Segment 12: trained=Navigation classified=Navigation
  Segment 13: trained=Navigation classified=Navigation
  Segment 14: trained=Navigation classified=Navigation
  Segment 15: trained=Form classified=Form
  Segment 16: trained=Navigation classified=Advertising
  Segment 17: trained=Navigation classified=Navigation
www.sciencenews.org_20031018_fob4.asp.war
  Segment 0: trained=Navigation classified=Navigation
```

```
  Segment 1: trained=Sidebar classified=Sidebar
  Segment 2: trained=Sidebar classified=Content
  Segment 3: trained=Content classified=Content
  Segment 4: trained=Decoration classified=Advertising
  Segment 5: trained=ExtraInfo classified=ExtraInfo
www.sfgate.com_cgi_bin_article.cgi_f__c_a_2003_10_19_BALCO.TMP.war
  Segment 0: trained=Content classified=Content
  Segment 1: trained=Advertising classified=Advertising
  Segment 2: trained=Sidebar classified=Sidebar
www.techweb.com_wire_story_TWB20031017S0010.war
  Segment 0: trained=Navigation classified=Navigation
  Segment 1: trained=Content classified=Navigation
  Segment 2: trained=Sidebar classified=ExtraInfo
  Segment 3: trained=Advertising classified=ExtraInfo
  Segment 4: trained=ExtraInfo classified=Sidebar

....

expected=Content actual=Content count=89
expected=Content actual=Title count=0
expected=Content actual=Navigation count=4
expected=Content actual=Advertising count=2
expected=Content actual=Logo count=0
expected=Content actual=Sidebar count=4
expected=Content actual=ExtraInfo count=0
expected=Content actual=Form count=0
expected=Content actual=Decoration count=0
expected=Title actual=Content count=0
expected=Title actual=Title count=5
expected=Title actual=Navigation count=0
expected=Title actual=Advertising count=1
expected=Title actual=Logo count=0
expected=Title actual=Sidebar count=0
expected=Title actual=ExtraInfo count=0
expected=Title actual=Form count=0
expected=Title actual=Decoration count=3
expected=Navigation actual=Content count=12
expected=Navigation actual=Title count=7
expected=Navigation actual=Navigation count=310
expected=Navigation actual=Advertising count=15
expected=Navigation actual=Logo count=28
expected=Navigation actual=Sidebar count=42
expected=Navigation actual=ExtraInfo count=50
expected=Navigation actual=Form count=3
expected=Navigation actual=Decoration count=13
expected=Advertising actual=Content count=1
expected=Advertising actual=Title count=2
expected=Advertising actual=Navigation count=11
expected=Advertising actual=Advertising count=38
expected=Advertising actual=Logo count=14
expected=Advertising actual=Sidebar count=6
expected=Advertising actual=ExtraInfo count=6
expected=Advertising actual=Form count=1
expected=Advertising actual=Decoration count=3
expected=Logo actual=Content count=0
expected=Logo actual=Title count=1
expected=Logo actual=Navigation count=6
expected=Logo actual=Advertising count=9
expected=Logo actual=Logo count=43
expected=Logo actual=Sidebar count=3
expected=Logo actual=ExtraInfo count=1
```

```
expected=Logo actual=Form count=5
expected=Logo actual=Decoration count=9
expected=Sidebar actual=Content count=3
expected=Sidebar actual=Title count=0
expected=Sidebar actual=Navigation count=4
expected=Sidebar actual=Advertising count=3
expected=Sidebar actual=Logo count=4
expected=Sidebar actual=Sidebar count=48
expected=Sidebar actual=ExtraInfo count=7
expected=Sidebar actual=Form count=2
expected=Sidebar actual=Decoration count=3
expected=ExtraInfo actual=Content count=1
expected=ExtraInfo actual=Title count=3
expected=ExtraInfo actual=Navigation count=16
expected=ExtraInfo actual=Advertising count=4
expected=ExtraInfo actual=Logo count=2
expected=ExtraInfo actual=Sidebar count=8
expected=ExtraInfo actual=ExtraInfo count=68
expected=ExtraInfo actual=Form count=2
expected=ExtraInfo actual=Decoration count=9
expected=Form actual=Content count=1
expected=Form actual=Title count=0
expected=Form actual=Navigation count=1
expected=Form actual=Advertising count=0
expected=Form actual=Logo count=3
expected=Form actual=Sidebar count=8
expected=Form actual=ExtraInfo count=1
expected=Form actual=Form count=49
expected=Form actual=Decoration count=6
expected=Decoration actual=Content count=0
expected=Decoration actual=Title count=1
expected=Decoration actual=Navigation count=0
expected=Decoration actual=Advertising count=3
expected=Decoration actual=Logo count=0
expected=Decoration actual=Sidebar count=0
expected=Decoration actual=ExtraInfo count=2
expected=Decoration actual=Form count=0
expected=Decoration actual=Decoration count=25

Total segments 1034, matching 675 (65%)

Content: 89% (89/99)
Title: 55% (5/9)
Navigation: 64% (310/480)
Advertising: 46% (38/82)
Logo: 55% (43/77)
Sidebar: 64% (48/74)
ExtraInfo: 60% (68/113)
Form: 71% (49/69)
Decoration: 80% (25/31)
```

Bibliography

[1] E. J. Bredensteiner and K. P. Bennett. Feature minimization within decision trees. *Computational Optimizations and Applications*, 10:111– 126, 1998.

[2] T. M. Breuel. Two algorithms for geometric layout analysis. In *Proceedings of the Workshop on Document Analysis Systems*, Princeton, NJ, USA, 2002.

[3] O. Buyukkokten, H. Garcia-Molina, A. Paepcke, T. Winograd. Power Browser: Efficient Web Browsing for PDAs. *Proceedings of The CHI 2000 Conference on Human Factors in Computing Systems*, The Hague, The Netherlands, April 1-6, 2000.

[4] O. Buyukkokten, H. Garcia-Molina, A, Paepcke. Accordion Summarization for End-Game Browsing on PDAs and Cellular Phones, *Proceedings of The CHI 2001 Conference on Human Factors in Computing Systems*, Seattle, Washington, 31 March-5 April 2001.

[5] R. Cattoni, T. Coianiz, S. Messelodi, and C. M. Modena. Geometric layout analysis techniques for document image understanding: a review. Technical report, IRST, Trento, Italy, 1998.

[6] S. Chandra, A. Gehani, C. Schlatter Ellis, and A. Vahdat. Transcoding characteristics of web images. Technical Repport CS–1999– 17, Department of Computer Science, Duke University, November 1999 (submitted to WWW9).

[7] Y. Chen, W. Ma, H. Zhang. Detecting Web Page Structure for Adaptive Viewing on Small Form Factor Devices. *Proceedings of The Twelfth International World Wide Web Conference*, 20-24 May 2003, Budapest, Hungary.

[8] N. Christian Juul and N. Jørgensen. WAP May Stumble over the Gateway. Security in WAP-Based E-Commerce. *Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR) 2001*, August 2001, L'Aquila, Italy.

[9] O. de Bruijn, R. Spence, M. Yin Chong. RSVP Browser: Web Browsing on Small Screen Devices. Department of Electrical & Electronic Engineering, Imperial Collage of Science, Technology and Medicine, London, UK, 2001.

[10] European Computer Manufacturers Association. ECMAScript Language Specification. Standard ECMA-262. 3rd Edition - December 1999.

[11] E. Frank, L. E. Trigg, G. Holmes, and I. H. Witten. Naive Bayes for regression. Working Paper 98/15. Hamilton, NZ: Waikato University, Department of Computer Science, 1998.

[12] L.E. Holmquist and C. Ahlberg. Flip Zooming: A Practical Focus+Context Approach to Visualizing Large Information Sets. *Design of Computing Systems: Social and Ergonomics Considerations* (Eds. Smith, M., Salvendy, G.), 1997.

[13] J. Hu, R. Kashi, and G. Wilfong. Document classification using layout analysis. *Proceedings of the First International. Workshop on Document Analysis and Understanding for Document Databases*, Florence, Italy, September 1999.

[14] J. Hu, A. Bagga. Functionality-Based Web Image Categorization. Avaya Labs Technical Report ALR-2003-007. February 2003.

[15] D.A. Hull. Improving text retrieval for the routing problem using latent semantic indexing. W.B. Croft and C.J. van Rijsbergen, editors, *Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval*. Springer Werlag, Heidelberg, DE, 1994.

[16] T. Joachims. Text categorization with Support Vector Machines: Learning with many relevant features. *Machine Learning: ECML-98, Tenth European Conference on Machine Learning*, 1998, pp. 137–142.

[17] M. Frans Kaashoek, T. Pinckney, and J. A. Tauber. Dynamic Documents: Mobile Wireless Access to the WWW. *Proceedings of the Workshop on Mobile Computing Systems and Applications*, December, 1994.

[18] E. Kaasinen, M. Aaltonen, J. Kolari, S. Melakoski, T. Laakko, Two Approaches to Bringing Internet Services to WAP Devices, *Proceedings of the 9th World-Wide-Web Conf. WWW'9*. Amsterdam, 15-19 May 2000, pp. 231-246.

[19] T. Kamada. Compact HTML for Small Information Appliances. W3C NOTE 09-Feb-1998.

[20] G. Karypis and E. Han. Concept indexing: A fast dimensionality reduction algorithm with applications to document retrieval and categorization. Technical Report TR-00-0016, University of Minnesota, 2000.

[21] KDE Homepage. `http://www.kde.org/`

[22] Konqueror Homepage - Web Browser. `http://www.konqueror.org/features/browser.php`

[23] J.T. Kwok. Automated text categorization using support vector machine. *Proceedings of the International Conference on Neural Information Processing*, Kitakyushu, Japan, Oct. 1998, pp. 347-351.

[24] P. Langley, W. Iba, & K. Thompson. An analysis of Bayesian classifiers. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 223–228). San Jose, CA: AAAI Press, 1992.

[25] L. S. Larkey and W. B. Croft. Combining classifiers in text categorization. *SIGIR '96: Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1996, pp. 289–297.

[26] D. D. Lewis and M. Ringuette. A comparison of two learning algorithms for text classification. *Proc. of the Third Annual Symposium on Document Analysis and Information Retrieval*, 1994, pp. 81–93.

[27] D. D. Lewis. Naive Bayes at forty: The independence assumption in information retrieval. *Conference proceedings of European Conference on Machine Learning*, 1998, pp. 4–15.

[28] M. Hori, G. Kondoh, K. Ono, S. Hirose, and S. Singhal. Annotation-based Web content transcoding. *Proceedings of the 9th World Wide Web Conference (WWW-9)*, Amsterdam, 2000.

[29] A. McCallum and K. Nigam. A comparison of event models for Naive Bayes text classification. *AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[30] J. Mynttinen. End-to-end security of mobile data in GSM. Helsinki University of Technology. November 27, 2000.

[31] K. Nagao, S. Hosoya, Y. Kawakita, S. Ariga, Y. Shirai, and J. Yura. Semantic transcoding: making the World Wide Web more understandable and reusable by external annotations. Unpublished manuscript, 1999.

[32] K. Nagao, Y. Shirai, and K. Squire. Semantic annotation and transcoding: Making web content more accessible. *IEEE Multimedia*, 8(2):69–81, April-June 2001.

[33] Opera Software ASA. Opera Small-Screen Rendering. `http://www.opera.com/products/smartphone/smallscreen/`

[34] S. Paek and J. R. Smith. Detecting Image Purpose in World-Wide Web Documents. *Proceedings of the IS&T/SPIE Symposium on Electronic Imaging: Science and Technology - Document Recognition*, San Jose, CA, January 1998.

[35] Trollech. Qt Overview. `http://www.trolltech.com/products/qt/index.html`

[36] V. V. Raghavan, G. S. Wang, and P. Bollmann. A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transactions on Information Syst*ems, Vol 7, No 3, 1989.

[37] M. Ramsay and J. Nielsen. WAP Usability Déjà Vu: 1994 All Over Again. Nielsen Norman Group, December 2000.

[38] M. Röscheisen, C. Mogensen, and T. Winograd. Shared web annotations as a platform for third-party value-added information providers: Architecture, protocols, and usage examples. *Stanford Integrated Digital Library Project STAN-CS-TR-97-1582*, Computer Science Dept., Stanford University, November 1994.

[39] S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall, Englewood Cliffs, NJ, 1995.

[40] M. Sahami. Using Machine Learning to Improve Information Access. Ph.D. thesis, Computer Science Department, Stanford University, 1999.

[41] B. N. Schilit, J. Trevor, D. M. Hilbert and T. Khiau Koh. m-Links: An Infrastructure for Very Small Internet Devices, *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, Rome, Italy, Jul 2001.

[42] S. Scott and S. Matwin. Feature engineering for text classification. I. Bratko and S. Dzeroski, editors, *Proceedings of ICML-99, 16th International Conference on Machine Learning*, pages 379-388, Bled, SL, 1999. Morgan Kaufmann Publishers, San Francisco, US.

[43] F. Sebastiani, Machine learning in automated text categorization, Tech. Rep. IEI-B4-31-1999, Consiglio Nazionale delle Ricerche, Pisa, Italy, 1999.

[44] B. Staffan, L. E. Holmquist, J. Redstrom, I. Bretan, R. Danielsson, J. Karlgren, and K. Franzen. WEST: A Web Browser for Small Terminals. *UIST 99, ACM Symposium on User Interface Software and Technology, CHI Letters, 1*(1), 1999, pp. 187-196.

[45] Wireless Application Protocol Forum Ltd. Wireless Application Protocol: Wireless Markup Language Specification Version 1.2, 4 November 1999.

[46] Wireless Application Protocol Forum Ltd. XHTML Mobile Profile. Version 29-Oct-2001.

[47] World Wide Web Consortium (W3C). HTML 4.01 Specification. W3C Recommendation 24 December 1999.

[48] World Wide Web Consortium (W3C). XSL Transformations (XSLT) Version 1.0. W3C Recommendation 16 November 1999.

[49] World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation 6 October 2000.

[50] World Wide Web Consortium (W3C). Document Object Model (DOM) Level 1 Specification, Version 1.0. W3C Recommendation 1 October, 1998.

[51] World Wide Web Consortium (W3C). Cascading Style Sheets, level 2 CSS2 Specification. W3C Recommendation 12-May-1998.

[52] Yang, Y. and G. I. Webb (2003). Discretization For Naive-Bayes Learning: Managing Discretization Bias And Variance. Technical Report 2003/131, School of Computer Science and Software Engineering, Monash University.