Spring 2014

Document Classification in Support of Automated Metadata Extraction Form Heterogeneous Collections

Paul K. Flynn
Old Dominion University

DOCUMENT CLASSIFICATION IN SUPPORT OF AUTOMATED METADATA EXTRACTION FROM HETEROGENEOUS COLLECTIONS

by

Paul K. Flynn
B.S., December 1989, University of Florida
M.S., December 1998, Old Dominion University

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
May 2014

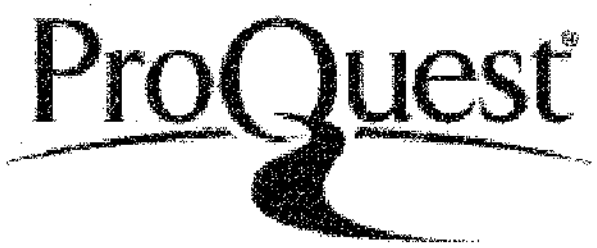
Approved by:

Steven Zeil (Co-Director)

Kurt Maly (Co-Director)

Mohammad Zubair (Member)

Harris Wu (Member)



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

ABSTRACT

Document Classification in Support of Automated Metadata Extraction from Heterogeneous Collections

Paul K. Flynn
Old Dominion University, 2014
Co-Directors: Dr. Steven Zeil
Dr. Kurt Maly

A number of federal agencies, universities, laboratories, and companies are placing their documents online and making them searchable via metadata fields such as author, title, and publishing organization. To enable this, every document in the collection must be catalogued using the metadata fields. Though time consuming, the task of identifying metadata fields by inspecting the document is easy for a human. The visual cues in the formatting of the document along with accumulated knowledge and intelligence make it easy for a human to identify various metadata fields. Even with the best possible automated procedures, numerous sources of error exist, including some that cannot be controlled, such as scanned documents with text obscured by smudges, signatures, or stamps. A commercially viable process for metadata extraction must remain robust in the presence of these external sources of error as well as in the face of the uncertainty that accompanies any attempts to automate "intelligent" behavior. While extraction accuracy and completeness must be the primary goal of an extraction system, the ability to detect and report questionable results is equally important for a production quality system, since it promotes confidence in the system.

We have developed and demonstrated a novel system for extracting metadata. First, a document is examined in an attempt to recognize it as an instance of a known document layout. Then a template, a scripted description of how to associate blocks of text in the layout with metadata fields, is applied to the document to extract the metadata. The extraction is validated after post-processing to evaluate the quality of the extraction and, if necessary, to flag untrusted extractions for human recognition.

The success or failure of the template approach is directly tied to document classification, which is the ability to match the document to the proper template correctly and consistently. Document classification in our system is implemented as a module which applies every template available in the system to a document to find candidate templates that extract any data at all. The candidate templates are evaluated by a validation module to select the best performing template. This method is called "post hoc" classification. Post hoc classification is not only effective at selecting the correct class but it also excels at minimizing false positives. It is, however, very sensitive to changes in the template collection and to poorly written templates.

While this dissertation examines the evolution and all the major components of an automated metadata extraction system, the primary focus is on the problem of document classification. The main thrust of my research has been investigating alternative methods of document classification to replace or supplement post hoc classification. I experimented with machine learning techniques as an additional input factor for the post hoc classification script or the final validation script.

ACKNOWLEDGMENTS

This dissertation would not have been possible without the support and encouragement of my committee members and many other people. My sincere appreciation and gratitude goes to my advisors, Dr. Steven Zeil and Dr. Kurt Maly for their support, encouragement, and guidance through this epic journey. I thank Dr. Mohammed Zubair and Dr. Harris Wu, the members of my committee, for their thorough review of this dissertation and for their valuable feedback.

I am grateful to the faculty, staff, and colleagues at the computer science department of Old Dominion University for their help. I also thank the many former members of the Digital Library Research Group at Old Dominion University for their helpful discussions and contributions to this research project. I also thank my friends and colleagues at Applied Research Associates Suffolk office for their patience and understanding and especially my good friend Evan Madsen who always offered an encouraging word.

I cannot end without thanking my family, Lily, Michelle and PJ for their encouragement and love. It is to them that I dedicate this work.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

A number of federal agencies, universities, laboratories, and companies are placing their documents online and making them searchable via metadata fields such as author, title, and publishing organization. To enable this, every document in the collection must be catalogued using the metadata fields. A typical cataloguing process requires a human to view the document on the screen and identify the required metadata fields such as title, author, and publishing organization, and to enter these values in some online searchable database. Manually creating metadata for a large collection is an extremely time-consuming task. According to Crystal [1], it would take about 60 employee-years to create metadata for 1 million documents. These enormous costs for manual metadata creation suggest a need for automated metadata extraction tools. The Library of Congress Cataloging Directorate recognized this problem [2] and sponsored a study, Automatic Metadata Generation Applications (AMeGA) [3], to identify challenges in automatic metadata creation.

Though time consuming, the task of identifying metadata fields by inspecting the document is easy for a human. The visual cues in the formatting of the document along with accumulated knowledge and intelligence make it easy for a human to identify various metadata fields. Writing a computer program to automate this task is a research challenge. Researchers in the past have shown that it is possible to write programs to

extract metadata automatically for a homogeneous collection (a collection consisting of documents with a common layout and structure). There are also commercial products capable of the same. Unfortunately a number of federal organizations such as Defense Technical Information Center (DTIC), U.S. Government Printing Office (GPO), and National Aeronautics and Space Administration (NASA) manage heterogeneous collections consisting of documents with diverse layout and structure, where these programs do not work well. Furthermore, even with the best possible automated procedures, numerous sources of error exist, including some that cannot be controlled, such as scanned documents with text obscured by smudges, signatures, or stamps. A commercially viable process for metadata extraction must remain robust in the presence of these external sources of error as well as in the face of the uncertainty that accompanies any attempts to automate "intelligent" behavior. While extraction accuracy and completeness must be the primary goal of an extraction system, the ability to detect and report questionable results is equally important for a production quality system, since it promotes confidence in the system.

1.2 Problem Statements

Earlier research by the Old Dominion University Digital Library Group [4] explored the feasibility of using a template-based approach to extract metadata automatically from large heterogenous legacy collections. The success or failure of the template approach is directly tied to document classification which is the ability to match the document to the proper template correctly and consistently. While this dissertation examines the evolution and all the major components of an automated metadata extraction system, the primary focus will be on the problem of document classification to

support template-based automatic metadata extraction from large heterogenous legacy collections.

1.3 Approach

We have developed and demonstrated a novel system for extracting metadata. First, each document is subjected to extraction using each of the defined document class templates, which is a scripted description of how to associate blocks of text in the layout with metadata fields. For example, a template might state that the text set in the largest type font in the top-half of the first page is, in that layout, the document title. Then in the second part of the process, the best resulting extraction is chosen based on a scoring system which attempts to score the confidence in the quality of the individual fields. The extraction is then validated a second time to evaluate to quality of the extraction and possible flagging for human recognition.

We have tested our process and software against the DTIC collection, which contains more than one million documents and which adds tens of thousands of new documents each year. The documents are diverse, including scientific articles, slides from presentations, PhD theses, (entire) conference proceedings, promotional brochures, public laws, and acts of Congress. Contributions to DTIC come from a wide variety of organizations, each with their own in-house standards for layout and format, so, even among documents of similar kind, the layouts vary widely.

The template-based metadata extraction system is composed of commercial and public domain software in addition to components developed by our team and me. Documents are input into the system in the form of PDF files.

In this dissertation I will use the term "image PDF" to refer to a document consisting of a series of scanned page images and the term "text PDF" to refer to a document where the text is actually encoded as PDF instructions. Some documents are a mixed format and contain both images of pages and textual content. This commonly arises from an image PDF document being passed through an OCR program. These documents are treated as text PDF during input processing.

Some documents may contain a Report Document Page (RDP), one of several standardized forms that are inserted into the document when the document is added to the collection. For the DTIC collection, more than 50% of the documents contain RDPs offering more than 20 metadata fields.

Fig. 1 shows the complete process. The documents enter the input processing system where they are truncated and the pages are processed to extract textual content which is converted to a standardized XML format. Image PDF pages are processed by an Optical Character Recognition (OCR) program to extract the textual content. The first extraction step is to search for and recognize any RDP forms present. Any documents without recognized forms enter the non-form extraction process. The non-form extraction process generates a candidate extraction solution from the templates available. After extraction, the metadata from both form and non-form processing enter the output processor. The output processor is comprised of two components: a post-processing module and a validation module. The post-processing module handles cleanup and normalization of the metadata. The final automated step of the process is the validation module which, using an array of deterministic and statistical tests, determines the

acceptability of the extracted metadata. Any document that fails to meet the validation criteria is flagged for human review and correction.

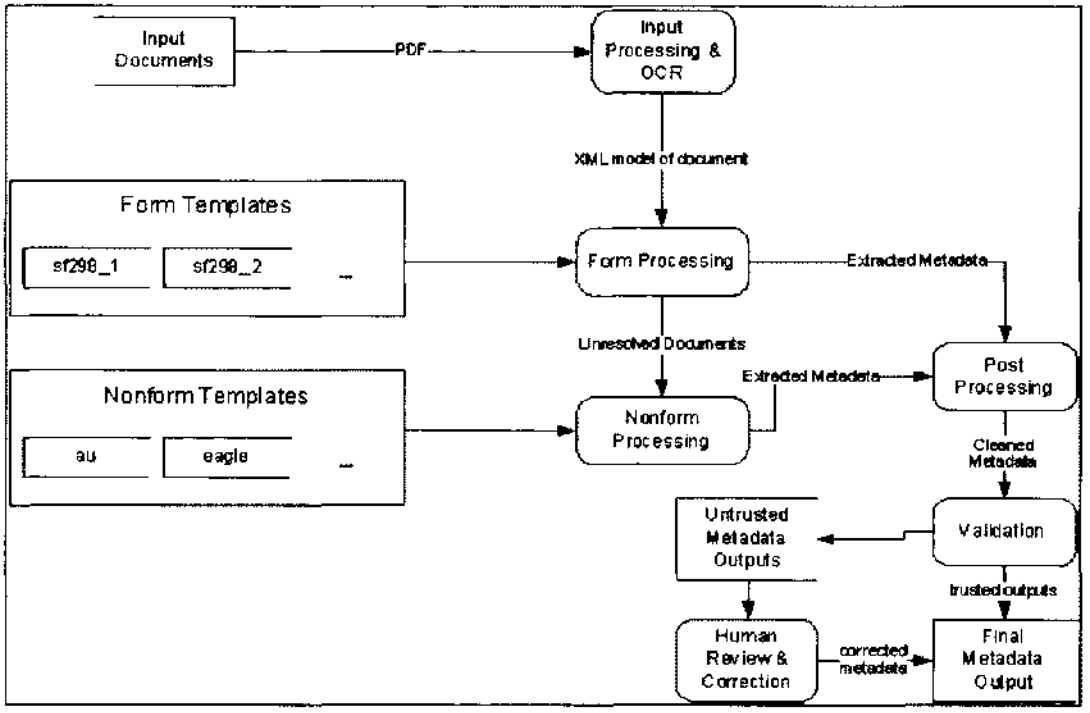


Fig. 1. Metadata Extraction Flow Diagram

1.4 Objectives

Our metadata extraction methodology is dependent upon the quality of the templates used and upon document classification, the ability to match the proper template to a document. Through our research we have developed a robust scripting language for the templates, to address a wide variety of layout complexities. The cost of the ability to handle these layout complexities is an increased complexity in writing and validating

templates. I helped to reduce this cost by developing a template creation program that provides the user with immediate feedback of about a proposed template.

In this thesis I will examine the evolution of the extraction system from a prototype thesis project to a fully capable application. I detail my contributions to the system in the areas of input processing, development of a document model and field normalization.

Document classification in our system is implemented as a module which applies every template available in the system to a document to find which templates will extract any data at all. These are called candidate templates. The candidate templates are evaluated by a validation module to select the best performing template. This method is called "post hoc" classification. The strength of post hoc classification is that the templates define the contextual meaning of blocks as well as the geometric relationship relative to other blocks as well as the page. In chapter 6, I provide analysis of the performance of the post hoc classification and it shows that when the post hoc classifier makes a decision it is correct 83% of the time. But just as significantly, the False Positive rate where it incorrectly assigns a classification to documents not represented in the template collection is only 8%. The main weakness of the system is that it is sensitive to templates with poorly defined or overly general rules. Researching methods to detect problem templates and reduce the sensitivity to changes in the template collection was a major focus area for my work.

The main area of my research has been into investigating alternative methods of document classification to replace or supplement post hoc classification. I experimented

with machine learning techniques as an additional input factor for the post hoc classification script or the final validation script.

In summary, I have the following objectives in this thesis:

- Detection of problem templates;
- Reducing sensitivity to template changes;
- Document classification to aid post hoc classification;
- Description of evolution of a large software system.

1.5 Organization of the Dissertation

The rest of the dissertation is organized as follows:

Chapter 2 – Background: In chapter 2, I will present the background and related works in the areas of document classification and metadata extraction

Chapter 3 – Architecture and Framework: In chapter 3, I will describe the major development phases of the extraction system as well as details of the architecture. I detail the major components looking at in turn: input processing, form processing, non-form processing, validation and finally field normalization or post-processing. The chapter ends with a description of a series of experiments I conducted to locate cover pages or pages of interest (POINT) in a document.

Chapter 4 – IDM: In chapter 4, I present an in depth look at the Independent Document Model (IDM), a representation of the document suitable for reasoning by the rule engine of the extraction system.

Chapter 5 – Non-form Templates: In chapter 5, I will present the specifications and rules of the non-form template language. Following the description of the language

and usage I detail techniques for reducing the sensitivity of the post hoc classification and minimizing the influence of overly general templates.

Chapter 6 – Experimentation: In chapter 6, I will describe the experiments I conducted into testing visual similarity measures for document classification and the effectiveness of various machine learning techniques based on these similarity measures.

Chapter 7 – Conclusions and future work: Finally, in chapter 7, I will summarize the contributions of my research as well as the issues I addressed. In this chapter, I will also offer suggestions for future work.

CHAPTER 2

BACKGROUND

This chapter summarizes prior research in the areas of extracting metadata from documents automatically and document classification techniques. I will review research into metadata extraction approaches in section 2.1 and document classification techniques in section 2.2.

2.1 Metadata Extraction Approaches

Existing automated metadata extraction approaches can be divided into two main categories: learning systems and rule-based systems.

Learning techniques including SVM [5],[6] and HMM [7] have been employed with promising results but to relatively homogeneous document sets. Experiments with these techniques [8] suggest a significant decline in effectiveness as the heterogeneity of the collection increases. A plausible explanation for this decline is that application of these learning systems to heterogeneous collections tends to dilute the internal probabilities that control their internal transitions. Evolution (changing characteristics of a document collection over time, such as acquiring a new source of documents in an unfamiliar format) poses a difficulty for these techniques as well, as they necessarily exhibit significant inertia resisting changes to the internally acquired "knowledge" until a significant number of examples of the new characteristics have been encountered.

Rule-based systems [9-11] use programmed instructions to specify how to extract the information from targeted documents. With sufficiently powerful rule languages, such techniques are, almost by definition, capable of extracting quality meta-data.

Heterogeneity, however, can result in complex rule sets whose creation and testing can be very time-consuming [11]. Analogies to typical software complexity metrics [12] suggest that complexity will grow much more than linearly in the number of rules, in which case even a well-trained team of rule-writers will be hard-pressed to cope with changes in an evolving heterogeneous collection and maintain a conflict-free rule set. Our own approach [4, 8] can be seen as a variant of the rule-based approach, but we finesse the complexity induced by heterogeneity and evolution by first classifying documents by layout, then providing a template for each layout, so that templates are independent of one another and individually simple.

The use of machine learning to generate "wrappers" for automated extraction of data from web pages is well-documented in the literature [13-22]. The head-left-right-tail algorithm [23] for inducing wrappers is a common strategy. Some systems provide user interfaces which allow the user to specify target elements by highlighting or selecting. The Data Extraction By Example (DEByE) system described in [24] is one example of a system for inducing wrappers of HTML pages by allowing a user to select and highlight. DEByE creates regular expression based building blocks for finding the appropriate data.

The WISDOM++ system [25-27] uses an inductive learning programming called Atre to classify documents and metadata extraction. WISDOM is trained using an interactive system that first segments and labels training documents. The user then corrects the segmentation and block labeling, with WISDOM recording the steps taken. Once an adequate number (5-15) of training documents are processed, the system attempts to create a set of rules using first order logic. A separate set of rules is generated for classification and extraction. The authors reported classification results better than

95% but only used three classes, so it is hard to estimate how well the system would scale.

Aumann et al. [28] created an information extraction system based on visual similarity of documents. They named their document model an "O-tree", which is a hierarchical model built using a bottom-up approach. The O-tree is created iteratively from primitive blocks which are joined and expanded into higher level objects based on proximity and visual similarity (font size, style, etc). During training, the objects in the O-tree are labeled by the user to indicate targeted fields of interest (i.e., metadata fields). Similarity for classification is done by exhaustively searching all training documents by attempting a block by block matching. Classification is based on the class with the highest average similarity with all the members of the training documents of that class. While they do report accuracy levels of 90%, the feature extraction is fairly coarse, concentrating primarily on titles, authors and other easily recognized features.

2.2 Document Classification

As we have noted, accurate document classification is one of the keys to solving the heterogeneity problem. Document classification, also known as "document layout analysis" or "document image classification", has been the target of numerous researchers in recent years. Mao [29] and Chen [30] noted 27 different systems in their surveys of the literature. The systems used a wide variety of features, models and algorithms in their classification methodologies. Additionally, there was a wide variety of the number of classes from very coarse classification schemes using few classes to fine grained classification using more than 500 classes. The need for some sort of document

classification is common to many metadata extraction systems. Reis [31] uses a tree edit distance algorithm to classify against sample templates for web page extraction.

2.2.1 XY Cut approaches

A common approach to document classification is using XY cuts where page segmentation is accomplished using alternating horizontal and vertical cuts on whitespace until a threshold is reached. Laven [32] enhances the basic XY cuts by including information from the surrounding features when deciding to make a cut. They tested a number of statistical methods for logical labeling of the segmented regions. Baldi [33] introduces a modified version of the basic XY segmentation called MXY where they include cuts on lines in addition to whitespace. For classification they used a set of tree-grammar rules to increase the coverage of the training set and used the K Nearest Neighbors (KNN) method measuring the tree-edit distance [34, 35]. Maranai [36-38] created a technique for encoding the features of the MXY tree into fixed length vectors required for many machine learning systems. Maranai also experimented with using tree grammar rules for document retrieval [39]. Cesarini used a neural network perceptron for classifying from MXY trees created from both OCR output [40] and image inputs [41]. Nattee [42] tested an on-line induction learning program called Winnow for document layout analysis and classification.

Appiani [43, 44] created a classification system using decision tree based machine learning and using MXY trees as the document model. They reported success rates in excess of 90% when classifying invoice forms into 9 separate classes. One factor in their success is that because MXY trees include lines as cut points, they are particularly suited to analyzing forms. When combined with the consistent cut patterns of an MXY

segmentation of a form, a decision tree can be expected to do a good job of classifying. They demonstrated excellent results in their experiments with approximately 500 documents and claim the system would scale appropriately in the area of form processing.

2.2.2 Visual layout approaches

Alternative methods using image analysis and semantic matching have also been explored in the literature. Hu [45-47] divided the page into a grid of M by N blocks. Each block is marked as text, whitespace or graphic depending on the content. They classified based on the edit distance between encodings. Van Beusekom [48] compared the similarity of documents based on measuring the Manhatten distance between all the blocks on the page. Le [49] used a combination of geometry-based and content-based zone features. These features are encoded in strings and classification is done using a rule-based learning system. Shin [50] and Eglin [51] measured visual similarity by segmenting the page into blocks and recording features about each block. Pages were clustered using K-means measurements.

2.2.3 Multiple classifiers

The use of multiple classifiers for classification is common in the areas of handwriting analysis and pattern recognition [52-56]. Duin [57] evaluated a number of fixed combination rules, Maximum, Median, Mean, Minium, Product and Majority(Voting) and found that selection of a combination scheme is dependent on the type of data and classifiers in use. They also found that the best results are obtained when the classifiers function on complementary features, an observation also noted by

other investigators [54, 58, 59]. Wenzel [60] applied a voting mechanism to combine two disparate classifiers for document classification.

CHAPTER 3

ARCHITECTURE AND FRAMEWORK

We have created a template-based metadata extraction system to solve the problem of automating the metadata extraction from heterogenous legacy collections. The templates contain a set of rules for locating desired metadata in a specific type of document layout.

This chapter examines the evolution of our extraction system in section 3.1, from a prototype thesis project to a complete system capable of handling extraction from multiple heterogenous document collections. It examines the major development phases of the system as well as the details of the architecture. After reviewing the evolution of the architecture, I detail in section 3.2, the major components looking at, in turn: input processing, form processing, non-form processing, validation and finally field normalization or post-processing. Following the architecture overview, I review the internal flow of information in the system, section 3.3, and the operation of the GUI, section 3.4. Finally in section 3.5, I detail a series of experiments I conducted to investigate how to locate pages of interest (POINT) that may contain metadata along with various extraction engine and field normalization improvements.

3.1 Evolution of architecture over time

I joined the project just after the release of the Version 1 prototype system which was based on the PhD work of Tang [4] and was capable of extracting from four types of RDP forms. Version 1 used OmniPage 14 as the input source to convert PDF documents into XML for processing. My earliest work with the project was to investigate methods

for detecting *Pages of Interest* also known as POINT pages --- pages which have the potential to contain metadata suitable for extraction, e. g. cover pages or title pages. As we will see, this work served to validate our design decision to limit processing to only the first and last five pages of each document. I also developed our initial field normalization module to clean up "Distribution Statement" fields in the RDP form using an Authority File and fuzzy string matching. The next phase of development involved expanding the system to handle nonform documents. We investigated several methods for classifying documents into template classes including expanding on my earlier POINT page algorithms. None of these methods proved reliable enough for use. Unable to find a suitable existing classification methodology, we decided to leverage work done by our metadata validation group to produce and innovative method of *post hoc classification* to select the best results from applying all the templates available in the system [61]. Post hoc classification will be discussed in detail later in this chapter. During these investigations we introduced the Independent Document Model (IDM), to be discussed in Chapter 4, which I developed in response to the introduction of a non-compatible XML schema in OmniPage 15. All of these advances were released as Version 3.0. The release was followed by feasibility study for a new collection of documents from the GPO EPA. One of the requirements of the study was to extract the metadata into a standardized MARC XML format [62] which required an expansion of the field normalization modules. Version 3.x saw four different releases as we expanded the commands available to template writers in the template language as well as the validation and field normalization modules. Version 3.4 was marked by a complete re-engineering of the entire system into a series of data transformers joined by connectors.

This allowed for robust testing of individual modules as well as flexibility to add or change features by connecting new transforms. I also introduced a separate template editor application that I created to assist the user in quickly designing and debugging new templates. Release 4.0 was the first release to offer native (text) PDF handling which allowed most documents to be processed without the need for OCR. This avoids a major source of error in typical document processing and also speeds up processing considerably. We will now examine the major components of the architecture in detail.

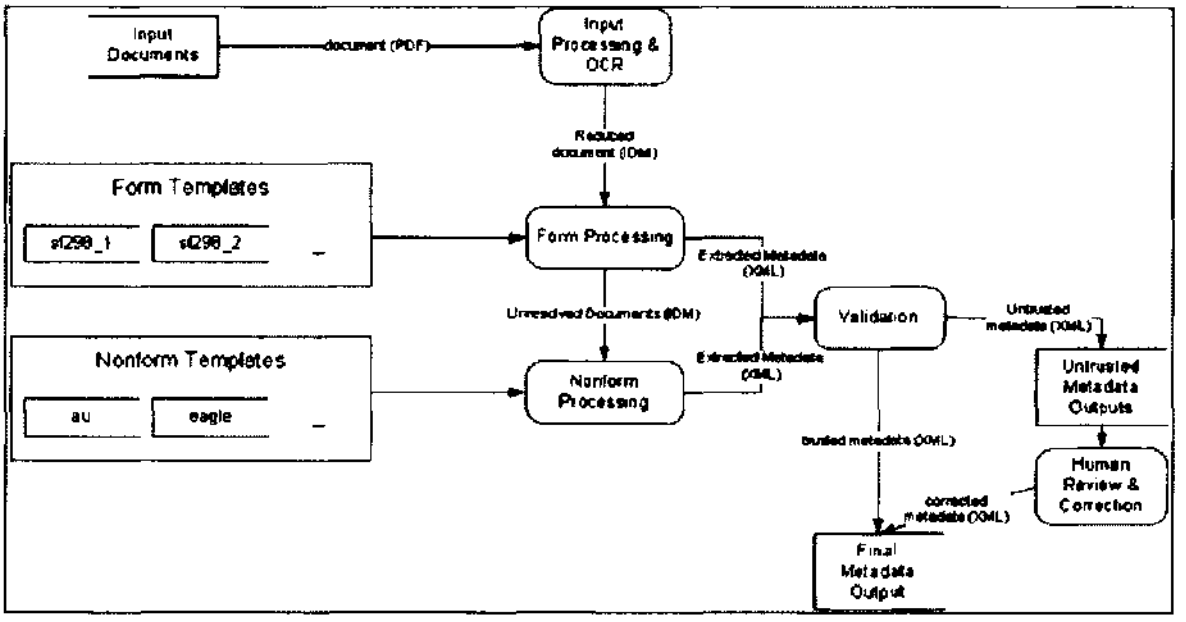


Fig. 2. Architecture of Extraction Process

3.2 Architecture overview

In the remainder of this chapter we will be referring to the final version of the architecture, as seen in Fig. 2, unless otherwise noted. In this section I will introduce the functions of the major components of the extraction system: input processing, form processing, non-form processing, validation, and field normalization.

3.2.1 Input processing

The entry point into the extraction system is the input processing module which is responsible for reading source documents and transforming them into segmented IDM format documents for further processing by the extraction engines. This section describes the components of the input processing module: page selection method; processing of OCR and XML production; direct processing of text PDF documents and final transformation into IDM format.

3.2.1.1 Page Selection

The source documents come into the system as PDF format files. These documents range from several pages to hundreds of pages in length. My POINT page selection work in section 3.5.1, has shown that the metadata we are interested in can typically be found in the first or last five pages of a document. While looking for metadata in the first five pages makes intuitive sense including the last five was based on the observation that many documents contained an RDP form appended to the end of the document. The system originally used the program ***pdftk*** [63] to split the first and last five pages out of the document and into a new PDF document. This truncated PDF document was fed into a commercial optical character recognition (OCR) for conversion into an XML format.

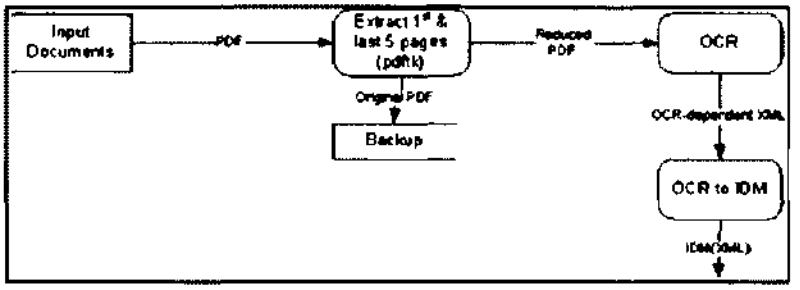


Fig. 3. Original Input Processing

3.2.1.2 OCR Processing XML Formats

As noted above, the first couple of versions were based on the Omnipage Pro 14 XML format. When Omnipage upgraded to version 15, the XML schema used for output was incompatible with the previous version. Due to the extremely tight coupling between the extraction code and the Omnipage Pro 14 schema, converting to a new version would have entailed a large expenditure of developer resources. The IDM schema detailed in chapter 4 is designed to eliminate the tight coupling with any specific program. The choice of moving to IDM was validated when DTIC subsequently moved to Luratech's OCR program and it took me less than 40 hours to create an XSL stylesheet to convert Luratech to IDM.

3.2.1.3 Text PDF usage

The Version 4.x introduced faster and more accurate processing for Text PDF documents, which changed the flow shown in Fig. 3. As shown in Fig. 4, the page extraction (via ***pdftk***) transform disappears. In its place is the Text PDF module which can extract the IDM directly from the file without need to scan into an OCR program.

The Text PDF module was built using ***Apache PDFBox*** Java PDF library [64]. The module also detects pages which may need OCR and sends just those pages to the OCR for recognition. The two outputs are merged into a single set of pages which are merged and segmented, organized into words, lines, paragraphs, regions, etc.

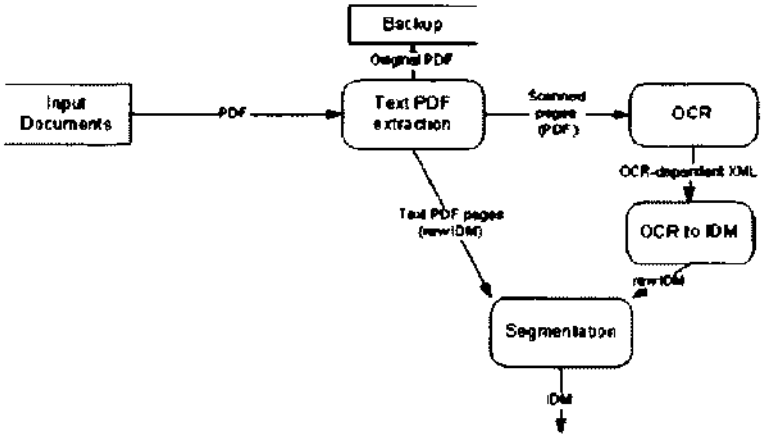


Fig. 4. Final Input Processing Dataflow

3.2.2 Form processing

Our experience with the DTIC collection has shown that roughly 50-60 percent of the documents contain an RDP form provided by the document submitter. The regular layout present in an RDP form makes it an attractive target for a template-based extraction process. In order to take advantage of the geometric relationships between fields in a form, we created an alternate version of our template language and extraction engine. The metadata fields are specified by a matching string and a set of rules indicating a positional relationship to one or more other fields as seen in Fig. 5. The number and layout of the fields for each different form constitute a unique signature for

that form class. If a template describing form A is applied to a document containing form B, the resultant metadata returned will contain few, if any, fields. We have leveraged this property in the design of our extraction process. Input processing finishes with IDM based documents exiting the input processor and entering the form processor. The processor is populated with a template developed for each version of RDP form found in the collection. We have found six different RDP forms within 9825 documents in the DTIC collection. The form processor runs the extraction process against the document using each of the templates and then selects the template, which returns the highest percentage of detected fields. If the form processor fails to match any template the document moves into the non-form extraction process described below. The extracted metadata is sent into the output processor.

```
<field num="19"><line>19. SECURITY CLASSIFICATION OF ABSTRACT
                    </line>
</field>
<field num="20"><line>20. LIMITATION OF ABSTRACT</line>
</field>
</fixed>
<extracted>
<metadata name="">
        <rule relation="belowof" field="1"/>
        <rule relation="leftof" field="2"/>
        <rule relation="aboveof" field="4|5"/>
</metadata>
<metadata name="ReportDate">
        <rule relation="belowof" field="2"/>
        <rule relation="rightof" field="1"/>
        <rule relation="leftof" field="3"/>
        <rule relation="aboveof" field="4|5"/>
</metadata>
```

Fig. 5. Form Based Template Fragment

3.2.3 Non-form processing

Any documents which do not contain a recognizable RDP form are passed on for further processing by the non-form engine, which is also a template-based extraction methodology. However in the case of the non-form engine the templates are designed based on layouts of individual document classes. Each template contains a set of rules designed to extract metadata from a single class of similar documents. Fig. 6 shows a template example. Each desired metadata item is described by a rule set designating the beginning and the end of the metadata. The first step in constructing a template is to identify a set of documents which share a structural or visual similarity. Once a class is

selected, the template author determines the set of rules for each metadata tag by identifying the appropriate function to select the beginning and the end of the tag. The non-form template language and structure are discussed in detail in Chapter 5.

The two main parts of the non-form process are the classification of a document by assigning it to the document layout class and the application of the template for the selected class to extract the metadata. We investigated multiple methods of implementing a document classification method as detailed below. None of the methods proved to be discriminating enough to be a reliable template selector. While I was investigating classification methods, our validation group completed initial development of the validation modules for evaluating the quality of extracted metadata. As noted previously in the evolution discussion, we replaced the a priori document classification methodology with post hoc classification.

```
<structdef pagenumber="1" templateID="amarac">
      <DescriptiveNote >
            <begin inclusive="current">
                  <stringmatch case="no" loc="beginwith">
                        Award Number:
                  </stringmatch>
            </begin>
            <end>onesection</end>
      </DescriptiveNote>
      <UnclassifiedTitle require="yes" min="1" max="1"
filter="(?i:TITLE):\s+(.*)">
            <begin inclusive="current">
                  <stringmatch case="no" loc="beginwith">TITLE:
                  </stringmatch>
            </begin>
            <end inclusive="before">
                  <stringmatch case="no" loc="beginwith">
                        Principal Investigator:
                  </stringmatch>
            </end>
      </UnclassifiedTitle>
```

Fig. 6. Non-form Template Fragment

3.2.4 Validation

The function of the validation module is to reliably determine the acceptability of extracted metadata. The metadata record extracted from a document will typically consist of many fields. Each field may be subject to several different validation tests. Which tests are most appropriate will vary from one field to another. [65]

The validation engine examines a set of metadata produced by earlier stages of the program and attempts to produce a score indicating how much confidence may be

placed in that result. These confidence values range from 0.0 (completely untrustworthy) to 1.0 (highly trusted).

The confidence values are used for two purposes. The final confidence for the output (the "*validation score*") is used to decide whether to call a human operator's attention to the output for review and possible correction. An intermediate "*classification score*" is computed earlier to help determine which template has done the best job for a given document and therefore which candidate set of extracted metadata to send along to the output module. This process is called "post hoc" classification. [61]

The validation engine provides a number of different tests that can be applied to different data fields [66]. The data for these tests is constructed by analyzing large samples of metadata for the collection for data values for each field, which is used both to construct the dictionary and to estimate the statistical properties of the rate of recurring entries that can be expected. For the DTIC collection we analyzed over 850,000 sets of historical metadata fields. The provided validation tests are:

- *DateFormat*: is the data formatted as a date.
- *Dictionary*: what percentage of the words in the data can be found in a dictionary? How does this number compare to a statistical mode of "typical" past data for this field?
    - By default, an English-language dictionary is used, but alternate dictionaries, e.g., other languages, or dictionaries of technical terms such as chemical names (used in conjunction with the GPO EPA collection) can be used.

    - For data fields that have specialized vocabularies, special-purpose dictionaries can be build. For example, a dictionary of common first and last names is available for use in checking PersonalAuthor fields.

- *(Not)Empty*: Is the data field empty?
- *Length*: Does the number of words in the data conform to a statistical mode of "typical" past data for this field?
- *Regexp*: Does the data match a particular pattern of characters (a "regular expression")?
- *Phrase (Dictionary):* what percentage of the K-word-long phrases in the data can be found in a dictionary of phrases? How does this number compare to a statistical mode of "typical" past data for this field?
- *Stub*: Returns a specific confidence value - mainly for testing purposes.

In addition, the validation engine provides a number of functions that can be used to modify confidence values, to combine confidence values from several tests for the same field, and to combine confidence values from several fields into an overall score for the entire metadata record.

- *Average*: compute the mean average of several confidence values. Can be weighted.
- *Max*: take the largest of several confidence values.
- *Min*: take the smallest of several confidence values.
- *Rescale*: multiply a confidence value by a scaling factor (larger or smaller) and/or add/subtract constant values from it.
- *Sum*: compute the sum of several confidence values. Can be weighted.

The choice of which tests to apply to which metadata fields and of how to combine the results of those various tests is controlled by a "validation script", which is executed by the validation engine. Separate scripts are used to compute the classification score and the final validation score, though these scripts are usually closely related. The script used for post hoc classification is different from the final validation script. The post hoc classification script uses a sum function at the outer level since we want to select the template which extracts the most metadata, while the final validation script uses a minimum function. The minimum function serves as a threshold below which the extracted metadata is suspect and requires human intervention.

Fig. 7 shows a fragment of the validation script for the DTIC collection. The fragment indicates that the overall classification score is the minimum (<val:min>) of the confidence scores for the individual fields.

The first field for which such a score is computed is UnclassifiedTitle (<val:field name="UnclassifiedTitle">). The score for that field is the smaller (another <val:min>) of two tests: dictionary and length. Each of these tests relies on our having previously collected statistics on titles from prior metadata records - specifically knowing what percentage of words in a typical title can be found in an English dictionary and what is the range of words found in typical titles with in the specific collection.

The next field shown is PersonalAuthor, and the score for an author is the minimum of the scores for three different tests. One is a test for length (number of words). The second is a fairly complicated pattern (regular expression) designed to check for a name in "last-name-first" format. The final test is a "phrase dictionary" check which

looks to see how many of the words (phrases of length one) in an extracted name can be found in a phrase dictionary built from names occurring in older DTIC metadata records.

```
<?xml version="1.0"?>
<val:validate collection="dtic"
 xmlns:val="http://www.cs.odu.edu/extract/validation">
 <val:min>
  <val:field name="UnclassifiedTitle">
    <val:min>
    <val:dictionary/>
    <val:length/>
 </val:min>
  </val:field>

  <val:field name="PersonalAuthor">
    <val:min>
      <val:length/>
      <val:regexp pattern="[-'A-Za-z]+(?: [-A-Za-z]+){0,2}(?:
(?:Jr|Sr|II|III|IV|V|VI)[.]?)?,(?: (?:[A-Z][.]?|[-A-Za-z]+)){1,2}"/>
      <val:max>
        <val:phrases length="1"/>
      </val:max>
    </val:min>
  </val:field>

  <val:field name="ReportDate">
       <val:regexp pattern="(?:(?:\d\d?)
)?(?:(?:JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DE
C) )?\d\d\d\d"/>
  </val:field>

  <val:field name="DatesCovered">
       <val:regexp pattern="(?:(?:\d\d?)
)?(?:(?:JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DE
C) )?\d\d\d\d (—|-|to) (?:(?:\d\d?)
)?(?:(?:JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DE
C) )?\d\d\d\d"/>
  </val:field>
```

Fig. 7. Validation Script Fragment for DTIC Collection

The tests that are currently used with the various DTIC fields are as follows:

- Abstract: length, dictionary (English)
- AbstractClassification: pattern
- AbstractLimitation: pattern
- ContractNumber: pattern
- DescriptiveNote: dictionary (English)
- DatesCovered: pattern
- Descriptor: pattern
- DescriptorClassification: pattern
- DistributionStatement: pattern
- PersonalAuthor: length, pattern, phrase dictionary (1)
- ReportClassification: pattern
- ReportDate: pattern
- ReportType: dictionary
- SupplementaryNotes: dictionary (English)
- TitleClassification: pattern
- UnclassifiedTitle: length, dictionary (English)

The reference models for the DTIC collection used to collect the names phrase dictionary and to determine average and standard deviation of field lengths and of the rate of word occurrence in dictionaries and phrase dictionaries are obtained by analysis of a samples of approximately 850,000 DTIC metadata records. The GPO Congressional and

EPA collections used separate reference models tailored specifically for those collections. These models are reconstructed a part of the software build process.

3.2.5 Field Normalization

The original post processing module was a module I designed to use an authority file and fuzzy string matching to correct OCR errors found in the Distribution Statement fields of the form processor. In the process of designing the post hoc classification system, we recognized the need to standardize the extracted inputs for the validation script to remove extraneous text or separate fields into multiple entries as needed. Our personal names post processor eventually evolved from a simple regular expression rule set into a named entity extractor that can separate multiple authors or corporate authors from personal names. Field normalization modules are injected into the data flow in multiple locations, after extraction prior to classification, after classification prior to validation, and after validation to format the metadata into the final desired format. One of the requirements of the GPO project was to deliver the final extractions encoded as MarcXML [62]. We added a normalization module to the flow which is invoked after the final validation of the selected metadata. Fig. 8 shows an example of the metadata extracted from a GPO-EPA document and Fig. 9 shows the output transformed into the MARCXML schema.

```
<metadata confidence="0.649" templateId="coverHeader-13b">
 <title_245a confidence="0.819">Air emissions from scrap tire
combustion</title_245a>
 <reportNumber_500 confidence="1.0">EPA-600/R-97-
115</reportNumber_500>
 <date_500 confidence="1.0">Oct. 1997</date_500>
 <personalAuthor_245c confidence="0.649">Joel I.
Reisman</personalAuthor_245c>
 <publisher_260b confidence="0.919">U.S. Environmental Protection
Agency, Office of Research and Development</publisher_260b>
 <placeOfPubl_260a confidence="0.902">Washington D.C.
20460</placeOfPubl_260a>
 <dateOfPubl_260c confidence="1.0">1997</dateOfPubl_260c>
 <note_500 confidence="0.0" warning="unvalidated">EPA Contract
No. 68-D30035</note_500>
</metadata>
```

Fig. 8. Example GPO-EPA Metadata Extract

```
<marc:collection xmlns:marc="http://www.loc.gov/MARC21/slim"
xsi:schemaLocation="http://www.loc.gov/MARC21/slim
http://www.loc.gov/standards/marcxml/schema/MARC21slim.xsd">
  <marc:record>
    <marc:leader>00441cmm a22001092a 4500</marc:leader>
    <marc:controlfield tag="003">DGPO</marc:controlfield>
    <marc:controlfield tag="005">200948213228.8</marc:controlfield>
    <marc:datafield tag="035" ind1=" " ind2=" ">
      <marc:subfield code="a">[vinotire_eng_9978369316002]</marc:subfield>
    </marc:datafield>
    <marc:datafield tag="245" ind1=" " ind2=" ">
      <marc:subfield code="a">Air emissions from scrap tire
combustion</marc:subfield>
      <marc:subfield code="h">[electronic resource] /</marc:subfield>
      <marc:subfield code="c">Joel I. Reisman.</marc:subfield>
    </marc:datafield>
    <marc:datafield tag="260" ind1=" " ind2=" ">
      <marc:subfield code="a">Washington D.C. 20460:</marc:subfield>
      <marc:subfield code="b">U.S. Environmental Protection Agency, Office of
Research and Development,</marc:subfield>
      <marc:subfield code="c">1997.</marc:subfield>
    </marc:datafield>
    <marc:datafield tag="300" ind1=" " ind2=" ">
      <marc:subfield code="b">Digital, PDF file.</marc:subfield>
    </marc:datafield>
    <marc:datafield tag="500" ind1=" " ind2=" ">
      <marc:subfield code="a">"Oct. 1997."</marc:subfield>
    </marc:datafield>
    <marc:datafield tag="500" ind1=" " ind2=" ">
      <marc:subfield code="a">"EPA Contract No. 68-D30035."</marc:subfield>
    </marc:datafield>
    <marc:datafield tag="500" ind1=" " ind2=" ">
      <marc:subfield code="a">"EPA-600/R-97-115."</marc:subfield>
    </marc:datafield>
  </marc:record>
</marc:collection>
```

Fig. 9. Resulting Output After MarcXML Transformation

We can see the effects of the field normalization process by following an example through the process. Fig. 10 shows the raw output from the application of the

"rand_arroyo_1" template. At this step in the process, the PersonalAuthor field includes 3 authors and some extraneous text. A special field named "_template" is also present. The leading underscore in the name is a flag for future field normalization.

```
<candidate templateId="rand_arroyo_1">
 <paper templateId="rand_arroyo_1">
  <metadata>
   <_template>This product is part of the RAND Corporation
occasional paper series. RAND</_template>
   <UnclassifiedTitle>Oversight of the
Liberian National Police</UnclassifiedTitle>
   <PersonalAuthor>David C . Gompert , Robert C . Davis ,
Brooke Stearns Lawson
Prepared for the Office of the Secretary of Defense</PersonalAuthor>
   <DistributionStatement>Approved for public release; distribution
unlimited</DistributionStatement>
   <ReportDate>2009</ReportDate>
  </metadata>
 </paper>
</candidate>
```

Fig. 10. Result of Initial Non-form Processing

After the initial extraction, the extracted metadata is processed to put the data into the correct format for executing the post hoc classification script. Fig. 11 contains the output of the classification script, the PersonalAuthor field has been normalized into three PersonalAuthor fields and the extraneous text is discarded. Note the confidence of the "_template" field is 100.0, this is a special boosting value to help separate this template from similar templates.

```
<selected>
 <paper templateId="rand_arroyo_1">
  <metadata>
   <_template confidence="100.0">This product is part of the RAND
Corporation occasional paper series. RAND</_template>
   <UnclassifiedTitle confidence="0.851">Oversight of the Liberian
National Police</UnclassifiedTitle>
   <PersonalAuthor confidence="0.952">Gompert, David
C</PersonalAuthor>
   <PersonalAuthor confidence="0.952">Davis, Robert
C</PersonalAuthor>
   <PersonalAuthor confidence="0.952">Lawson, Brooke
S</PersonalAuthor>
   <DistributionStatement confidence="1.0">Approved for public release;
distribution is unlimited.</DistributionStatement>
   <ReportDate confidence="1.0">2009</ReportDate>
  </metadata>
 </paper>
 </selected>
</classification>
```

Fig. 11. Sample Classification Script Output

The standardization step prepares the selected metadata for final validation scoring. Fig. 12 shows the output from the final validation step, we see that the "_template" field is no longer needed and the final confidence score has been applied. Note that the confidence score for the UnclassifiedTitle differs from the score used in the classification step. The reason for the difference is that the classification score is based on the average score of the dictionary and length tests, while the final validation score is the minimum of the dictionary and length tests.

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata confidence="0.846" templateId="rand_arroyo_1">
 <DistributionStatement confidence="1.0">Approved for public
release; distribution is unlimited.</DistributionStatement>
 <PersonalAuthor confidence="0.952">Gompert, David
C</PersonalAuthor>
 <PersonalAuthor confidence="0.952">Davis, Robert
C</PersonalAuthor>
 <PersonalAuthor confidence="0.952">Lawson, Brooke
S</PersonalAuthor>
 <ReportDate confidence="1.0">2009</ReportDate>
 <UnclassifiedTitle confidence="0.846">Oversight of the
Liberian National Police</UnclassifiedTitle>
</metadata>
```

Fig. 12. Result from Validation Script

The final step in processing is to convert the extracted metadata into the ".cit" file and a ".txt" file for ingestion into the DTIC system. The .cit file is intended to carry the actual metadata and the .txt file contains auxiliary information, including any warning messages generated by the validator regarding untrusted output values that should be reviewed by a human operator.

TABLE 1
Cit Code Mapping

Cit Code	Field Name
3	DescriptorClassification
6	UnclassifiedTitle
9	DescriptiveNote
10	PersonalAuthor
11	ReportDate
12	PaginationOrMediaCount
14	ReportNumber
15	ContractNumber
16	ProjectNumber
17	TaskNumber
18	MonitorAcronym
19	MonitorSeries
20	ReportClassification
21	SupplementaryNotes
22	DistributionStatement
25	Descriptor
25b	WorkUnitNumber
25d	ProgramElementNumber
27	Abstract
28	AbstractClassification
33	AbstractLimitation
34	ReportType

The mapping between .cit field numbers and the metadata field names used in the templates is shown in Table 1. Fig. 13 contains the final output of our example in .cit format.

Field-11, 2009
Field-06, Oversight of the
Liberian National Police
Field-10, Gompert, David C
Davis, Robert C
Lawson, Brooke S
Field-22, Approved for public release; distribution is unlimited.
Field-25c,

Fig. 13. Sample .cit File Text

3.3 Explanation of transform dataflows

Beginning with version 3.4, the system was re-engineered into a series of data transformers joined by connectors. Each module/function is implemented as a transformer object which takes as input a dataflow object and emits a dataflow object as output after performing the transformation procedure. This plug-in type of design allows us to add new capabilities or test experimental capabilities simply by inserting a new transform into the processing chain. The state of each transform is retrievable by the system for instrumentation purposes like tracking in the GUI, as seen in Fig. 14 below. The operation of the GUI is explained in the next section.

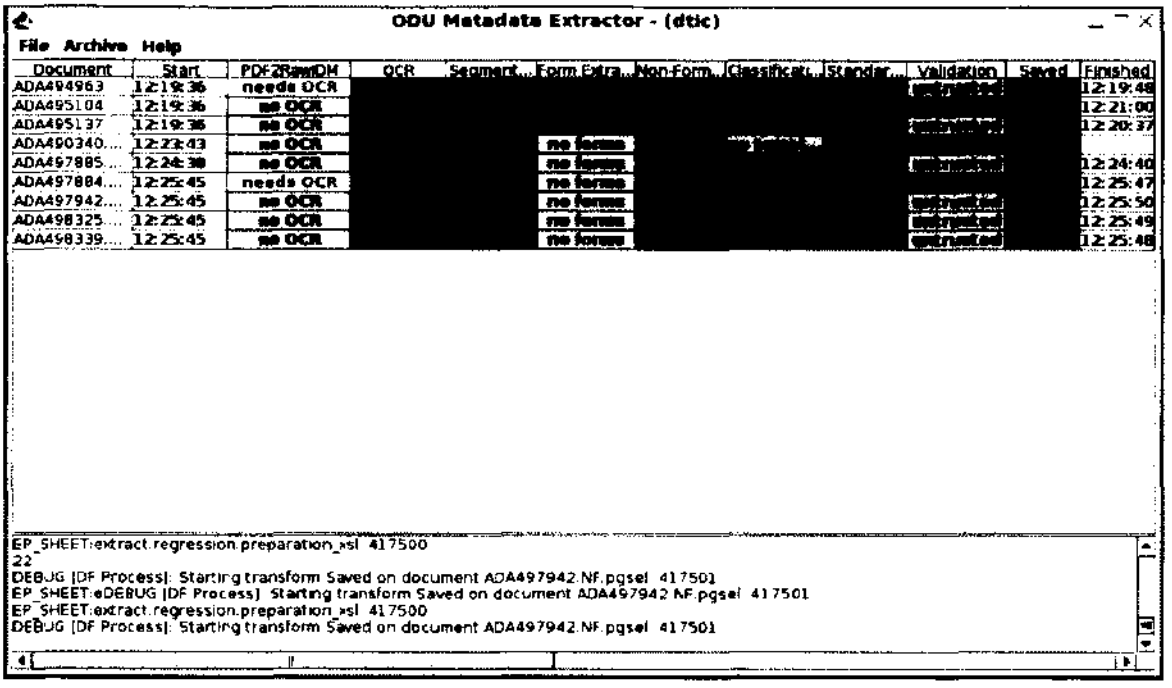


Fig. 14. Extraction GUI

3.4 Operation of software and implementation of GUI

The following description of the GUI operation is adapted from the Metadata Extraction Software, Operation manual [66]. As the software is running, the GUI allows the user to supply inputs to the program and to monitor progress being made on analyzing documents. The various columns shown in Fig. 14 for each document represent a specific step in the extraction of metadata from a document. As the document passes through the various steps, the columns reflect the status of each step (Fig. 14):

Green indicates a step that completed successfully.

Yellow is a step in progress.

White is a process that started but was found to be unsuited to the document.

Grey indicates a step that has not started or that is not required for this document.

Dark red indicates a step that failed. Such failures should be reported to the support staff.

Pale red indicates a step that failed but for a document specific reason.

The meaning of the pale red is context sensitive to the processing phase. A pale red "no templates" entry under the Classification column indicates that the software could not find a matching template for that document. This could be because the document itself is in such poor condition that nothing much could be done with it, or because it is a new type of document and no template has yet been written for its layout.

A pale red "untrusted" entry in the Validation column indicates that metadata was extracted, but that the software itself judges the output as being suspicious and is requesting human inspection and, if necessary, correction of the output. Clicking on this entry will open up a metadata editor that will allow direct correction of the metadata. The failure can be handled in accordance with established procedures. The most common reason for this is that metadata was extracted for a document, but the validation step decided that one or more output fields looked suspicious and is recommending that they be reviewed or corrected by a human.

3.5 Issues

In the following sections, I detail the search for POINT pages and then examine some of the incremental improvements we created to support the GPO-EPA collection and finish with a look at post-processing improvements.

3.5.1 POINT page experiments

One of our earliest research questions was to determine if it is possible to locate pages in a document that have a high probability of containing extractable metadata. One of my lines of inquiry was to investigate if I could identify a set of rules for identifying POINT pages based on various statistics about the composition of a page.

3.5.1.1 Experimental setup

I selected 108 documents from the DTIC collection and manually classified them into 12 classes based on the cover pages contained in each document. The documents were then processed into IDM format using the statistics option which produces both document and page level statistical information about the document. Fig. 15 shows an example of the IDM with statistics output. Table 2 contains the definitions of each of the statistics used. Table 3 shows example documents for each of the 12 classes along with statistic averages for the documents in each class.

```
<doc>
 <docInfo>
  <data id="avgWordPage " value="174.13"/>
  <data id=" stdevWordPage" value="81.32"/>
  <data id="docFontMode " value="Times New Roman"/>
  <data id="docFontModePct " value="0.87"/>
  <data id="docWords " value="87064"/>
  <data id="docAvgFontSize " value="932.761531746761"/>
  <data id="docStdevFontSize " value="128.17"/>
  <data id="docFontSizeMode" value="900"/>
 </docInfo>
 <page width="11962" height="15662" x-res="300" y-res="300" bpp="1" orientation="0"
   pgno="1">
  <pageInfo>
   <fontinfo>
    <fontsz name="1300" wds="17"/> <fontsz name="1100" wds="29"/>  <fontsz name="2100"
    wds="12"/>  <fontnm name="Arial" wds="58"/>
   </fontinfo>
   <parametrics>
    <data id="stdevFontSize" value="388.23"/>
    <data id="avgFontSize" value="1365.52"/>
    <data id="fontModePct" value="1.0"/>
    <data id="fontMode" value="Arial"/>
    <data id="fontSizeMode" value="1100"/>
    <data id="block" value="10"/>
    <data id="line" value="16"/>
    <data id="letter" value="300"/>
    <data id="word" value="58"/>
    <data id="digitline" value="7"/>
    <data id="digitwd" value="6"/>
    <data id="avgwordline" value="3.62"/>
    <data id="avglinesize" value="1337.5"/>
    <data id="capline" value="11"/>
    <data id="shortline5" value="12"/>
    <data id="shortline7" value="14"/>
    <data id="shortline9" value="16"/>
   </parametrics>
  </pageInfo>
  <region left="144" top="144" right="3398" bottom="4123">
   <image l="144" t="144" r="3398" b="4123"/>
  </region>
  <region left="3542" top="144" right="8726" bottom="4123">
   <vert-white-space t="144" b="970" pct="5.274" loc="top" unit="px"/>
   <para t="970" l="3802" r="8529" b="1891" li="0" ri="0" align="centered"
     line-spacing="336">
    <line l="4075" t="970" r="8304" b="1243" ff="Arial" fs="1300">
     <wd l="4075" t="970" r="5088" b="1181">Defense</wd>
     <wd l="5179" t="974" r="5971" b="1181">Threat</wd>
     <wd l="6077" t="974" r="7282" b="1186">Reductive</wd>
     <wd l="7382" t="979" r="8304" b="1243">Agency</wd>
    </line>
```

Fig. 15. IDM Fragment with Document Statistics

TABLE 2
Document Statistics Field Names

Document Level Metric	Definition
avgWordPage	Average number of words per page
docAvgFontSize	Average font size for document
docFontMode	Font used by the greatest number of words in the document
docFontModePct	Perecent of document using the Mode Font value
docFontSizeMode	Font size mode for the document
docStdevFontSize	Standard deviation for the font size for document
docWords	Total Words in document
stdevWordPage	Standard deviation for the number of words per page for document

Page Level Metric	Definition
avgFontSize	Average font size for words on page
avglinesize	average font size per line
avgwordline	Average number of word per line on page
block	Number of regions on page
capline	Number of lines in all Caps
digitend	Number of lines ending in digits
digitline	Number lines composed of only digits
fontMode	Font used by greatest number of words on page
fontModePct	Percent of words using Mode Font
fontSizeMode	Font size used by greatest number of words on page
letter	number of letters on page
line	Number of lines on page
shortline5	Number of Lines with less than 9 words
shortline7	Number of Lines with less than 9 words
shortline9	Number of Lines with less than 9 words
stdevFontSize	Standard deviation for font size on page
word	Number of words on page

TABLE 3
POINT Page Class Sample Documents and Statistics

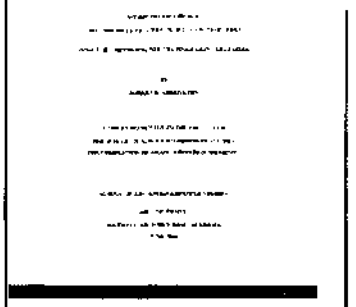
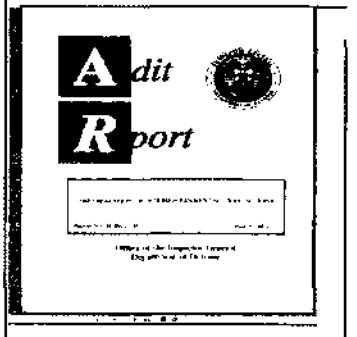
Cover Class	Document Level Metric	Metric	Averages for the cover page	Averages for the rest of pages	Sample Image
1	**docAvgFontSize:** 1104.48	**shortline9**	12.2	20.18	
	avgWordPage: 277.83	**word**	67.83	283.13	
	docWords: 24808.11	**avgwordline**	4.71	8.63	
	docFontModePct: 0.89	**avglinesize**	1348.47	1091.46	
	docStdevFontSize: 116.41	**fontSizeMode**	1240	1087.58	
	docFontSizeMode: 1105.71	**block**	2.77	4.19	
	stdevWordPage: 143.87	**letter**	305.89	1245.4	
		shortline7	10.94	17.98	
		avgFontSize	1318.36	1094.13	
		shortline5	7.86	15.89	
		digitline	2.69	16.93	
		capline	9.23	13.75	
		digitend	2.09	8.48	
		fontModePct	0.89	84.37	
		line	14.03	37.52	
2	**stdevWordPage:** 137.27	**shortline9**	10.29	22.72	
	docAvgFontSize: 1111.01	**word**	47.86	243.74	
	avgWordPage: 239.13	**avgwordline**	4.21	6.93	
	docWords: 27401.0	**avglinesize**	1758.15	1079.23	
	docFontModePct: 0.85	**fontSizeMode**	1657.14	1079.46	
	docStdevFontSize: 146.68	**block**	6.86	5.64	
	docFontSizeMode: 1128.57	**letter**	259.86	1102.8	
		shortline7	9.86	94.59	
		avgFontSize	1667.92	1082.56	
		shortline5	7.43	18.89	
		digitline	2.71	17.69	
		capline	7.29	15.5	
		digitend	2.14	10.95	
		fontModePct	0.97	0.91	
		line	11.43	21.02	

TABLE 3 Continued

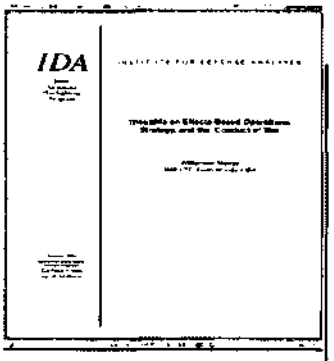
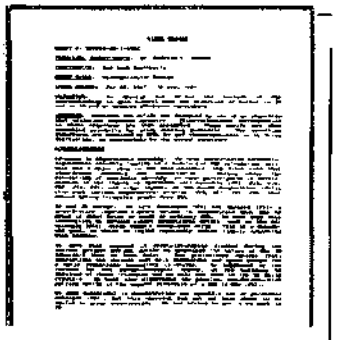
Cover Class	Document Level Metric	Metric	Averages for the cover page	Averages for the rest of pages	Sample Image
3	**stdevWordPage:** 166.51	**shortline9**	12.5	17.84	
	docAvgFontSize: 1097.89	**word**	55.62	235.6	
	avgWordPage: 230.4	**avgwordline**	4.24	7.72	
	docWords: 16693.62	**avglinesize**	1348.6	1039.34	
	docFontModePct: 0.8	**fontSizeMode**	1262.5	1054.33	
	docStdevFontSize : 140.59	**block**	4	3.3	
	docFontSizeMode : 1137.5	**letter**	229.12	1060.96	
		shortline7	12.12	16.21	
		avgFontSize	1333.48	1053.35	
		shortline5	10.5	13.93	
		digitline	2.5	11.12	
		capline	9.62	11.34	
		digitend	2	6.36	
		fontModePct	0.83	80.82	
		line	13.12	31.81	
4	**stdevWordPage:** 127.71	**shortline9**	14	18.73	
	docAvgFontSize: 1171.14	**word**	278.6	365.81	
	avgWordPage: 355.81	**avgwordline**	8.51	9.88	
	docWords: 5235.6	**avglinesize**	1252.41	1115.4	
	docFontModePct: 0.86	**fontSizeMode**	1200	1135.66	
	docStdevFontSize : 155.84	**block**	2.2	2.8	
	docFontSizeMode : 1220.0	**letter**	1331	1714.85	
		avgFontSize	1216.25	1126.01	
		shortline5	9.4	13.19	
		digitline	9.8	13.97	
		capline	9.8	11.51	
		digitend	4.2	3.01	
		fontModePct	0.96	87.62	
		line	32.2	41.24	
		shortline7	11.6	15.75	

TABLE 3 Continued

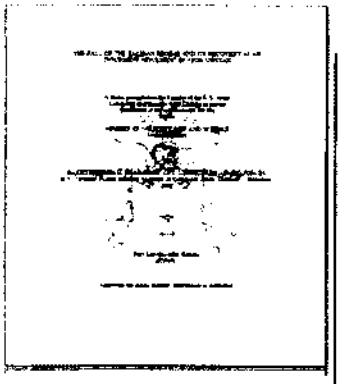
Cover Class	Document Level Metric	Metric	Averages for the cover page	Averages for the rest of pages	Sample Image
5	**stdevWordPage:** 137.52	**shortline9**	3	42.86	
	docAvgFontSize: 1151.36	**word**	9	500.3	
	avgWordPage: 296.6	**avgwordline**	3	7.12	
	docWords: 26694.0	**avglinesize**	1300	1026.56	
	docFontModePct: 0.88	**fontSizeMode**	1300	1040.5	
	docStdevFontSize : 142.37	**block**	3	10.85	
	docFontSizeMode : 1200.0	**letter**	58	2212.13	
		avgFontSize	1288.89	1029.82	
		shortline5	3	22.34	
		digitline	0	13.3	
		capline	2	14.98	
		digitend	0	5.58	
		fontModePct	1	91.59	
		line	3	71.6	
		shortline7	3	30.54	
6	**stdevWordPage:** 91.71	**shortline9**	4	6.86	
	docAvgFontSize: 1186.68	**word**	24	226.36	
	avgWordPage: 223.73	**avgwordline**	4.8	9.77	
	docWords: 17227.0	**avglinesize**	1160	1176.72	
	docFontModePct: 1.0	**fontSizeMode**	1200	1171.05	
	docStdevFontSize : 53.58	**block**	4	2.16	
	docFontSizeMode : 1200.0	**letter**	114	1057.41	
		avgFontSize	1191.67	1183.22	
		shortline5	3	4.53	
		digitline	0	5.61	
		capline	4	3.42	
		digitend	0	2.04	
		fontModePct	1	22.49	
		line	5	22.58	
		shortline7	3	5.82	

TABLE 3 Continued

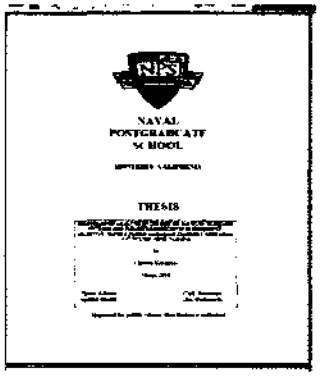
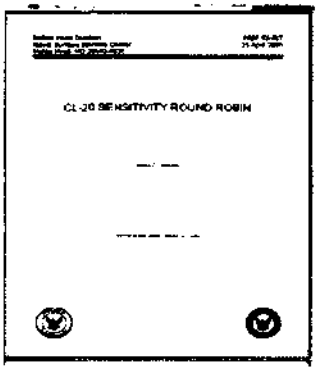
Cover Class	Document Level Metric	Metric	Averages for the cover page	Averages for the rest of pages	Sample Image
7	**stdevWordPage:** 131.23	**shortline9**	14	13.33	
	docAvgFontSize: 1152.25	**word**	50.08	248.93	
	avgWordPage: 246.88	**avgwordline**	3.48	8.82	
	docWords: 31148.75	**avglinesize**	1399.91	1127.94	
	docFontModePct: 0.96	**fontSizeMode**	1283.33	1155.5	
	docStdevFontSize : 100.94	**block**	5.08	3.88	
	docFontSizeMode : 1183.33	**letter**	273.75	1092.21	
		avgFontSize	1326.75	1140.97	
		shortline5	10.75	9.85	
		digitline	3	10.61	
		capline	12.25	7.51	
		digitend	2.5	4.08	
		fontModePct	0.92	59.85	
		line	14.33	28.9	
		shortline7	12.75	11.83	
8	**stdevWordPage:** 137.93	**shortline9**	12.75	14.64	
	docAvgFontSize: 1092.23	**word**	44.75	154.16	
	avgWordPage: 150.24	**avgwordline**	3.69	6.43	
	docWords: 5202.75	**avglinesize**	1737.1	1031.34	
	docFontModePct: 0.91	**fontSizeMode**	1500	1052.55	
	docStdevFontSize : 181.74	**block**	6.5	5.14	
	docFontSizeMode : 1125.0	**letter**	247.25	675.18	
		avgFontSize	1682.15	1041.42	
		shortline5	9	11.05	
		digitline	3.25	9.73	
		capline	8.5	8.41	
		digitend	2.25	4.54	
		fontModePct	0.78	98.23	
		line	12.75	22.98	
		shortline7	11.5	12.81	

TABLE 3 Continued

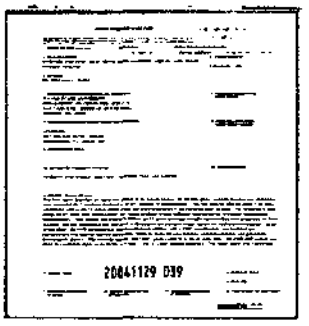
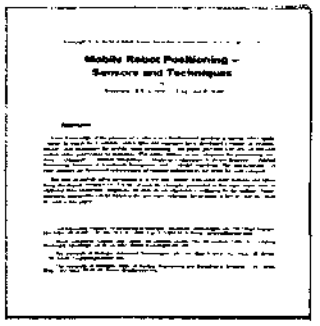
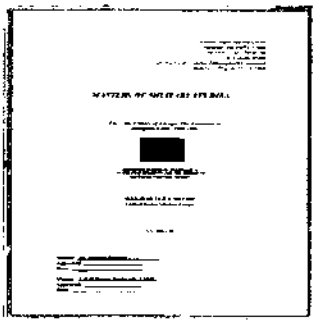
Cover Class	Document Level Metric	Metric	Averages for the cover page	Averages for the rest of pages	Sample Image
9	**stdevWordPage:** 137.79	**shortline9**	34.6	19.07	
	docAvgFontSize: 1151.97	**word**	333.8	230.02	
	avgWordPage: 237.37	**avgwordline**	7.07	7.59	
	docWords: 13987.0	**avglinesize**	1068.66	1192.9	
	docFontModePct: 0.75	**fontSizeMode**	1040	1207.01	
	docStdevFontSize : 257.13	**block**	5	5.59	
	docFontSizeMode : 1220.0	**letter**	1574	941.46	
		avgFontSize	1045.76	1213.3	
		shortline5	27	15.69	
		digitline	20.4	16.89	
		capline	28.4	12.55	
		digitend	6.2	6.41	
		fontModePct	0.7	0.89	
		line	50.4	17.63	
		shortline7	31.6	138.84	
10	**stdevWordPage:** 200.63	**shortline9**	17	18.91	
	docAvgFontSize: 1005.98	**word**	523.67	446.99	
	avgWordPage: 549.3	**avgwordline**	10.47	9.09	
	docWords: 4113.33	**avglinesize**	1086.37	1261.9	
	docFontModePct: 0.97	**fontSizeMode**	1033.33	1316.29	
	docStdevFontSize : 135.2	**block**	7.33	4.51	
	docFontSizeMode : 1033.33	**letter**	2339.67	2011.32	
		avgFontSize	1051.3	1255.52	
		shortline5	8	12.18	
		digitline	5	11.91	
		capline	7	9.35	
		digitend	2.33	2.53	
		fontModePct	0.98	0.98	
		line	51.67	15.44	
		shortline7	11.67	64.53	

TABLE 3 Continued

Cover Class	Document Level Metric	Metric	Averages for the cover page	Averages for the rest of pages	Sample Image
11	**stdevWordPage:** 218.17	**shortline9**	32.6	30.38	
	docAvgFontSize: 1051.09	**word**	570	391.66	
	avgWordPage: 435.74	**avgwordline**	9.3	8.69	
	docWords: 4766.5	**avglinesize**	1079.34	1040.58	
	docFontModePct: 0.98	**fontSizeMode**	1080	1047.98	
	docStdevFontSize : 92.25	**block**	3.1	3.84	
	docFontSizeMode : 1080.0	**letter**	2504.1	1736.08	
		avgFontSize	1072.18	1044.17	
		shortline5	9.1	17.91	
		digitline	12.7	17.31	
		capline	7.8	13.09	
		digitend	1.3	5.49	
		fontModePct	0.95	56.75	
		line	64.8	51.5	
		shortline7	17.1	23.24	
12	**stdevWordPage:** 128.66	**shortline9**	16	16.81	
	docAvgFontSize: 1151.68	**word**	62.5	278.35	
	avgWordPage: 272.09	**avgwordline**	3.81	9.18	
	docWords: 21812.75	**avglinesize**	1331.06	1140.5	
	docFontModePct: 0.95	**fontSizeMode**	1225	1160.79	
	docStdevFontSize : 124.24	**block**	4.75	3.62	
	docFontSizeMode : 1175.0	**letter**	316.5	1217.23	
		avgFontSize	1314.65	1152.16	
		shortline5	12.5	14.21	
		digitline	4.75	15.23	
		capline	11.75	11.88	
		digitend	3.75	7.64	
		fontModePct	0.87	60.64	
		line	16.5	33.99	
		shortline7	14.5	15.73	

3.5.1.2 Process

I developed a program to evaluate and optimize a set of rules utilizing the various statistics to identify a POINT page class. I began by specifying a set of simple rules and then the program would find the optimum value for the individual test which would select the desired cover page but would also minimize false positive matches in other pages. The initial set of rules included:

- Blocks GE 3 : Number of Blocks on page >= 3
- Lines GE 3 : Number of lines >= 3
- Lines LT 30 : Number of line < 30
- Letter LT 700 : Number of letters < 700
- Words LT 200 : Number of words < 200
- DigitLine LT 9 : Number of digit only lines < 9
- Digit End LT 4 : Number of lines ending in digits < 4
- Avg Word per Line LT 10 : Average number of words per line < 10
- Avg Line Size GT 1000 : Average font size of lines > 1000
- Title GT Half : Number of lines in title case > half the total lines
- Short line 7 LT Half: Number of lines shorter than 7 words < half the total lines

I ran the optimizer program against the documents in each class to determine the values for the test. Table 4 shows a compilation of these results. Each row represents one of the rules listed above. Note that it also shows the inverse test, i.e. (Lines >= N) and the inverse (Lines < N). The columns are grouped first by a class identifier and then under each class it shows the optimized value for the test, the total number of manually selected POINT pages matched by the rule value and finally the number of other pages

matched by the same rule. Under each class, the best performing version of the rule is highlighted in yellow. Notice that the class results can be grouped into 2 larger groups with fairly disjoint rule sets as noted by the color coding on the class labels. These groups correlate with the sparse type of traditional coverage and the dense scientific paper type.

TABLE 4
POINT Page Selection Rule Results

			c1			c2			c3						c5			c6			c7			c8									c12		
			Value	Mch	Fls	Value	Mch	Fls	Value	Mch	Fls	Value	Mch	Fls	Value	Mch	Fls	Value	Mch	Fls	Value	Mch	Fls	Value	Mch	Fls	Value	Mch	Fls	Value	Mch	Fls	Value	Mch	Fls
fr1	fontModePct	LT	1	43	30.4	1	43	30.4	1	43	30.4	1	43	30.4	1	43	30.4	1	43	30.4	1	43	30.4	1	43	30.4	0.9	29	11.7	1	43	30.4	1	43	30	1	43	30.4
fr2	stdevFontSize	GT	0	95	69.4	79.7	72	17.6	171.9	52	5.2	0	95	69.4	33.3	83	39.7	40.8	82	34.6	163	56	5.6	406.5	15	0.6	24.8	86	46.6	96.7	68	12.8	0	95	69	147.1	58	6.6
fr3	avgFontSize	GT	790.9	95	68.5	1128	72	32.7	1108	75	34.4	1056	78	41.4	1289	41	1.3	1192	68	21.8	1157	71	30.1	1235	57	1.8	882.4	91	66	942.5	89	62.9	956.6	88	63	1122	75	33
r2	line	GE	4	94	67.3	4	94	67.3	9	85	64.3	19	32	56.2	3	95	68	5	91	66.7	10	84	63.7	6	89	65.9	23	24	52	31	20	38	26	21	45	14	58	60.6
r3	line	LT	18	60	14.3	16	52	12.7	16	52	12.7	40	81	48.8	3	0	3.4	5	4	4.7	17	58	13.5	14	37	10.8	50	87	60.9	47	86	58.1	85	90	67	17	58	13.5
r4	letter	LT	422	62	12.4	410	61	12.2	268	41	9.1	1707	81	55.9	58	1	3.8	114	5	5	344	54	10.8	218	29	7.9	1944	85	61.4	2401	88	66.5	3084	92	70	330	52	10.4
r5	word	LT	98	68	12.2	76	51	10.1	63	45	8.8	384	83	55.4	9	1	3.2	24	5	4.7	65	49	9	39	20	6.3	415	85	58.1	521	88	65.4	722	93	70	61	44	8.5
r6	digitline	LT	5	62	23.9	5	62	23.9	4	54	19.4	13	87	49.6	0	0	0	0	0	0	7	76	32.3	5	62	23.9	34	94	66.3	6	71	28.5	13	87	50	6	71	28.5
r7	digitend	LT	5	81	56	4	72	53.4	3	60	48.1	6	87	57.6	0	0	0	0	0	0	6	87	57.6	4	72	53.4	10	93	62.2	3	60	48.1	2	44	37	5	81	56
r8	avgwordline	LT	6	70	21.3	5.1	59	16.1	5.4	65	17.5	10.2	88	46.1	3	7	7.1	4.8	53	14.5	4.5	49	13.3	3.9	28	10.7	8.9	85	39.2	11.3	92	54.2	11	90	52	4.2	36	11.9
r9	avglinesize	GT	792.8	95	68.3	1141	75	29.7	1143	74	29.5	1189	68	20	1300	41	1.3	1160	70	27.2	1320	39	1.2	1255	52	1.6	883.6	91	65.3	1031	83	40.7	938.9	90	62	1147.6	72	29
fr1a	fontModePct	GT	0.4	94	70	0.7	80	65.8	0.5	90	69.3	0.8	75	63.7	1	0	0	1	0	0	0.5	90	69.3	0.4	94	70	0.3	95	70.1	0.9	66	59.7	0.6	87	68	0.5	90	69.3
fr2a	stdevFontSize	LT	312	72	69.4	307.7	71	69.3	460.9	84	70.2	200	50	66.7	33.3	12	31	40.8	13	36.1	297	68	69.1	536.5	87	70.3	204.8	52	66.9	109.7	30	60.7	124	33	62	272.1	64	68.8
fr3a	avgFontSize	LT	1554	81	70.6	1667	86	70.7	1365	72	70.4	1300	57	70.2	1289	54	70.1	1192	27	49.6	1415	75	70.5	1931	90	71	1104.4	19	36.1	1023	16	28.3	1200.6	34	67	1426	77	70.5
r2a	line	LE	17	60	14.3	15	52	12.7	15	52	12.7	39	81	48.8	3	1	4.1	5	6	5.5	16	58	13.5	13	37	10.8	49	87	60.9	46	86	58.1	84	90	67	16	58	13.5
r3a	line	GT	4	91	66.7	4	91	66.7	9	84	63.7	19	30	55.3	3	94	67.3	5	89	65.9	10	79	63.1	6	87	65.5	23	21	49.9	31	19	36.3	26	20	44	14	52	59.5
r4a	letter	GT	40	94	68.2	101	90	66.7	132	85	66	516	27	57	58	93	67.6	114	89	66.4	181	76	64.4	146	84	65.7	688	22	53.2	1589	15	18.8	1327	19	28	262	55	62.5
r5a	word	GT	8	94	68.2	18	93	67.1	36	78	65.3	99	26	59	9	93	68.1	24	89	66.7	32	83	65.8	30	85	66.1	130	22	56.3	342	16	20.4	281	19	32	54	55	63.5
r6a	digitline	GT	0	92	67.1	1	68	61.6	4	33	47.5	16	5	15.5	0	92	67.1	0	92	67.1	8	14	31.5	5	24	42.9	1	68	61.6	3	41	52	1	68	62	2	48	56.5
r7a	digitend	GT	0	84	63.5	4	14	15.4	0	84	63.5	0	84	63.5	0	84	63.5	0	84	63.5	6	4	12.5	4	14	15.4	0	84	63.5	3	23	18	0	84	64	2	35	23.3
r8a	avgwordline	GT	2	94	67.4	3.1	83	63.2	3.4	79	62.5	5.2	33	54.8	3	86	63.5	4.8	41	56.9	2.6	93	65.8	2.9	89	64.6	4.3	51	59.2	9	9	31.5	8	17	37	3.6	73	61.7
r9a	avglinesize	LT	1576	81	70.6	1701	85	70.8	1481	76	70.5	1301	54	70.1	1300	50	70	1160	24	43.9	1434	73	70.5	1941	89	71	1192.6	18	38.3	1036	15	31	1207.9	37	68	1440.6	74	70.5

3.5.1.3 Analysis

TABLE 4 shows the results for matching against the following set of rules:

- r1-Blocks GE 3
- r2-Lines GE 3
- r3-LInes LT 30
- r4-Letter LT 700
- r5-Words LT 200
- r6-DigitLine LT 9
- r7-Digit End LT 4
- r8-Avg Word per Line LT 10
- r9-Avg Line Size GT 20

For each rule, we try to match the manually selected cover page. We list the total number of rules matched and also list the other pages in the document matching every rule. (False Positives) Table 5 lists all the rules. The large number of false positives was an issue since the purpose of searching for POINT Pages was to limit the number of pages processed by the system. In this implementation, we are testing each page in the document, some of which may have several hundred pages. After evaluating the computational expense of running a POINT page detection algorithm, we realized we were not seeing any significant improvement over the default selection of processing the first and last five pages of a document. We confirmed this assessment by sampling another 100 documents and found only 1 in which a POINT page was found outside the

first five pages. Note that the primary reason for including the last five pages is that RDP pages are frequently found appended to the end of a document.

TABLE 5
Optimization Program Rules Matching Results

Class	File	Manual Cover Page	Total Rules Matched	Pages Matching All Rules	r1	r2	r3	r4	r5	r6	r7	r8	r9
0	ADA426321	0	0		0	4	1	1	1	3	3	1	3
0	ADA429458	0	0		12	14	1	1	1	4	8	8	12
0	ADA425007	0	0		1	4	0	0	0	3	3	1	3
0	ADA428639	0	0	7	7	8	5	1	2	2	5	6	8
1	ADA427827	1	9	1	2	54	53	2	3	35	53	9	53
1	ADA426518	2	0		1	1	1	1	1	1	0	1	1
1	ADA391330	1	8	71 78	14	100	83	16	25	62	60	27	100
1	ADA391330	4	8	71 78	14	100	83	16	25	62	60	27	100
1	ADA397295	1	8	75	14	98	66	15	23	63	75	44	94
1	ADA424326	3	9	3 18 25	38	47	10	8	10	15	25	25	42
1	ADA428589	1	8		25	32	4	3	3	2	14	25	6
1	ADA399127	1	8	55 58	10	66	46	12	18	37	49	42	65
1	ADA428319	3	9	1 3 8 10 102 155 206	77	239	48	13	23	##	188	53	19
1	ADA428140	3	9	1 3 4 6 10 12 13 23 31 39 41 49 53 61 69 71 79 80 84 92 94 97 104 106	102	106	11	20	27	##	105	102	105
1	ADA423163	3	8	1 33 44 46 48	25	44	21	24	25	20	31	45	27
1	ADA398805	1	8		2	59	47	4	4	41	52	14	58
1	ADA427476	2	9	1 2 57 72	44	89	34	12	18	44	64	41	73
1	ADA426518	1	8		0	0	0	0	0	0	0	0	0
1	ADA428190	1	8		7	21	7	4	10	7	16	17	19
1	ADA423705	1	7		2	24	9	7	9	12	22	13	1
1	ADA423304	1	7		1	23	9	7	8	17	22	13	0
1	ADA407719	1	8	22 37 44 52 55	26	71	65	19	28	55	63	18	71
1	ADA428463	1	8		1	5	2	1	2	2	3	3	4
1	ADA428069	3	7	1 38	38	47	27	22	23	36	42	30	37
1	ADA391748	5	8	32 46 58 70	51	76	71	13	16	39	69	20	75

TABLE 5 Continued

Class	File	Manual Cover Page	Total Rules Matched	Pages Matching All Rules	r1	r2	r3	r4	r5	r6	r7	r8	r9
1	ADA425874	1	8	27 38 55 69	64	109	64	9	15	80	99	21	109
1	ADA394846	3	8	138	113	207	8	23	13	29	68	135	85
1	ADA424493	2	7	20	28	39	21	9	11	18	28	18	26
1	ADA428260	3	8	1 11 21 61 64 65 67 69 70 71 72 74 76 77 79 80 83 84 88 91 92 93 101 104 105 106 107 111 112 113 114 115 116 122 131 132 133 142 143 146 190 253	217	281	125	104	##	##	171	261	161
1	ADA429436	1	7	32 40	15	41	31	14	17	11	36	20	42
1	ADA424345	1	7		0	26	9	8	8	24	29	10	0
1	ADA427618	4	8		6	43	35	13	15	22	36	20	41
1	ADA394974	1	8	45	2	47	41	8	10	38	41	14	46
1	ADA428731	4	9	4 17 18 21 31 40 43 86 91 111 130 141 156 166 171 177 194 201 211 220 225 235 237 247 252 264 273 278 290 294 303 308 312 316 319 321 329 332 335 352 356 371 425 462	272	455	137	122	##	##	432	479	30
1	ADA428315	1	8	10 37 48 66 73 80	29	78	70	18	27	63	74	31	76
1	ADA421205	2	8	17 21 23 31 34	21	39	22	21	22	18	18	27	34
1	ADA427430	2	4	10 23 38 48 106	56	121	34	25	29	46	88	65	33
1	ADA429162	1	7	9	7	8	3	1	3	2	3	7	5
1	ADA424083	1	7		2	21	10	7	10	10	17	15	0
1	ADA397297	1	8		1	42	35	11	11	20	40	14	42

TABLE 5 Continued

Class	File	Manual Cover Page	Total Rules Matched	Pages Matching All Rules	r1	r2	r3	r4	r5	r6	r7	r8	r9
10	ADA422391	1	6		5	6	0	0	0	1	4	5	0
10	ADA428024	0	0	10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 30 31 32 33 34 35 36 37 38 39 40 41 42 43	41	42	37	35	35	36	40	36	42
10	ADA428261	1	5		9	10	5	1	2	6	9	2	5
10	ADA423886	1	5		2	4	0	0	0	3	3	1	3
11	ADA421226	1	4		7	11	5	6	4	1	6	2	6
11	ADA425692	1	6		1	25	1	1	1	5	22	4	19
11	ADA425313	1	5		3	19	3	1	1	9	15	3	13
11	ADA412482	1	3		3	4	0	0	1	2	3	4	1
11	ADA410242	1	6		1	2	0	0	1	1	2	2	1
11	ADA422844	1	5		14	23	1	0	0	4	18	15	20
11	ADA410854	1	5		3	4	0	0	1	1	1	4	2
11	ADA412480	1	4		1	3	0	0	1	1	1	3	1
11	ADA409431	1	3		2	5	2	2	3	4	5	1	2
11	ADA412126	1	5		2	4	1	1	2	2	4	3	1
12	ADA393943	1	8	4	4	18	5	2	4	12	15	7	17
12	ADA427800	1	8	19 32 63 66 68 72 85 91 106 112 117 120 128 129 131 137 140 148 159 163 172 173 175 178 179 184 185 187 188 190 192 204 209 219 221 226 234 235 240	175	246	144	106	##	##	173	178	206
12	ADA403968	1	9	1 40	42	72	48	6	7	34	59	13	68
12	ADA421488	2	9	2 17 19	19	27	10	7	11	3	14	18	24
2	ADA422180	1	8	3 28 33 102 107 151 171 188 205 213 218 219 223	226	222	84	29	38	83	162	103	127

TABLE 5 Continued

Class	File	Manual Cover Page	Total Rules Matched	Pages Matching All Rules	r1	r2	r3	r4	r5	r6	r7	r8	r9
2	ADA389024	1	9	1 12 14 18	16	18	4	4	4	5	15	14	17
2	ADA388593	1	9	1 22 30 31 33	7	30	19	15	19	26	30	28	29
2	ADA422180	3	9	3 28 33 102 107 151 171 188 205 213 218 219 223	226	222	84	29	38	83	161	103	127
2	ADA422074	1	8	46 57 76	55	80	23	39	50	16	25	73	79
2	ADA391608	1	9	1 6 8 12 15 18 19	11	7	19	18	19	19	20	20	14
2	ADA393335	1	9	1 10 26	25	30	8	2	4	10	24	18	25
3	ADA425272	1	9	1 9	5	8	8	8	8	7	9	8	8
3	ADA428069	1	9	1 38	37	47	27	22	23	36	42	29	37
3	ADA423163	1	9	1 33 44 46 48	24	44	21	24	25	20	31	45	27
3	ADA428179	3	9	1 2 3 28 41	16	51	27	20	26	36	41	33	47
3	ADA396993	1	9	1 7 10 14 17 21 24 26	22	27	15	12	15	26	26	27	23
3	ADA426757	1	8		5	20	9	4	4	17	18	5	19
3	ADA428260	1	9	1 11 21 61 64 65 67 69 70 71 72 74 76 77 79 80 83 84 88 91 92 93 101 104 105 106 107 111 112 113 114 115 116 122 131 132 133 142 143 146 190 253	216	281	125	104	##	##	171	261	161
3	ADA428179	1	9	1 2 3 28 41	16	51	27	20	26	36	41	33	47
4	ADA428626	2	4		0	6	1	1	1	1	4	6	5
4	ADA427171	1	8		3	22	1	1	1	3	16	8	21
4	ADA424327	2	3	9	6	12	2	1	2	2	8	9	11
4	ADA428109	0	0		1	2	1	1	1	2	1	1	0
4	ADA425011	2	5		1	15	12	0	1	8	13	4	14
4	ADA429294	1	4	2	11	13	4	2	2	9	10	7	12
4	ADA413522	0	0	2	2	7	2	2	2	4	6	3	6

TABLE 5 Continued

Class	File	Manual Cover Page	Total Rules Matched	Pages Matching All Rules	r1	r2	r3	r4	r5	r6	r7	r8	r9
5	ADA428731	0	0	4 17 18 21 31 40 43 86 91 111 130 141 156 166 171 177 194 201 211 220 225 235 237 247 252 264 273 278 290 294 303 308 312 316 319 321 329 332 335 352 356 371 425 462	273	456	138	123	##	##	433	480	31
5	ADA428140	0	0	1 3 4 6 10 12 13 23 31 39 41 49 53 61 69 71 79 80 84 92 94 97 104 106	103	107	12	21	28	##	106	103	106
5	ADA427476	1	9	1 2 57 72	44	89	34	12	18	44	64	41	73
5	ADA425444	0	0	2 10 34 42 46 50 67 79	98	100	5	4	8	52	86	95	73
5	ADA423430	0	0	1 36	40	43	1	1	1	15	30	42	10
6	ADA428904	1	9	1 24 62	9	76	69	16	20	62	67	27	72
7	ADA422366	1	9	1 36 41 59 68 76 82 86 89 90 105 110 111 122 125 127 130 142 143 147 148 150 154 155 156	115	153	110	106	84	71	128	85	130
7	ADA426291	1	8	19 24	17	49	19	12	21	36	42	38	49
7	ADA401603	1	9	1 6	24	110	37	19	28	65	88	58	110
7	ADA401622	1	9	1 52 61 99 105	34	107	61	25	28	##	109	39	98
7	ADA401720	1	9	1 6 24 31 32 47 50 51	20	45	43	32	36	36	51	21	47
7	ADA416272	1	8	16 21 31 46 50 54 58 59 114 123 138 165 214	87	211	102	75	92	##	217	157	67

TABLE 5 Continued

Class	File	Manual Cover Page	Total Rules Matched	Pages Matching All Rules	r1	r2	r3	r4	r5	r6	r7	r8	r9
7	ADA429872	1	9	1 28 38 49 50 83 96 100 104 106 121 126 143 145 147 152 153 154 155 157 158	90	147	95	59	84	##	134	121	140
7	ADA429849	1	9	1 26 76	85	106	28	17	20	28	91	46	113
7	ADA403150	1	9	1	21	70	37	23	26	56	72	38	64
7	ADA415118	1	9	1 5 35 45 46 50 63 66 77 85 92 96 110	39	110	104	28	40	98	103	38	104
7	ADA421001	1	8	39 46	34	45	19	14	6	8	23	23	40
8	ADA421566	1	9	1 20	6	19	11	7	9	17	17	14	18
8	ADA415096	1	9	1 8 10 20 24 26 31 34 35 38 42 46 49 52 56 60 64 68	66	68	56	56	56	39	52	65	7
8	ADA425810	1	8	4 8 11 13 15 16 17	25	27	15	15	16	18	22	22	22
8	ADA395819	1	9	1 9 13 17 21 25	26	8	23	22	22	26	27	29	9
9	ADA428043	1	5		3	5	2	1	1	2	2	4	2
9	ADA426275	2	4		10	15	5	5	5	1	7	10	9
9	ADA427800	2	7	19 32 63 66 68 72 85 91 106 112 117 120 128 129 131 137 140 148 159 163 172 173 175 178 179 184 185 187 188 190 192 204 209 219 221 226 234 235 240	176	246	144	106	##	##	172	178	207
9	ADA426484	1	4	6 9 11 12 13 14 15 16 18 22 23 24	19	23	19	18	20	17	19	19	22
9	ADA428076	1	3	2	12	34	27	7	7	3	23	12	31

3.5.2 Incremental Engine Improvements

While much of the research conducted during this project was focused on the DTIC collection, we made significant progress on the system with experiments with two collections of documents from the Government Printing Office (GPO). The first collection was a collection of 1000 sample documents from the Environmental Protection Agency (EPA). The second collection consisted of 921 GPO Congressional documents which were used in a feasibility study to estimate the number of templates needed to cover the collection [67]. While the largest class (epa-ord) representing EPA study reports covered more than 25% of the collection, the other documents were very diverse. The 108 templates developed cover 633/994 (64%) of EPA collection documents. Another 78 documents were covered by classes with fewer than 5 member documents, and the remaining 283 documents were singleton classes. We did not note any significant use of forms in the GPO collection, so our efforts concentrated on non-form template development. In addition to developing new templates and validation scripts, we also implemented improvements to the extraction engine and additional post-processing modules [68]:

• Add ability to process non-form documents on multiple-pages. The original DTIC engine was oriented towards extraction of data from a single page. The analysis of available metadata, particularly for the epa-ord class, noted several instances of desired fields that are available on pages separate from the cover page where the bulk of the metadata can be located (e.g., Fig. 16 GPO Sample file metadata on multiple pages).

• Add text filter to metadata field descriptions to facilitate both removal of extraneous strings (e.g., "Title: ") and splitting of metadata fields occurring together on

one line. Several instances were found of metadata values preceded by "boilerplate" strings that are useful as markers for locating metadata, but should not be included in the extracted values themselves. Other instances were found where two metadata fields occurred within a single line. Because the extraction engine processes and extracts entire lines at a time, some finer control is necessary in these cases.

• Allow templates to describe placement of "marker" fields that do not actually generate metadata but can be used to indicate relative locations of actual metadata. It is intended that the development of templates describing metadata placement should be a task that could be performed by staff with technical expertise well short of full programming skills. This change would simplify the development of templates for a wide variety of experience levels.

• Allow templates to mark metadata fields as mandatory or optional. The epa-ord class contains many minor variants, particularly in the pages following the cover page. In the current template language, it would be necessary to create a separate template and document class for each variant. This change would allow a number of these variants to be handled within a single template. It should also increase the robustness of the process of recognizing which template is most suited to a given document.

• Allow some selection from geometric relationships. " rightof(meta)" - Locates 1st line to the right of the begining of the tag. Being to the right means that the midline of the testing line is between the top or bottom of the 1st block of the previously extracted tag. "endrightof(meta)" - Locates 1st line not to the right of the tag.

• Add enhanced vertical space selection rules. "verticalSpace(s)" searches for a line that is followed by whitespace of at least s*h, where h is the height of that line. (The

height of the line is estimated as 1.15 * the bounding box height. "verticalSplit(k,n)" that splits the current page into n blocks by locating the $(n-1)$ largest inter-line spaces not at the very top or bottom of the page. The operator then selects the line beginning the k'th block. k starts at 0, so verticalSplit(0,n) always selects the first non-empty line on the page. As a special case, verticalSplit(n,n) selects the last non-empty line on the page.

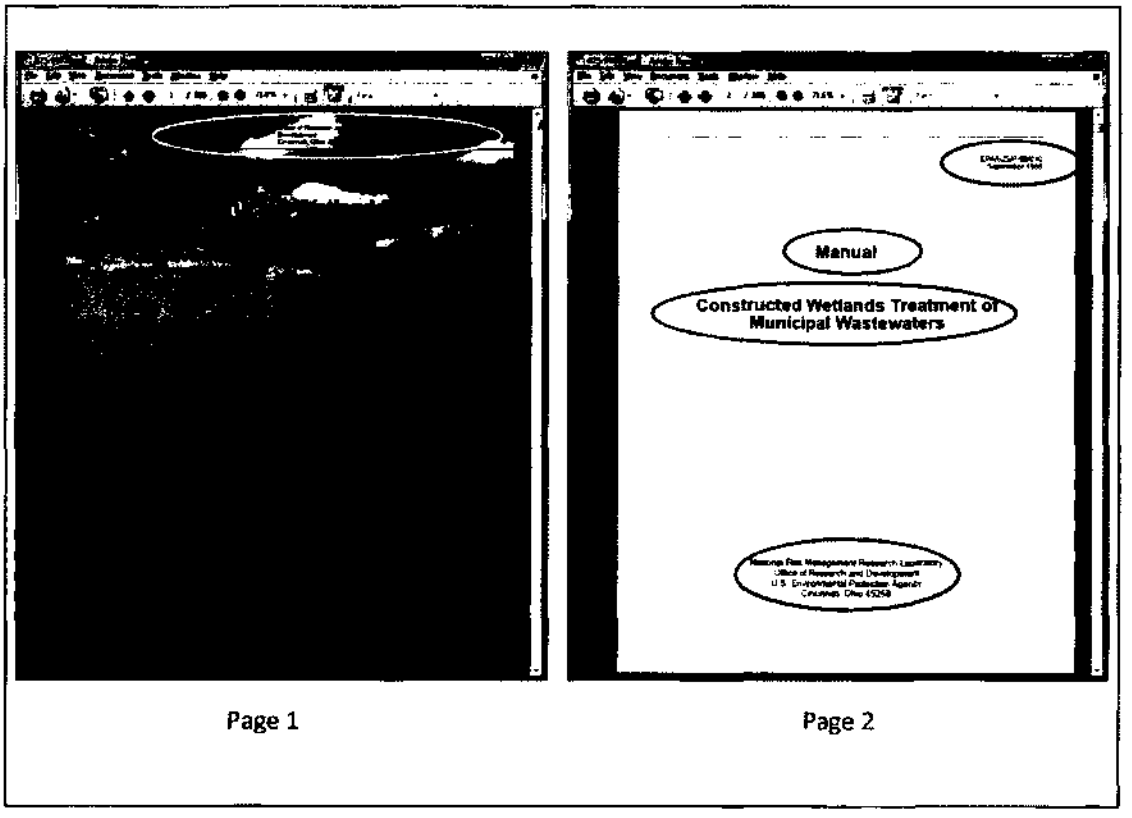


Fig. 16. GPO Sample File Metadata on Multiple Pages

3.5.3 Field Normalization Improvements

In addition to incremental changes to the extraction engine we had to create several post-processing functions specific to the GPO-EPA collection. We found a relatively large number of documents which contained a group of metadata contained in three columns at the top of the first page. Extraction of these metadata items was complicated by the OCR ignoring the column and treating each of the lines as a single line. As seen in Fig. 17, the data from the lines marked 1, 2 and 3 is broken into separate metadata items. This post-processor takes a single template rule tag as input and will output up to 9 metadata fields, depending on those present.

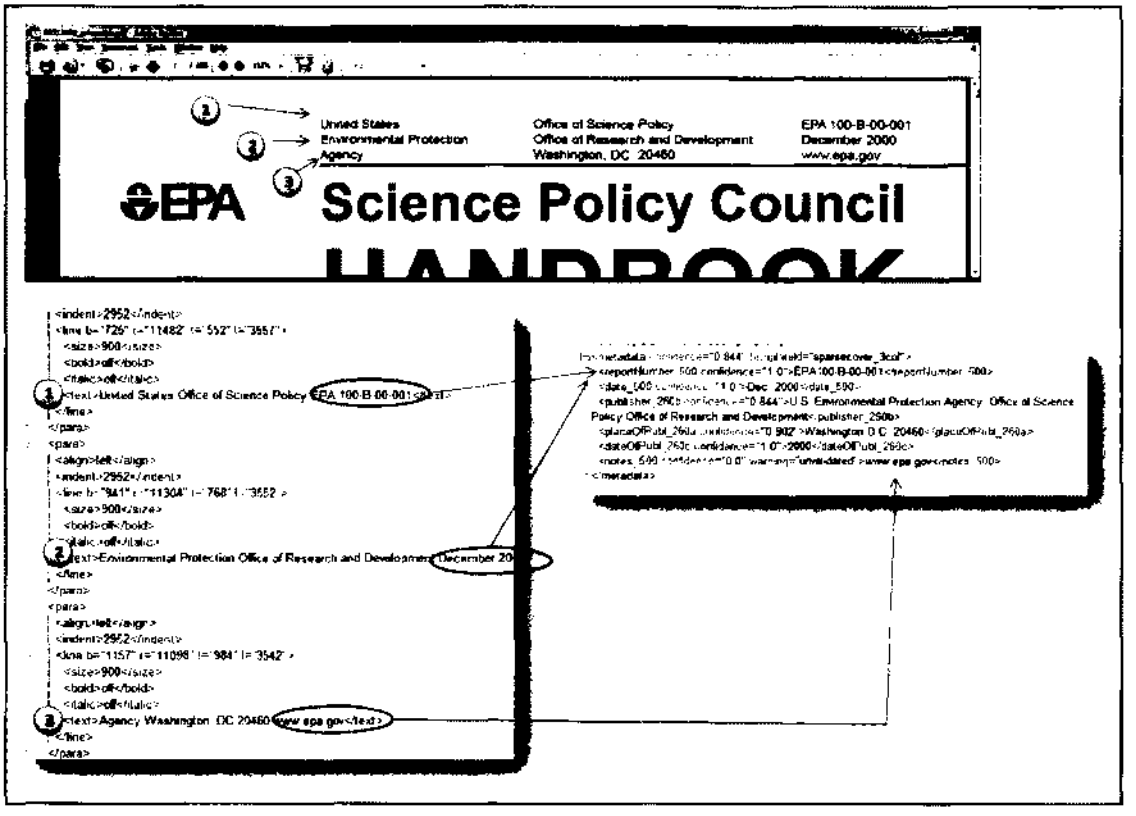


Fig. 17. GPO 3 Column Header

One of the requirements of the GPO project is to deliver the final extractions encoded as MarcXML [7]. We added a post-validation module to the flow which is invoked after the final validation of the selected metadata.

CHAPTER 4

IDM

As noted during the exploration of the evolution of the system, the earliest versions of the software was tightly coupled to the OmniPage 14 XML schema. When OmniPage upgraded their software and dramatically change their XML schema, we were faced with the need to conduct a major refactoring of the system to support the new schema. This chapter discusses how we solved this problem.

In this Chapter I review in depth the development and usage of the Independent Document Model (IDM). We begin in section 4.1 by looking at the motivations and the need for developing the IDM schema followed by a review of the structural elements in section 4.2. We finish up in section 4.3 with a look at version 2 of the schema and the accompanying development of the TextPDF module which allowed us to bypass the OCR recognition phase and convert text PDF documents directly into the IDM format for processing. The final version of the IDM schema can be found in Appendix A.

4.1 Motivations for evolution

Many commercial OCR programs provide a method for saving the recognized text in some form of XML format. Unfortunately, the output schemas of the formats often vary, even among updated versions of the same program. We experienced this variation using different versions of the OmniPage OCR programs.

The IDM schema is a platform independent schema used to support extraction of metadata from XML documents. I created the IDM to allow us to control the specification upon which we build our processing engines, which effectively isolates the

processes from input format variations and introduces standards for describing page segmentation. The extraction and classification engines originally operated on XML documents that were tightly coupled to the OmniPage 14 OCR XML schema. The release of OmniPage 15 introduced a dramatically different XML schema for its documents. Due to the tight coupling with the OmniPage 14 schema, transitioning to the 15 schema would have required an inordinate amount of recoding of the extraction engine. Fig. 18 shows a snippet from a document formatted using OmniPage 14 schema and Fig. 19 shows the same document formatted in OmniPage 15. Notice that the attribute names for font size and font face have been changed from "fs" to "fontsize" and "ff" to "fontface" in addition to other structural changes. I proposed that a better approach would be to create a separate module to convert XML documents produced by OCR documents into a standard model for internal processing, the IDM.

```
<page width="12240" height="15840" x-res="300" y-res="300" bpp="1" orientation="0" skew="0"
filename="C:\Documents and Settings\navi\Desktop\sample file collection\error1+2\20030093540.pdf" language="0">
 <region reg-type="horizontal">
  <rc l="10" t="14913" r="12240" b="15384"/>
  <paragraph para-type="text" align="left" left-indent="1368" right-indent="0" start-indent="0" line-spacing="312">
   <ln baseline="15231" ff="Times New Roman" fs="1400">
    <wd l="1445" t="15005" r="2851" b="15312">September</wd>
    <wd l="2938" t="15014" r="3518" b="15235">2003</wd>
   </ln>
  </paragraph>
 </region>
 <region reg-type="horizontal">
  <rc l="10" t="858" r="12240" b="1450"/>
  <paragraph para-type="text" align="left" left-indent="1368" right-indent="0" start-indent="0" line-spacing="312">
   <ln baseline="1171" ff="Times New Roman" fs="1400">
    <wd l="1445" t="950" r="4853" b="1210">NASA/TM—2003-212395</wd>
   </ln>
  </paragraph>
  <paragraph para-type="text" align="left" left-indent="9216" right-indent="0" start-indent="0" line-spacing="312">
   <ln baseline="1166" ff="Times New Roman" fs="1400">
    <wd l="9331" t="950" r="11626" b="1176">AIAA–2003–0903</wd>
   </ln>
  </paragraph>
 </region>
 <region reg-type="graphic">
  <rc l="10" t="1450" r="12240" b="2645"/>
 </region>
 <region reg-type="horizontal">
  <rc l="10" t="2645" r="12240" b="4426"/>
  <paragraph para-type="text" align="left" left-indent="1296" right-indent="0" start-indent="0" line-spacing="576">
   <ln baseline="3312" ff="Times New Roman" fs="2300">
    <wd l="1445" t="2957" r="4195" b="3317">Mixed-Phase</wd>
    <wd l="4334" t="2971" r="5386" b="3446">Icing</wd>
    <wd l="5515" t="2957" r="7829" b="3317">Simulation</wd>
    <wd l="7963" t="2957" r="8746" b="3317">and</wd>
    <wd l="8890" t="2971" r="10406" b="3446">Testing</wd>
   </ln>
  </paragraph>
  <paragraph para-type="text" align="left" left-indent="1296" right-indent="0" start-indent="0" line-spacing="528">
   <ln baseline="3888" ff="Times New Roman" fs="2300">
    <wd l="1454" t="3581" r="1834" b="3893">at</wd>
    <wd l="1968" t="3533" r="2606" b="3893">the</wd>
    <wd l="2750" t="3542" r="3586" b="3893">Cox</wd>
    <wd l="3720" t="3547" r="4771" b="4022">Icing</wd>
    <wd l="4891" t="3533" r="6043" b="3893">Wind</wd>
    <wd l="6182" t="3533" r="7642" b="3893">Tunnel</wd>
   </ln>
  </paragraph>
 </region>
```

Fig. 18. Sample OmniPage 14 Formatted Page Snippet

```
<page ocr-vers="OmniPage Pro 15">
<description><source file="C:\Documents and Settings\nratkal\Desktop\monupdate\dump\nasa2form\20030093540.pdf"
dpix="300" dpiy="300" sizex="2550" sizey="3300"/><theoreticalPage size="Letter" marginLeft="1446"
marginTop="951" marginRight="503" marginBottom="523" offsetX="-472" width="12240" height="15840"/>
<language>en</language></description>
<styleTable>
   <style styleID="paraStyle_1_3_31" alignment="left" lsp="exactly" lspExact="300" underlined="none"
fontSize="1100" fontFace="Bookman Old Style" fontFamily="roman" fontPitch="variable">
</style> ...
</styleTable>
<body>
<dd l="10" t="950" r="12240" b="1214">
<para l="1353" t="950" r="11695" b="1214" alignment="left" li="1296" lsp="exactly" lspExact="262" language="en"
styleRef="paraStyle_1_3_32"><tabs position="1353"/>
  <ln l="1445" t="950" r="11621" b="1210" underlined="none" subsuperscript="none" fontSize="1400"
fontFace="Bookman Old Style" fontFamily="roman" fontPitch="variable" spacing="0" scale="1000">
  <wd l="1445" t="950" r="2338" b="1181">NASA</wd><space/>
  <wd l="1445" t="950" r="2501" b="1210">/</wd><space/>
  <wd l="1445" t="950" r="9331" b="1210">TM-2003-212395</wd><tab position="4853"/>
  <wd l="1445" t="950" r="11621" b="1210">AIAA-2003-0903</wd>
</ln></para></dd>
<section l="1308" t="1214" r="10548" b="8839"><column l="1308" t="1214" r="10548" b="8839">
<picture l="1512" t="1445" r="2899" b="2650" alignment="left" li="144" ri="7649" spaceBefore="216"
spaceAfter="144"></picture>
<para l="1308" t="2813" r="10526" b="3411" alignment="centered" spaceBefore="72" lsp="exactly" lspExact="596"
language="en" styleRef="paraStyle_1_3_32">
<ln l="1445" t="2957" r="10406" b="3446" underlined="none" subsuperscript="none" fontSize="2100"
fontFace="Bookman Old Style" fontFamily="roman" fontPitch="variable" spacing="-15" scale="1102">
<wd l="1445" t="2957" r="4334" b="3317">Mixed-Phase</wd><space/>
<wd l="1445" t="2957" r="5515" b="3446">Icing</wd><space/>
<wd l="1445" t="2957" r="7963" b="3446">Simulation</wd><space/>
<wd l="1445" t="2957" r="8890" b="3446">and</wd><space/>
<wd l="1445" t="2957" r="10406" b="3446">Testing</wd>
</ln></para>
<para l="1308" t="3567" r="7762" b="4140" lsp="exactly" lspExact="571" language="en"
styleRef="paraStyle_1_3_31">
<ln l="1454" t="3533" r="7642" b="4022" underlined="none" subsuperscript="none" fontSize="2100"
fontFace="Bookman Old Style" fontFamily="roman" fontPitch="variable" spacing="0" scale="1102">
<wd l="1454" t="3538" r="1968" b="3893">at</wd><space/>
<wd l="1454" t="3533" r="2750" b="3893">the</wd><space/>
<wd l="1454" t="3533" r="3720" b="3893">Cox</wd><space/>
<wd l="1454" t="3533" r="4891" b="4022">Icing</wd><space/>
<wd l="1454" t="3533" r="6182" b="4022">Wind</wd><space/>
<wd l="1454" t="3533" r="7642" b="4022">Tunnel</wd>
</ln></para>
```

Fig. 19. Sample OmniPage 15 Formatted Page Snippet

The initial structure of the IDM schema was similar to that of OmniPage 14. This structure was, to a large degree, reflective of the visual structure of a page. Maintaining the same structural elements helped to minimize the re-coding cost to convert the extraction engine to be able to handle IDM as an input schema. Fig. 20 shows the resulting IDM version of the same sample document shown in OmniPage 14 format in Fig. 18 and OmniPage 15 format in Fig. 19.

```
<page width="12240" height="15840" x-res="300" y-res="300" orientation="0" pgno="3">
  <region left="10" top="858" right="12240" bottom="1450">
    <vert-white-space t="858" b="950" pct="0.581" loc="top" unit="px"/>
    <para t="950" l="1445" r="4853" b="1210" li="1368" ri="0" align="left"
        line-spacing="312">
      <line l="1445" t="950" r="4853" b="1210" ff="Times New Roman" fs="1400">
        <wd l="1445" t="950" r="4853" b="1210">NASA/TM—2003-212395</wd>
      </line>
    </para>
    <para t="950" l="9331" r="11626" b="1176" li="9216" ri="0" align="left"
        line-spacing="312">
      <line l="9331" t="950" r="11626" b="1176" ff="Times New Roman" fs="1400">
        <wd l="9331" t="950" r="11626" b="1176">AIAA–2003–0903</wd>
      </line>
    </para>
    <vert-white-space b="1450" t="1176" loc="bottom" unit="px" pct="1.730"/>
  </region>
  <region left="10" top="1450" right="12240" bottom="2645">
    <image l="10" t="1450" r="12240" b="2645"/>
  </region>
  <region left="10" top="2645" right="12240" bottom="4426">
    <vert-white-space t="2645" b="2957" pct="1.970" loc="top" unit="px"/>
    <para t="2957" l="1445" r="10406" b="3446" li="1296" ri="0" align="left"
        line-spacing="576">
      <line l="1445" t="2957" r="10406" b="3446" ff="Times New Roman" fs="2300">
        <wd l="1445" t="2957" r="4195" b="3317">Mixed-Phase</wd>
        <wd l="4334" t="2971" r="5386" b="3446">Icing</wd>
        <wd l="5515" t="2957" r="7829" b="3317">Simulation</wd>
        <wd l="7963" t="2957" r="8746" b="3317">and</wd>
        <wd l="8890" t="2971" r="10406" b="3446">Testing</wd>
      </line>
    </para>
    <para t="3533" l="1454" r="7642" b="4022" li="1296" ri="0" align="left"
        line-spacing="528">
      <line l="1454" t="3533" r="7642" b="4022" ff="Times New Roman" fs="2300">
        <wd l="1454" t="3581" r="1834" b="3893">at</wd>
        <wd l="1968" t="3533" r="2606" b="3893">the</wd>
        <wd l="2750" t="3542" r="3586" b="3893">Cox</wd>
        <wd l="3720" t="3547" r="4771" b="4022">Icing</wd>
        <wd l="4891" t="3533" r="6043" b="3893">Wind</wd>
        <wd l="6182" t="3533" r="7642" b="3893">Tunnel</wd>
      </line>
    </para>
    <vert-white-space b="4426" t="4022" loc="bottom" unit="px" pct="2.551"/>
  </region>
```

Fig. 20. IDM Formatted Page Snippet from OmniPage 14 Input

4.2 IDM generation

IDM documents were originally created by means of XSL 2.0 stylesheets. As discussed later, the TextPDF module was designed to output IDM directly. A different style-sheet is used for each type of OCR XML source document. I created stylesheets to support creation of IDM documents from either OmniPage 14 or 15 documents or Luratech ABBY 6 documents. I chose to use XSL 2.0 because of the many improvements over the XSL 1.0 version XSLT engines found in common usage in web browsers and distributed in most operating systems. Among the critical improvements are an expanded XPATH function library, expansion of XPATH expressions to include if-then-else tests and for-do loops, improved handling of temporary sub-trees and node sequences. One of the more challenging aspects of the XSL programming of the conversion stylesheets was the need to propagate style information both up and down the node tree. OCR programs do not always provide consistent information about fonts and font sizes in use on a page, since detection of this style information is very dependent upon the quality of the scanned page. The extraction program and template language are designed to detect style information at the line level. This style information may not be explicitly defined in the OCR XML "line" element. Rather, the style is at times defined at the paragraph or block levels and implicitly propagated to the subordinate structures. Or alternatively, the style information may be only defined at the word level. Either way, the stylesheet needs to propagate this style information to the line elements. Fig. 21 shows a snippet of the Luratech to IDM stylesheet. This function outputs a line element and accepts style information as parameters. Note that the font size is scaled into a

standardized size using the "meters per page" attribute. Appendix B shows the final version of the Luratech to IDM conversion stylesheet.

```
<xsl:template match="line">
   <xsl:param name="ff" as="xs:string"/>
   <xsl:param name="fs" as="xs:string"/>
   <xsl:param name="style"/>
   <xsl:param name="meter"/>
 <xsl:element name="line">
<!--  set up attribute for line -->
  <xsl:attribute name="l" select=" @l"/>
  <xsl:attribute name="t" select=" @t"/>
  <xsl:attribute name="r" select=" @r"/>
  <xsl:attribute name="b" select=" @b"/>
  <xsl:attribute name="base" select=" @baseline"/>
  <xsl:if test="exists($ff) and string-length($ff) gt 0">
   <xsl:attribute name="ff" select="$ff"/>
  </xsl:if>
  <xsl:if test="exists($fs) and string-length($fs) gt 0">
   <xsl:variable name="currfs" select="
                  (if (contains($fs, '.')) then substring-before($fs,'.') else $fs)
                  "/>
   <!-- scale font point sizes to the page size -->
   <xsl:attribute name="fs" select="my:scaleFS($currfs,$meter)"/>
  </xsl:if>
  <xsl:if test="exists($style) and string-length($style) gt 0">
   <xsl:attribute name="style" select="$style"/>
  </xsl:if>

  <xsl:apply-templates select="formatting" mode="split">
   <xsl:with-param name="face" select="$ff"/>
   <xsl:with-param name="fontsize" select="$fs"/>
   <xsl:with-param name="style" select="$style"/>
   <xsl:with-param name="l" select="@l"/>
   <xsl:with-param name="t" select="@t"/>
   <xsl:with-param name="r" select="@r"/>
   <xsl:with-param name="b" select="@b"/>
   <xsl:with-param name="meter" select="$meter"/>
  </xsl:apply-templates>
 </xsl:element>
</xsl:template>
```

Fig. 21. Line Function from IDM Conversion Stylesheet

4.3 Structural elements

The main high level structural elements used by the extraction engine of a document are pages, regions, paragraphs, whitespace, images and tables as shown in Fig. 22. The geometric boundaries of each of the structural elements are included as attributes. The "docInfo", "pageInfo", "regionInfo", "paraInfo" and "lineInfo" elements were originally included in the design to hold summary statistics about font and word usage for each of the corresponding elements. These statistics were used during experiments with document classification. We deprecated the use of these elements during IDM generation, since they can easily be calculated, if needed, from the input IDM and their generation dramatically slowed down the IDM XSL transformation and increased resource consumption.

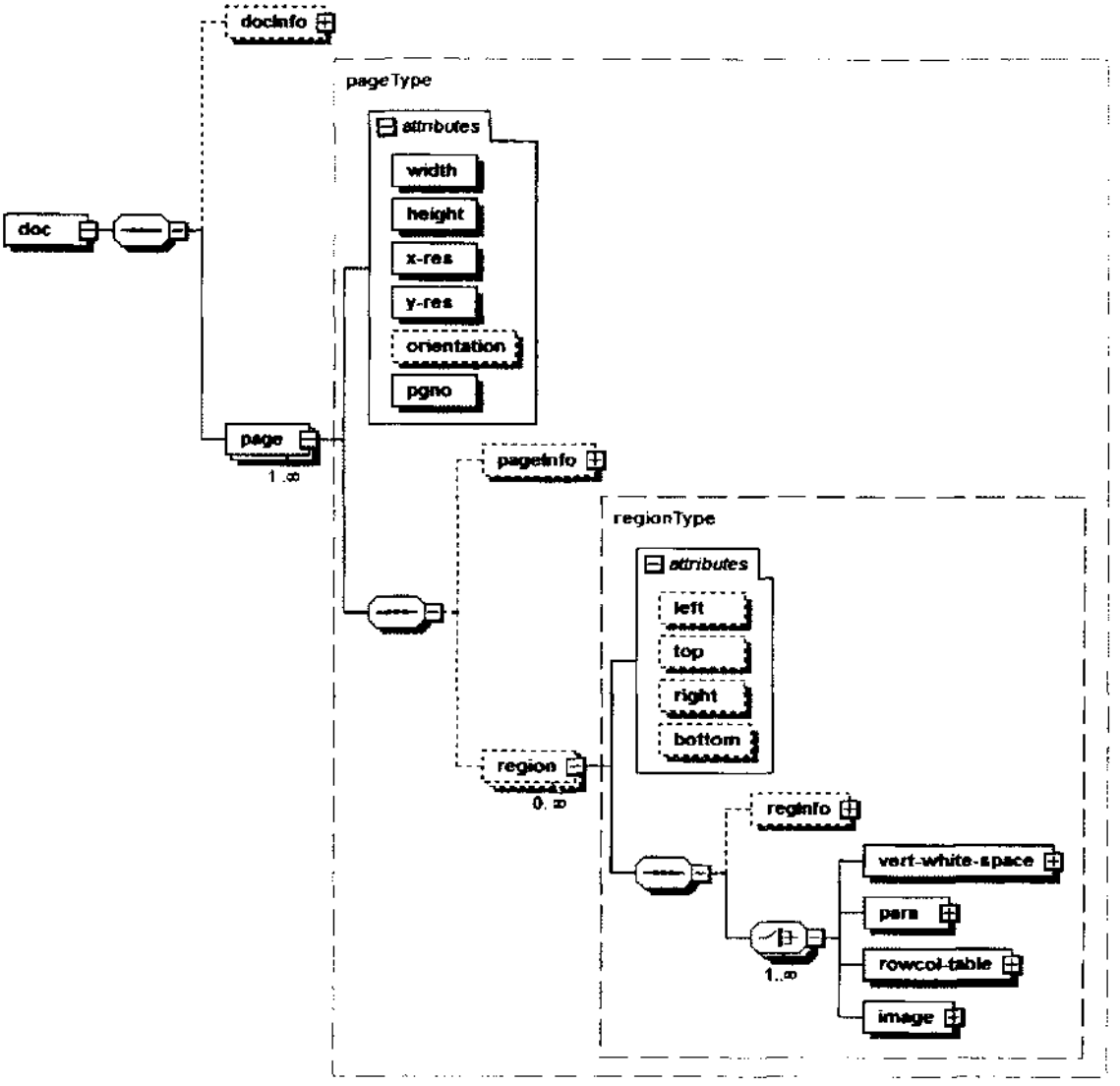


Fig. 22. IDM Version 1 Schema Structure

Style information such as font face, font size and font style, is recorded at the line and word levels. Alignment and line spacing are recorded at paragraph elements. Fig. 23 shows the structure of the paragraph, line and word elements. Explicit whitespace contained in the incoming document is encoded as a vert-white-space element. Tables

are composed of a sequence of cells that represent a virtual row-column table with each cell encoded with the upper-left coordinate and the row and column spans of the cell. While the stylesheets used for IDM generation faithfully replicate the table structures defined in the incoming documents, we found that due, to OCR uncertainties; we could not rely on using the table structures defined by the OCR for extraction. Often lines defining the tables are not correctly recognized resulting in table structures which do not match the original. Instead, the form extraction engine, which relies heavily on recognizing the table-like structure of a form uses the bounding box of the cell contents for processing.

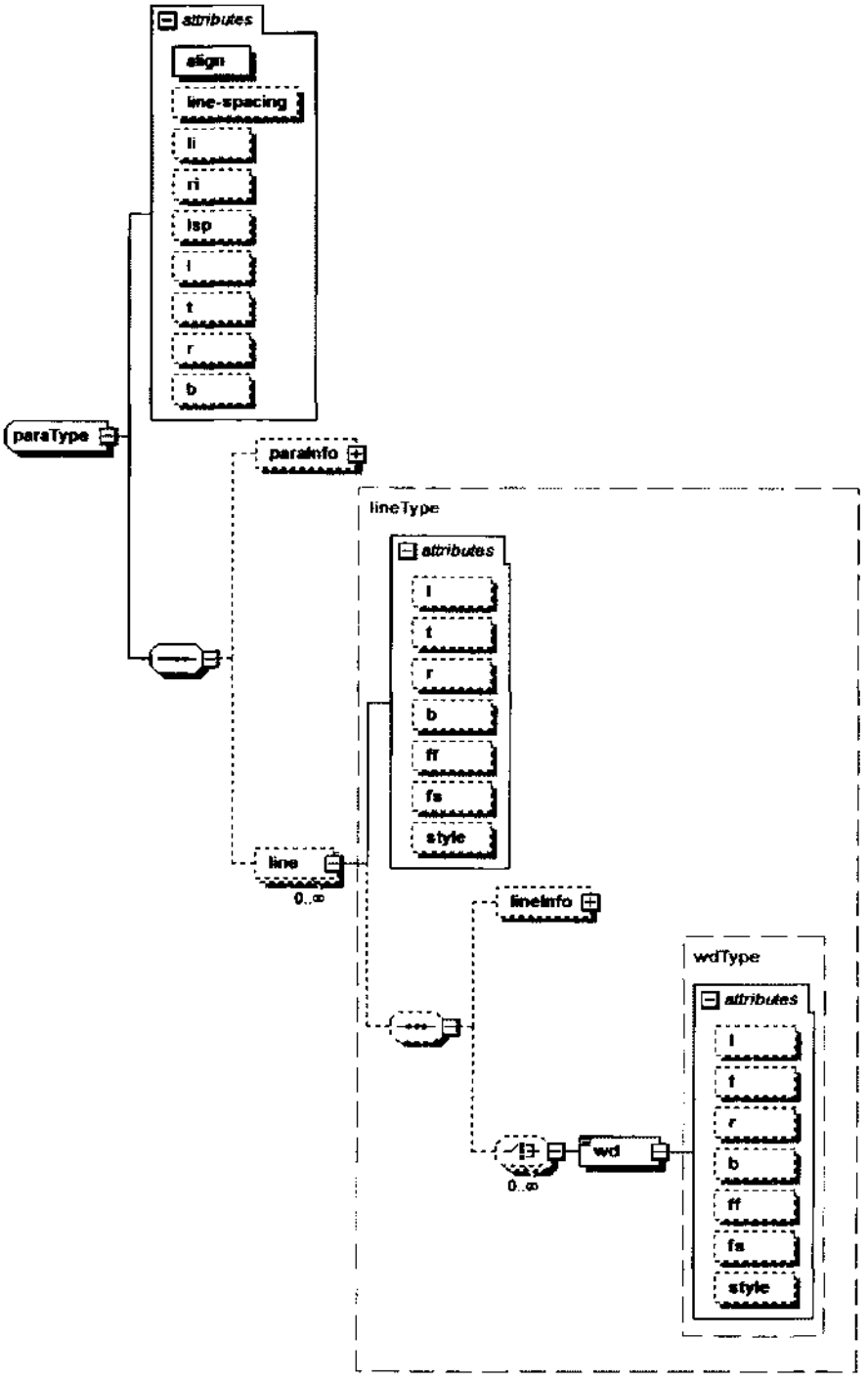


Fig. 23. IDM Paragraph, Line and Word Structure

4.4 Version 2

Not all documents require processing through an OCR program. A large portion of more recent documents are generated in text PDF format as opposed to scanned images of pages. As we started development of the TextPDF module to handle these text PDF incoming documents, we noted several areas in need of improvement in the IDM structure. The main issue faced was not having a consistent system of measuring geometry on a page. This affected both geometric positioning and font size values. OmniPage and Luratech employed different systems of specifying their font size which, in turn, caused some older templates to fail. The failures were caused by rule tests for specific font sizes. We addressed the geometry issue by introducing an attribute at the page level called "meter", defined as the number of measurement units contained in 1 meter. As an example, OmniPage output uses measurement units of 1/300 in, so a page from OmniPage OCR would be marked as <page meter="11811" ...>. TextPDF uses a much finer resolution so a typical meter value would be <page meter="56693" ...>. A common calculation is converting from measurement units to points when describing fonts, which would be computed as:

$$@fs * 2835 / (page@meter)$$

where *@fs* is in "measurement units", *2835* = the number of pts in a meter, and the new attribute *page@meter* is in measurement units/meter. Fig. 24 shows the structure of the document page and region elements of version 2 of the schema.

Another significant difference in version 2 is the ability to nest regions inside regions. This nesting of regions simplified processing for TextPDF and allows control of

the reading order for complicated column and row structures. Version 2 also deprecated the vert-white-space element, since such areas can be calculated from the bounding boxes of regions on a page. Fig. 25 shows the paragraph line and word structure.

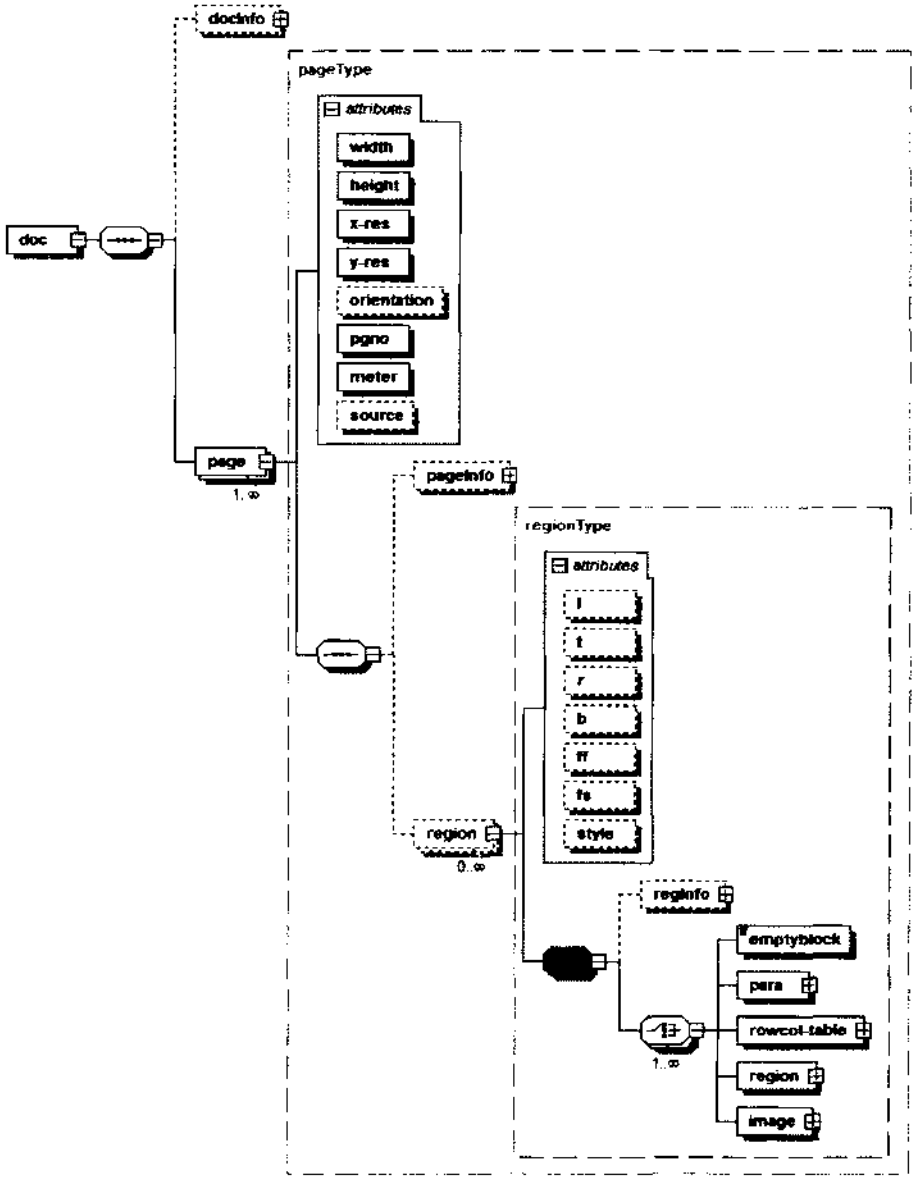


Fig. 24. IDM Version 2 Schema, doc, page and region Structure

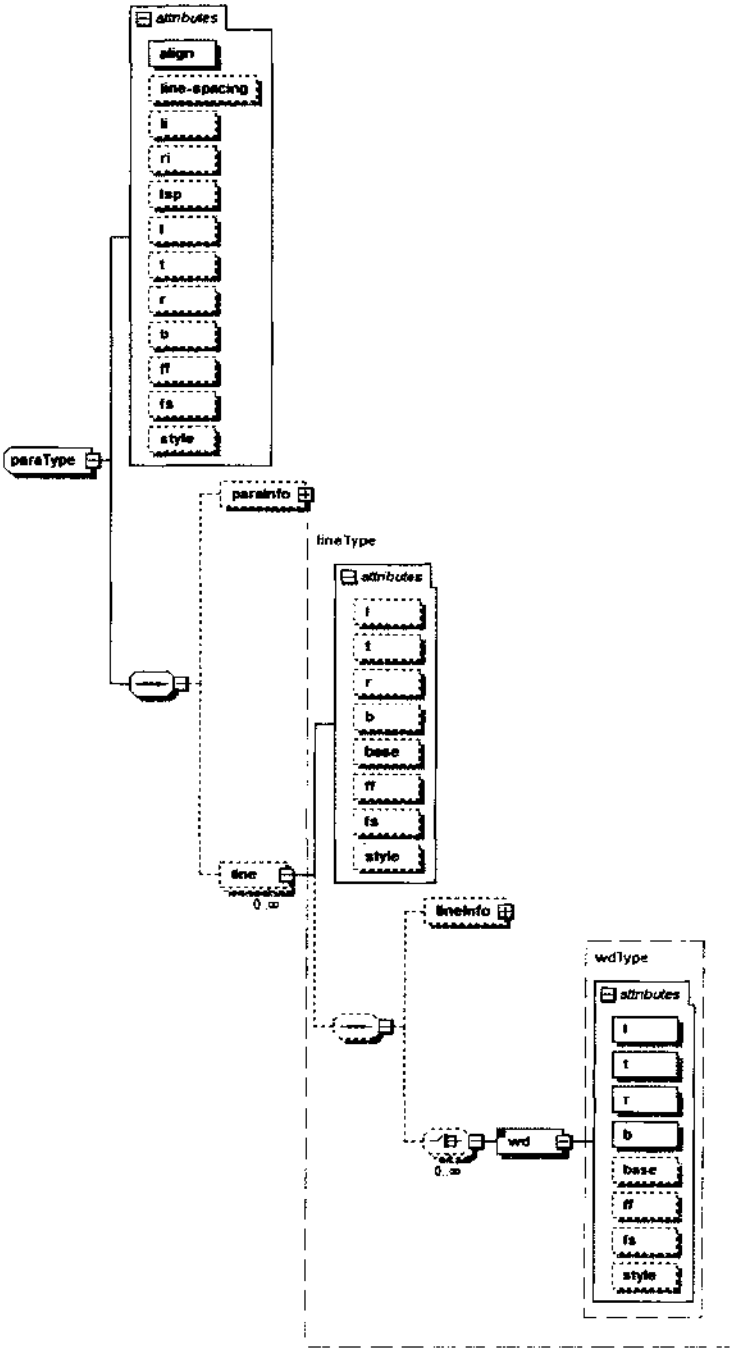


Fig. 25. IDM Version 2 Schema, Paragraph, Line and Word Structure

4.5 TextPDF generation

As noted earlier in Chapter 3, we replaced the OCR scanning process with a TextPDF module based on the Apache PDFBox library which loads the PDF document, attempts to extract formatted text from the desired pages (still defaulting to first and last five) and checks to see if the pages do indeed appear to be text PDF (a.k.a. "born digital" rather than scanned from paper copies). Text PDF pages are rendered into an IDM-like format called "raw IDM". Any of the desired pages that appear likely to be image PDF (i.e., scanned images of pages) are written into a new PDF file containing only those specific pages. That PDF file is passed on for OCR and conversion from the OCR-engine-specific format into raw IDM. It's quite likely that many documents will have all of their pages handled as text PDF, that a somewhat smaller number will have all of their pages treated as image PDF, and that some documents will be found to be a mixture of the two. In particular, someone could insert a text PDF version of an RDP form into a scanned document, or might scan an RDP form and insert that into a text PDF document. The raw IDM versions of the text PDF pages and the image PDF pages are merged and segmented, organized into words, lines, paragraphs, regions, etc.

Raw IDM is a superset of IDM – every IDM document is also a raw IDM document, but many raw IDM documents would not qualify as IDM. Raw IDM differs from the true IDM format in two ways:

-the text may be grouped into phrases containing multiple words and/or parts of words.

-all organizational structure required by IDM below the page level is optional. The only requirement is that each page contains at least one region.

The decision to pursue the IDM model was validated when DTIC changed their preferred OCR engine to the Luratech ABBY OCR program. It took me less than 20 hours to create an XSL stylesheet to convert Luratech to IDM. IDM also provided a standard page layout format for us to target during development of the TextPDF module.

CHAPTER 5

NON-FORM TEMPLATES

The processing of non-form templates is the heart of the system. The creation of a new template begins with identifying a group of documents which appear to the template author to be members of the same class. From there the author must systematically step through a sample document to generate the correct sequence of rules to extract the desired metadata. In most cases, there is no single "correct" sequence but rather a variety of sequences are possible depending on the experience of the author and the specific rules chosen to navigate the layout. There are two key goals that a template author must keep in mind: template must extract the correct metadata and template should not interfere with other templates already in the system. The first goal can be met by using the TemplateMaker program to assign in the template design, but the second goal is more difficult to meet without thorough testing to avoid greedy templates.

This chapter details the specifications and the rules of the template language in section 5.1 and section 5.2. The TemplateMaker program which enables rapid template development is described in section 5.3. The chapter finishes with a detailed review of the issue of greedy templates, section5.4.

5.1 Template language

The template language is derived from the PhD thesis by J. Tang [4] and retains the overall structure defined there, though many additions and modifications were made to refine the semantics resulting in the redesign of many elements.

5.1.1 Applying Templates to Documents

At the time of extraction, documents have been converted to IDM with the document divided into pages, pages divided into regions, regions into smaller regions and paragraphs, and paragraphs into lines of words. Each line is marked with certain "features" describing the primary font employed within that line. These features are: weight (bold or medium), slant (italic or normal), font size, allCaps (true or false), titleCase (true or false), and start of paragraph (i.e., is there one or more empty line preceding this one?) The non-form engine attempts to interpret the instructions encoded in a template in order to select lines that represent meaningful metadata. Templates are based on a line-by-line view of the document called a scroll. The scroll is a linear list of the lines that make up the pages of the document.

The essential structure of the template language is as follows: A template contains a set of rules designed to extract metadata from a document in a single class of similar documents. Each template has a unique name, called the template ID; a list of one or more page numbers indicating what portion of a document that template will examine; and an arbitrary number of rules. A rule describes how to locate and extract a block of text from the document. A rule consists of two main parts, the *begin* and *end* selectors, which describe how to locate the beginning and the ending of the desired block of text, respectively. The label of the rule gives a name to the block of text that it will extract, in most cases; it is the name of a metadata field (e.g., UnclassifiedTitle, PersonalAuthor).

When a template is applied to an IDM document, the template engine tries to apply the rules within the template, one after another, in the order they appear in the

template. As noted above, each rule has a *begin* and *end* selector. The engine first applies the *begin* selector. If that succeeds in finding the beginning of a block of text, then the engine applies the *end* selector. If that selector succeeds in locating the ending of a block of text, the lines of text are saved under the name given in the rule.

The template engine has a notion of a current line location within the document. When execution starts, the current line is the first line in the document. After any successful application of a rule, the current line is set to the first line just after the text extracted by that rule. *begin* selectors will normally search from the current line up to the end of the document. (More exactly, to the end of the set of pages designated for this template.) *end* selectors will normally search from the line matched by the *begin* selector to the end of the document. These search ranges can be modified by supplying a "scope" value on the *begin* or *end* selector. A scope denotes an alternative portion of the document that will be searched.

A scope of "document" in a *begin* selector resets the current line to the first line of the document and starts searching from there. The search continues to the end of the document. A scope of "document" in an *end* selector means that it should search from the line matched by the *begin* selector to the end of the document, since this is how *end* selectors normally work anyway, "document" scope are not usually found on an *end* selector.

A scope of "page" in a *begin* selector resets the current line to the first line of the current page and starts searching from there. The search continues to the end of that page. A scope of "page" in an *end* selector means that it should search from the line matched by the *begin* selector to the end of the current page.

A scope may also give the name of any previously extracted block of text. In a *begin* selector this resets the current line to the first line of that block of text and starts searching from there. The search continues to the end of that block of text. In an *end* selector, such a scope means that it should search from the line matched by the *begin* selector to the end of that block of text.

If no block of text has been successfully extracted with the given name (i.e., there is no prior rule with that name or the rule with that name failed to locate a block of text), and the rule is marked as required, then execution of the template is halted with no output. The mechanism for marking a rule as required is covered in the next section.

5.2 Rule Definition

The options for each rule are indicated by the attributes as defined in Table 6. Each of the rules in a template can be marked as ignored, meaning that the block of text located by the rule will not be included in the final metadata. Ignored rules are used as intermediate steps in aiding later rules to locate "real" metadata.

TABLE 6
Rule Modifier Attributes

Attribute	Required	Possible values	Default	Meaning
Min	no	Non-neg number	1	The minimum number of repetitions of this field that should be expected in the document.
Max	no	Non-neg number	1	The maximum number of repetitions of this field that should be expected in the document.
Ignore	no	"yes" or "no"	no	If "yes", this field is used merely as a convenience to identify a position within the document. No metadata value will actually be extracted into the output.
Require	no	"yes" or "no"	no	If "yes", then failure to successfully locate and extract this field indicates that something is wrong (e.g., this template describes a different document layout than is actually present in this document). In such a case, execution of the template is halted with no output generated for any metadata fields.
Filter	no	Regular expression	.*	Used to select a portion of the raw text in the indicated lines. If the regular expression contains no parentheses, then the portion of the text matching the entire regular expression is extracted. If the regular expression contains parentheses, then the portion of the text matching the parenthesized sub-expressions is extracted.

A rule can be marked as "required", meaning that if the rule cannot locate a block of text, then we assume that this document is not actually in the class that this template is trying to describe, and the attempt to apply the template halts with no output. For example, if documents in a class always have a date immediately after the title, then we would mark the rule to extract that date as "required", because if there is no date after the title, the document we are looking at must not be in that class. On the other hand, if the date is present in some documents of the class but not in others, we would mark the rule as not required, so that the date is extracted when present but execution of the template continues whether the date is there or not.

A rule may also have a filter, which can select a portion of the located block of text to be extracted. Filters are defined using regular expressions. For example, if a document contains a line "Report Date: 01/21/2009", we might apply a filter to extract only the date "01/21/2009" and not the words "Report Date:" that precede the desired value.

5.2.1 Line selectors

The *begin* and *end* selectors of a rule each contains a line selector expression which are described in Table 7. Each line selector offers a distinct way to search for a line of text that represents the beginning (or ending) of a block of text. Examples include searching for specific strings in the document, searching for the end of a paragraph, or searching for lines followed by large blank vertical spaces. Selector expressions are modified by an "inclusive" value, which indicates whether the desired beginning/ending location is supposed to be the line that actually matches the selector expression or the line just before the matching line, or the line just after the matching line.

TABLE 7
Line Selector Descriptions

Selector	Meaning
Mf	Names a previously extracted metadata field. (It is possible to extract multiple field values with the same field name, in which case this refers to the most recently extracted value with this name.) In a <begin> rule, selects the line chosen by the <end> rule of that metadata field. In an <end> rule, selects the line chosen by the <begin> rule of that metadata field.
beforeField*(n, mf)*	Selects the line *n* number of (non-blank) lines prior to an existing metadata field *mf*.
beforeTag*(n, mf)*	Deprecated in favor of 'beforeField' - selects the line *n* number of (non-blank) lines prior to an existing metadata field *mf*.
Begin	The first line of the document.
Beginwithmonth	Find a line begins with a month such as "March " , "January", etc.
Boldchange	Deprecated - Find a line that differs from the current line in that one is in bold and the other is not (same as changeWeight).
changeWeight	Find a line that differs from the current line in that one is in bold and the other is not (same as boldchange).
changeSizeOrWeight	Find a line that differs from the current line in that one of the following is true: - Font size is different - One is bold and the other is not bold.
changeSizeOrWeightOrAllCaps	Find a line whose features are different from those of the current line. A typography change occurs when any of the following are true: - Font size is different - One is bold and the other is not bold - One is Allupcase and the other is not.
changeSizeOrWeightOrCaps	Find a line whose features are different from those of the current line. A feature change occurs when any of the following are true: - Font size is different - One is bold and the other is not bold - One is Allupcase and the other not - One is leadingcase and the other is not.
cityState	For recognizing strings containing City, State or City, State Zip patterns.
chooseFieldBegin(*mf_1, mf_2, ... mf_n*)	The selector attempts to locate the listed meta fields left to right and selects the first which has a successful extraction. Depending on selector we then match the begining of that field.
chooseFieldEnd(*mf_1, mf_2, ... mf_n*)	The selector attempts to locate the listed meta fields left to right and selects the first which has a successful extraction. Depending on selector we then match the end of that field.
containsName	Find a line that appears to contain a personal name. Names may be last name first or first name first. [Replaces former unused nameformat selector]
containsOnlyName	Find a line that appears to contain only a personal name. Names may be last name first or first name first. [Checks if line has 4 or more words, it must have a comma,period,colon,semi, paren, bracket, square bracket, or the word 'and']

TABLE 7 Continued

Selector	Meaning
Current	Matches the current line (see above).
Dateformat(formats)	Find a line that has a date with specified format. Formats is a "|"-separated list of date formats, with each format being any pattern that would be accepted by Java's SimpleDateFormat class. Example: dateformat(MMMM dd, yyyy|MM/dd/yyyy) would accept lines such as "January 23, 2001" or "01/23/2001".
Dateformat	Find a line that has a date with format "dd month yyyy" "month dd, yyyy" or "month yyyy", where "month" means a month string such as "Jan ", "September", etc.
End	The last line of the document.
endleftof*(tag)*	Locates 1st line not to the left of the beginning of the *tag* (see 'leftof', below).
endrightof*(tag)*	Locates 1st line not to the right of the beginning of the *tag* (see 'rightof', below)).
Featurechange	Deprecated - old name for changeSizeOrWeightOrCaps.
Firstpart	Deprecated: matches the current line (same as "current" or "onesection" in an <end> rule)
Largersize	Find a line whose size is larger than current line (Lines with string length less than 10 are ignored.)
largeststrsize (*v1,v2*)	Searches for the largest font size among lines between positions v1 and v2 that meet the following criteria: - Its length is larger than 11 - It has more than 1 words - Average word length is between 4 and 13 - More than 70% of the characters are alphabetic.
looseLargeststrsize (*v1,v2*)	Searches for the largest font size among lines between positions v1 and v2 that meets the following criteria: - More than 70% of the characters are alphabetic.
Lastpart	Line of the previous field's end.
Layoutchange	Deprecated - old name for changeSizeOrWeight.
leftof*(tag)*	Locates 1st line to the left of and overlapping vertically with the block of statements extracted as the metadata field *tag*.
Onesection	Generally used only in an <end> rule. Selects the same line as the <begin> rule. This is functionally equivalent to "current".
pageChange	Finds the first line in the next page, [Note that only pages whose page numbers are given the template will be available.]
paraEnd	Finds a line preceding a line that was indicated as the start of a paragraph (by OCR).
ParaEnd	Deprecated (in favor of "paraEnd"): Finds a line preceding a line that was indicated as the start of a paragraph (by OCR).
regexps(*re*)	Find a line matching a regular expression *re*.
rightof*(tag)*	Locates 1st line to the right of and overlapping vertically with the block of statements extracted as the metadata field *tag*.

TABLE 7 Continued

Selector	Meaning
size (*s1,s2*)	Return true if a line's font size is between *s1* and *s2*.
sizechange (*x*)	Find a line whose font size is different from that of the current line. To overcome OCR errors, a change with difference less than x is ignored.
sizepctchange(x)	Find a line whose font size is different from that of the current line by more than the x percent.
Smallersize	Find a line whose size is smaller than current line (lines with string length less than 10 are ignored.)
Stringmatch	Match a special string -- see below.
titleCaseOrAllCaps(k)	Find a line that is in all caps or in title case (according to the usual English rules for capitalizing titles, which allows articles, propositions, etc., to remain unapitalized) and that has *k* or more words. If the parameter is omitted, k==4 as a default.
typoGraphychange	Deprecated - old name for changeSizeOrWeightOrAllCaps.
verticalSpace	Find a line preceded by an empty line.
verticalSpace(*s*)	Searches for any line that is followed by a vertical space greater than or equal to scale (*s*) * *lineheight*, where *lineheight* is *multipiler* times the *height* of the bounding box and *multiplier* ranges from *1.0..1.2*.
verticalSplit(*k,n*)	Searches the current page for the *k-1* largest internal vertical white spaces (ignoring the top and bottom margins), thereby splitting the page into *k* vertical blocks. Returns the line number beginning the (*n*)_st such block. By definition, if *n* == 0, returns the first nonempty line on the page. If *n*==*k*, returns the last nonempty line on the page.

The *stringmatch* selector is a special case in that it is formed as an XML element rather than a simple test within a *begin* or *end* rule. The XML structure is needed to support the set of options available as shown in Table 8. The text to be matched is inside the *stringmatch* element. The "fuzzy" attribute is primarily used to compensate for OCR recognition errors.

TABLE 8
Options for stringmatch Selector

Attribute	Required	Possible values	Default	Meaning
case	no	"yes" or "no"	yes	Yes: upper/lower case is significant No: upper/lowercase differences are ignored
loc	yes	"beginwith", "onsection", "contain", "endwith"		Modifies how much of the text in a line must match the provided text: - beginwith: the line must begin with the provided text - endwith: the line must end with the provided text - onesection: the entire line must match the provided text - contain: the provided text must occur somewhere within the line
fuzzy	no	Non-negative integer	0	Match succeeds even if the line differs from the provided text by this number of single-character changes (Levenshtein edit distance)

Even though a template writer has 45 line selectors available, usage analysis of 128 templates developed for the DTIC collection shows that the most common line selectors for the *begin* rule are the "stringmatch" and the "previous" metadata field selectors as shown in Table 9. The most common selectors for the *end* rule are the "onesection" and "stringmatch" selectors. These usage statistics are consistent with our experience in developing templates where a common pattern is to use a "stringmatch" to find the first metadata field on a page and then additional metadata fields follow sequentially after the first. The *end* selector usage is also consistent with the observation that in most cases, a field is contained on a single line.

TABLE 9
Line Selector Usage for DTIC Collection

begin command	Times Used	end command	Times Used
string	330	Onesection	437
Metadata field	263	String	160
dateformat	69	verticalSpace	58
begin	62	Regexps	40
regexps	47	changeSizeOrWeightOrCaps	40
lastpart	45	Dateformat	37
largeststrsize	35	changeSizeOrWeight	36
pageChange	17	End	19
titleCaseOrAllCaps	9	typoGraphychange	12
containsName	7	pageChange	11
containsOnlyName	5	paraEnd	7
leftof	3	changeSizeOrWeightOrAllCaps	7
verticalSplit	3	Featurechange	7
firstpart	2	Layoutchange	5
rightof	2	containsOnlyName	4
beginwithmonth	2	ParaEnd	4
largersize	1	beginwithmonth	3
chooseFieldBegin	1	Largersize	2
		Sizepctchange	2
		paraChange	2
		containsName	2
		titleCaseOrAllCaps	2
		chooseFieldEnd	1
		verticalSplit	1
		cityState	1
		Sizechange	1

5.3 TemplateMaker program

Developing templates using the native XML structure proved to be extremely difficult even for our most experienced developers. Once we were able to modularize the architecture and separate out the non-form extraction engine, I developed the TemplateMaker tool. The TemplateMaker, shown in Fig. 26, is a GUI tool to help in the

template creation process. It allows a template author to edit templates, adding and modifying rules, and to quickly apply the modified template to a set of documents to see the effects of each rule. TemplateMaker can be used by authors who have no understanding of XML or who simply wish to avoid dealing with the finicky details of writing valid XML. During on-site training of the system, a variety of DTIC personnel ranging from clerical staff to managers demonstrated an ability to use the TemplateMaker to craft at least one template after a half-day training session. One of the most valuable features of the TemplateMaker is the ability to show the user the actual "scroll" or list of lines resulting from the segmentation of the document. This is important because on multi-columnar type documents it may not be obvious how the lines will be ordered by the engine.

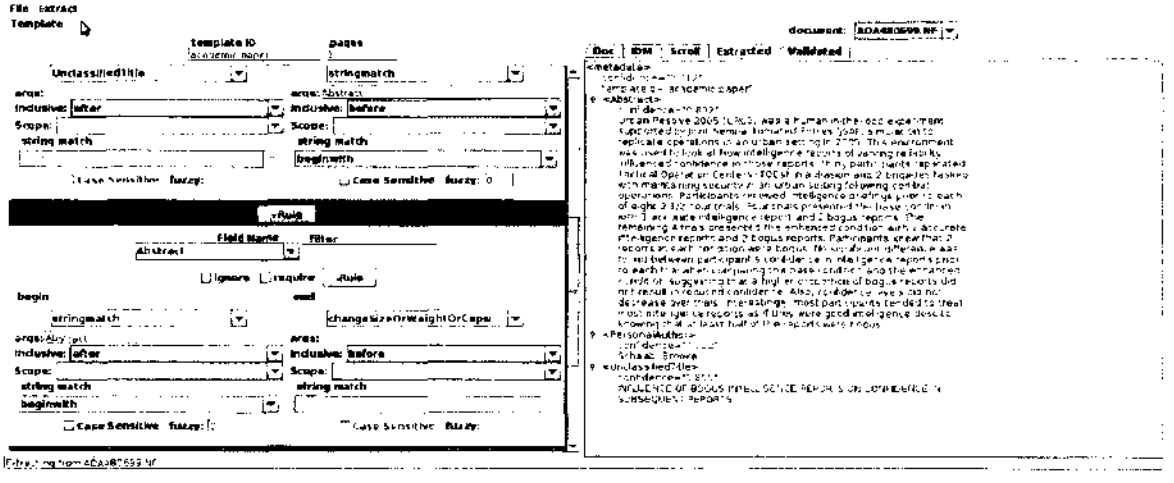


Fig. 26. TemplateMaker GUI

5.4 Greedy templates

One of the most persistent issues faced by template creators is the concept of a greedy template or a template that contains rules that are too general and that match unintended text on a page. This section addresses the problem in detail, offers methods for alleviating the issues and shows a case study demonstrating the problem.

5.4.1 Explanation and samples

The post hoc classification applies every template in the collection to each document and the expectation is that the template selected will be the one designed specifically for that document set. The most significant recurring issue for post hoc classification is that a new template developed for a new set of documents in a collection will match other documents and possibly erroneously outscore existing templates. This phenomenon is called the greedy template. When a greedy template is included among the template collection, it can result in numerous false positives, i.e., the template matching documents outside its class. This can be a problem when the greedy template fields match random text that does not represent the correct metadata. These erroneous fields are scored by the validator and may in fact receive reasonable scores. For example, the UnclassifiedTitle and Abstract validation may score high on a random sentence in a document.

Fig. 27 shows an example of a greedy template we encountered during our work with the GPO EPA collection. I detected this template while evaluating classification performance during template development. At the time we were attempting rapid expansion of the template set for the GPO EPA collection and had five different template authors with various levels of experience writing templates. During analysis of one run

of the document set, I found that the template shown in Fig. 27 selected 56 documents with no correct extractions and that another template that I was investigating, "title2col", was not matching any of the correct documents it previously matched. Once this greedy template was removed from the template set, the extractions cleaned up and the "title2col" template picked up the expected documents.

It is not intuitively obvious to even the experienced template writer that this is a bad template. It is too general since it will match virtually any document with the string "EPA" in it. Once the "EPA" is found, the "notes_500" and "series_490a" fields are designed to simply take the next two lines, regardless of the content. "series_490a" is a validated field so the template gets points for the irrelevant data.

The most reliable method for detecting greedy templates is to continually run regression tests during the template creation process. The creation of regression tests is one of our recommended best practices for operation of the extraction software. As templates are created, a sample document and resulting metadata are saved into the regression cache. As new templates are added, the collection of regression documents is rerun and the extractions are confirmed. The detection of any changes to the expected metadata is cause for investigation and adjustment.

```
<?xml version="1.0" encoding="UTF-8"?>
<structdef pagenumber="1" templateID="title2col-noPlace">

 <publisher_260b min="1" max="1" require="yes">
  <begin inclusive="current">regexps((?i)EPA|environmental
+protection +agency)</begin>
  <end inclusive="current">onesection</end>
 </publisher_260b>

 <notes_500 min="1" max="1" require="yes">
  <begin inclusive="after">publisher_260b</begin>
  <end inclusive="current">onesection</end>
 </notes_500>

 <series_490a min="1" max="1" require="yes">
  <begin inclusive="after">notes_500</begin>
  <end inclusive="current">onesection</end>
 </series_490a>

 <title_245a min="1" max="1" require="no">
  <begin inclusive="after">series_490a</begin>
  <end
inclusive="before">regexps((\w*[,]\s\s*)|(Jan|Feb|Mar|Apr|May|Jun|Jul|
Aug|Sep|Oct|Nov|Dec)[.a-z]+ +\d\d\d\d)</end>
 </title_245a>

 <dateOfPubl_260c min="1" max="1" require="no"
filter=".*?(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)([.a-z]+
+\d\d\d\d)">
  <begin
inclusive="current">regexps(.*?(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep
|Oct|Nov|Dec)[.a-z]+ +\d\d\d\d)</begin>
  <end inclusive="current">onesection</end>
 </dateOfPubl_260c>

</structdef>
```

Fig. 27. Sample Greedy Template from GPO EPA Collection

5.4.2 Techniques for avoiding greed

There is no guaranteed method for preventing greedy templates. We devised two methods to at least reduce the greedy behavior of some templates.

We added the "require" attribute to the metadata rule selector to help eliminate documents which do not contain key identifying content in the class. The effectiveness of the "require" is dependent on the uniqueness of the selected content. Note that in Fig. 27, the "require" on the publisher is a good start but putting the "require" on the notes_500 and series_490a is not useful since the selectors merely pick up a single line.

An additional method is to add a classification "boost" rule. The classification scoring system is designed to add a large value (+100) to any metadata field which begins with an "underscore". If the document class contains unique content, e.g. a corporate author name, then the template can be written to select that content with a boost rule to ensure that this template will be selected.

5.4.3 Greedy template case study

In order to test the impact of adding greedy templates to a template collection set, I added two templates which should be greedy. The first template "grabby1", Fig. 28, is a template which was removed from the DTIC collection because it was too greedy. The second template "grabby2", Fig. 29, was constructed using a set of basic rules to select the largest size string from the top half of the first page as the "title" and the largest size string from the bottom half of the page as the "abstract". These templates were added to our DTIC testbed of 1625 documents and run through the process. This run resulted in 325 documents being assigned to "grabby1" and 213 documents assigned to "grabby2". Of those 538 documents, 183 had "acceptable" validation scores with "acceptable" being

defined as greater than 0.64 (see section 7.10) only 106 of the "acceptable" documents had good title extractions and none had correct abstracts.

```
<structdef pagenumber="1" templateID="grabby1">
 <UnclassifiedTitle>
  <begin inclusive="current">largeststrsize(0, 0.4)</begin>
  <end inclusive="before">layoutchange</end>
 </UnclassifiedTitle>
 <PersonalAuthor>
  <begin inclusive="after">UnclassifiedTitle</begin>
  <end inclusive="current">onesection</end>
 </PersonalAuthor>
 <CorporateAuthor>
  <begin inclusive="after">PersonalAuthor</begin>
  <end inclusive="before"><stringmatch case="no"
loc="beginwith">ABSTRACT|Abstract|RESUM|Summary|INTRO|Int
ro|SUMMARY|Resu|RES</stringmatch></end>
 </CorporateAuthor>
 <Abstract>
  <begin inclusive="after"><stringmatch case="no"
loc="beginwith">ABSTRACT|Abstract|Summary|SUMMARY</string
match></begin>
  <end inclusive="before">layoutchange</end>
 </Abstract>
</structdef>
```

Fig. 28. Greedy Template Example "grabby1"

```
<structdef pagenumber="1" templateID="grabby2">
 <UnclassifiedTitle min="1" max="1">
  <begin inclusive="current">largeststrsize(0, 0.5)</begin>
  <end inclusive="before">layoutchange</end>
 </UnclassifiedTitle>

 <Abstract min="1" max="1">
  <begin inclusive="current">largeststrsize(0.5, 0.9)</begin>
  <end inclusive="before">layoutchange</end>
 </Abstract>
</structdef>
```

Fig. 29. Greedy Template Example "grabby2"

CHAPTER 6

DOCUMENT SIMILARITY EXPERIMENTS

This chapter details experiments conducted in document classification using similarity measures. Early in the project we were looking for a reliable method for document classification for selecting the correct template for extraction, where we define "reliable" as having a precision and recall measurement better than 80%. My initial efforts focused on investigating various document similarity measures as an extension of the POINT page search work detailed in Chapter 3. I designed these experiments to use a relatively small training set, on the order of 5-6 documents as opposed to a large training set typically found in machine learning systems. This is also a manageable number for a template writer to examine to develop a template.

6.1 Experimental Setup

The test collection was based on a randomly selected set of 2000 documents from the DTIC collection. 522 of those documents were found to be non-form documents. I conducted a visual classification procedure on these non-form documents and was able to divide 407 documents into 38 classes of two or more examples. The remaining 115 documents appeared to be unique (or singleton) files.

For these similarity experiments, I selected 5-6 training set documents for each class randomly from the test collection. Classes with fewer than 5 members were excluded resulting in 21 different classes from the original 38. Each experiment was run against the entire set of the 522 selected nonform documents. I used a k-nearest neighbors (k-NN) method with K equal to 5. The top 5 most similar training documents

are examined and if 4 of the 5 are the same class, that is the selected class, otherwise there is no selection.

6.2 Similarity testing experiments

The following sections detail the experimental results of testing six different methods for classifying documents based on similarity. From the test collection described above, we used the 21 different classes which had at least five member documents. Appendix D contains sample images of each of the classes used in this series of experiments.

6.2.1 Layout Distance

I investigated the distance measure algorithms laid out in [48]. The method that I tested attempts to minimize the Manhattan distance between corner points of paragraph blocks as the distance together with the percentage of block overlap. The blocks are matched using the minimum distances using the Hungarian algorithm [69] with the specific JAVA implementation found at [70]. Table 10 shows the results of the block layout distance. The "Correct" column is the total of correctly identified documents belonging to this class. The "Incorrect" column is the number of documents in this class which were not correctly identified. The "New" column is the number of documents from some other class that has been mis-identified. The "Precision" column is calculated as:

Precision= Correct / (Correct + Incorrect)

The "Recall" column is calculated as:

Recall = Correct / (Correct + New)

TABLE 10
Layout Distance Similarity Results

Class	Correct	Incorrect	New	Precision	Recall
AU	85	1	0	99%	100%
EAGLE-IMAGE	32	0	3	100%	91%
RAND-ARROYO2	4	24	0	14%	100%
LOGI	4	23	0	15%	100%
ERDC	18	8	1	69%	95%
RAND-BRIEF2	12	8	0	60%	100%
RANDTECH	8	8	3	50%	73%
RAND-NOTE	8	6	0	57%	100%
EAGLE-TEXT	13	0	0	100%	100%
RAND-ARROYO	6	6	1	50%	86%
SIGNATUR	0	10	0	0%	0%
RAND-ARC	0	9	0	0%	0%
RESEARCH	0	9	0	0%	0%
TOPLOG-2COL	0	9	0	0%	0%
BOTTOM-BLOCK	1	7	0	13%	100%
ATOM	0	7	0	0%	0%
CPRC	0	6	0	0%	0%
RAND-BRIEF1	2	4	0	33%	100%
RAND-LEFT	0	6	0	0%	0%
HORIZ	4	1	0	80%	100%
WARCOLLEGE	0	5	0	0%	0%

Table 10 shows the results of the Layout Similarity test. 14 out of 21 of the document classes had precision under 50% and only 3 of 21 had a precision of 90% or higher. This table and the tables in the following sections highlight the 11 classes with 10 or fewer instance documents. The performance of those classes may not be indicative of the performance of the similarity measure since there are so few sample documents available and the training set is included in the test set. The one conclusion we can draw is that any class which does not match at least 5 documents is not suited for this measure.

6.2.2 MxN Overlap

This similarity measure uses a variant of the MxN bin method proposed by [45]. In this method, a page is cut into 100*200 bins in equal size. Each bin is marked as a graphic bin, a text bin, or a white space bin. A bin is a graphic if any part of the bin is overlapped by a graphic block and no part is overlapped by a text block. Any bin overlapped by any part of a text block is classified as text. The similarity is the percentage of bins on each page that match markings. Table 11 shows the results of the MxN Overlap results.

TABLE 11
MxN Overlap Similarity Results

Class	Correct	Incorrect	New	Precision	Recall
AU	83	3	0	97%	100%
EAGLE-IMAGE	32	0	0	100%	100%
RAND-ARROYO2	19	9	5	68%	79%
LOGI	2	25	0	7%	100%
ERDC	24	2	0	92%	100%
RAND-BRIEF2	18	2	0	90%	100%
RANDTECH	2	14	1	13%	67%
RAND-NOTE	11	3	0	79%	100%
EAGLE-TEXT	9	4	0	69%	100%
RAND-ARROYO	6	6	0	50%	100%
SIGNATUR	0	10	0	0%	0%
RAND-ARC	8	1	1	89%	89%
RESEARCH	0	9	0	0%	0%
TOPLOG-2COL	4	5	0	44%	100%
BOTTOM-BLOCK	0	8	0	0%	0%
ATOM	0	7	0	0%	0%
CERC	1	5	0	17%	100%
RAND-BRIEF1	4	2	0	67%	100%
RAND-LEFT	0	6	0	0%	0%
HORIZ	4	1	0	80%	100%
WARCOLLEGE	0	5	0	0%	0%

11 out of 21 of the document classes had precision under 50% and only 4 of 21 had a precision of 90% or higher.

6.2.3 Common Vocabulary

This method is based on the intuition that documents in the same class may come from the same source and may therefore have similar publisher information or codes. The method gathers the words common to each document in the training set for the class and attempts to find the best match with a floor of 75% matching. I ran two different experiments. The first used words from just the 1st page and the other used the common words of the 1st five pages. The complete list of word can be found in Appendix D.

TABLE 12
Vocab Matching Similarity Results

	Vocabulary 1 page					Vocabulary 5 page				
Class	Correct	Incorrect	New	Precision	Recall	Correct	Incorrect	New	Precision	Recall
AU	83	3	0	97%	100%	78	8	0	91%	100%
EAGLE-IMAGE	0	32	0	0%	0%	25	7	2	78%	93%
RAND-ARROYO2	20	8	5	71%	80%	21	7	6	75%	78%
LOGI	26	1	11	96%	70%	7	20	0	26%	100%
ERDC	14	12	1	54%	93%	17	9	4	65%	81%
RAND-BRIEF2	9	11	0	45%	100%	16	4	0	80%	100%
RANDTECH	0	16	0	0%	0%	13	3	10	81%	57%
RAND-NOTE	14	0	0	100%	100%	12	2	0	86%	100%
EAGLE-TEXT	13	0	0	100%	100%	7	6	0	54%	100%
RAND-ARROYO	0	12	0	0%	0%	9	3	2	75%	82%
SIGNATUR	10	0	0	100%	100%	10	0	1	100%	91%
RAND-AEC	0	9	0	0%	0%	5	4	1	56%	83%
RESEARCH	6	3	3	67%	67%	8	1	9	89%	47%
TOPLOG-2COL	5	4	0	56%	100%	5	4	0	56%	100%
BOTTOM-BLOCK	0	8	0	0%	0%	4	4	1	50%	80%
ATOM	0	7	0	0%	0%	7	0	0	100%	100%
CPRC	0	6	0	0%	0%	5	1	6	83%	45%
RAND-BRIEF1	4	2	0	67%	100%	6	0	0	100%	100%
RAND-LEFT	2	4	22	33%	8%	5	1	0	83%	100%
HORIZ	0	5	0	0%	0%	5	0	28	100%	15%
WARCOLLEGE	5	0	8	100%	38%	5	0	2	100%	71%

For 1 page, 10 out of 21 of the document classes had precision under 50% and only 6 of 21 had a precision of 90% or higher. For 5 page, 2 out of 21 of the document classes had precision under 50% and only 6 of 21 had a precision of 90% or higher.

6.2.4 MXY Tree

This method is an implementation of the MXY Tree [4] with the tree being built by alternating horizontal and vertical cuts along whitespace between blocks. The MXY Tree structure is encoded in a string by concatenating a "V" or an "H" for each horizontal cut in sequence, and finally at the last block in each branch, assigning a "g" for a graphic block and a "t" for a text block. The similarity is measured by calculating the edit distance between encodings.

TABLE 13
MXY Tree Similarity Results

Class	Correct	Incorrect	New	Precision	Recall
AU	84	2	0	98%	100%
EAGLE-IMAGE	16	16	1	50%	94%
RAND-ARROYO2	0	28	0	0%	0%
LOGI	3	24	2	11%	60%
ERDC	5	21	0	19%	100%
RAND-BRIEF2	10	10	1	50%	91%
RANDTECH	0	16	0	0%	0%
RAND-NOTE	0	14	0	0%	0%
EAGLE-TEXT	13	0	1	100%	93%
RAND-ARROYO	0	12	0	0%	0%
SIGNATUR	0	10	0	0%	0%
RAND-ARC	0	9	0	0%	0%
RESEARCH	0	9	0	0%	0%
TOPLOG-COL	0	9	0	0%	0%
BOTTOM-BLOCK	0	8	0	0%	0%
ARMS	0	7	0	0%	0%
CSRC	0	6	0	0%	0%
RAND-BRIEF1	2	4	0	33%	100%
RAND-LEFT	0	6	0	0%	0%
HORIZ	1	4	0	20%	100%
WARCOLLEGE	0	5	0	0%	0%

19 out of 21 of the document classes had precision under 50% and only 2 of 21 had a precision of 90% or higher.

6.2.5 MXY Tree Plus MxN

This method calculates the similarity by adding values found by the MXY Tree and the MxN methods described above.

TABLE 14
MXY Tree Plus MxN Similarity Results

Class	Correct	Incorrect	New	Precision	Recall
AU	85	1	0	99%	100%
EAGLE-IMAGE	30	2	0	94%	100%
RAND-ARROYO2	0	28	0	0%	0%
LOGI	3	24	1	11%	75%
ERDC	12	14	0	46%	100%
RAND-BRIEF2	12	8	2	60%	86%
RANDTECH	0	16	0	0%	0%
RAND-NOTE	0	14	0	0%	0%
EAGLE-TEXT	13	0	1	100%	93%
RAND-ARROYO	0	12	0	0%	0%
SIGNATUR	0	10	0	0%	0%
RAND-ARC	1	8	0	11%	100%
RESEARCH	0	9	0	0%	0%
TOPLOG-2COL	0	9	0	0%	0%
BOTTOM-BLOCK	0	8	0	0%	0%
ATOM	0	7	0	0%	0%
CPRC	0	6	0	0%	0%
RAND-BRIEF1	2	4	0	33%	100%
RAND-LEFT	0	6	0	0%	0%
HORIZ	4	1	0	80%	100%
WARCOLLEGE	0	5	1	0%	0%

16 out of 21 of the document classes had precision under 50% and only 3 of 21 had a precision of 90% or higher.

6.3 Analysis

Examining the precision results in the preceding five sections, none of the methods by itself exceeds 80% precision for every class. The vocabulary five page (Vocab 5) method performs best with 15 of 21 classes exceeding 80%. The Vocab 5 is suited to these classes since the classes represent report type documents which have a lot of common content on pages two and three. This implementation of the MXY tree did the horizontal and vertical cuts based on the blocks generated in the OCR process. The MXY tree appears to be very susceptible to OCR segmentation differences. It performs nearly perfectly for the "au" and "eagle-text" classes which are distinguished by evenly dispersed blocks of centered text. However, it failed completely for the "rand-arroyo2" class, which is distinguished by a set of tightly spaced blocks in the middle of the page along with a header and footer block. The spacing and left-right positioning of the blocks varies between documents. This also explains the poor performance of the "Layout" and MxN classifiers. The Manhattan distance and MxN work well with documents possessing images and logos. An additional complication for our usage is that we have found a number of classes which do not select metadata from page one of the document.

CHAPTER 7

MACHINE LEARNING EXPERIMENTS

This chapter details experiments conducted is document classification using machine learning. Early in the project we were looking for a reliable method for document classification for selecting the correct template for extraction, where we defined reliability as having a precision and recall measurement better than 80%. My later experiments in machine learning were looking at classification as an additional factor to supplement the validation process in determining extraction quality. This application is more tolerant of inaccuracy since the primary mechanism is the post hoc classification process.

This section examines a series of experiments I conducted to investigate the use of machine learning techniques to conduct document classification. I used the WEKA machine learning package version 3.7.2 to conduct these experiments.

7.1 WEKA usage

The Machine Learning Group at the University of Waikato developed a machine learning toolset called Waikato Environment for Knowledge Analysis (WEKA) [71]. WEKA provides access to multiple standard machine learning techniques in an integrated environment that allow the user to quickly test a variety of techniques against any dataset. WEKA also provides advanced functionality for performing attribute selection and dimensionality reduction of data as well as the ability to chain or combine classifiers or make comparison between classifiers. It also provides a few methods for defining

training and test sets. I primarily relied on using the 10-fold cross validation method for evaluating performance [72].

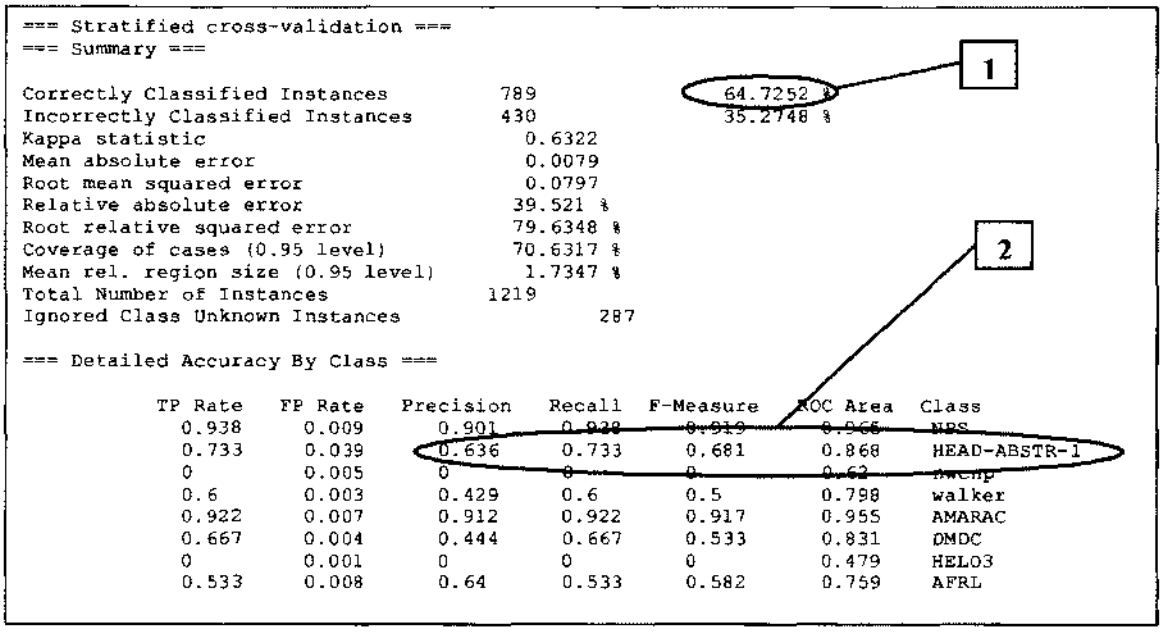


Fig. 30. Sample WEKA Output Showing Cross-validation

Fig. 30 shows sample output from the WEKA cross-validation. Using the 10-fold cross-validation, in some experiments I compared the "Correctly Classified Instances" percentage reported by the cross validation, see call out 1. Alternatively, I looked at the per class performance, call out 2. For each class I looked at Precision and Recall. Where the *Recall* is defined as the proportion of examples which were classified as class X, among all examples which truly have class X, and the *Precision* is defined as the proportion of the examples which truly have class X among all those which were classified as class X. In comparing the overall performance between classifiers, I used the *F-Measure* which is defined as:

$$F\text{-}Measure = 2*Precision*Recall/(Precision+Recall)$$

7.2 Baseline Document Collection for Experiments

I conducted each of the WEKA machine learning exeriments using the same baseline document collection. This section first describes the methodology for selecting and filtering the collection and then the process and results of the manual classification. Finally, I provide a detailed analysis of the results of the post hoc classification for the baseline collection.

7.2.1 Selecting baseline documents

I selected a set of documents to use for baseline comparisons for document classification methods. I downloaded two sets of documents from DTIC to select representative baseline documents. The first set was a random sample of 2000 documents from the 10825 documents available from DTIC during the period of February to June of 2009. The second set was a random sample of 2000 documents from

the 43468 documents available from DTIC during the period of February 2008 to February 2009. I eliminated poor quality documents consisting of older scanned documents and documents with excessive stamps or other image noise. These documents were removed because I was testing classification and not performance of the extraction engine. That left with 1675 documents most of which contained RDP forms. I manually removed the form pages from the PDF documents using **pdftk**.

7.2.2 Manual classification

I conducted a manual classification processing on the 1675 documents which resulted in 99 total classes with 310 documents remaining unclassified. Two of the classes can also be considered special cases; the "glossy" class which consists of 54 documents that are magazine type documents that can vary greatly in layout and are distinguished by large images on the first page; and the "horiz" class which consists of 44 documents that are exclusively pages which are oriented in landscape format. The "glossy" and "horiz" documents are considered unclassified for our purposes, resulting in a total of 1267 classified documents. This manual classification was used as the training data for the machine learning experiments.

The manual classification was also the starting point for developing templates as I targeted the largest classes as the first priority for template development. An integral part of the template development process is to identify minor variations within a specific class which would require development of a different template. These variations were detected by checking the extraction results of a template and then adjusting for extraction errors. A class and its variants can be grouped into a family of similar templates. The variations between some templates in a group may depend on the beginning or ending conditions of

a single field or multiple fields. As an example 13 out 25 templates in the "head-abstract" group begin with the same "UnclassifiedTitle" and "PersonalAuthor" criteria. Fig. 31 shows examples of classes in the "head-abstract" group.

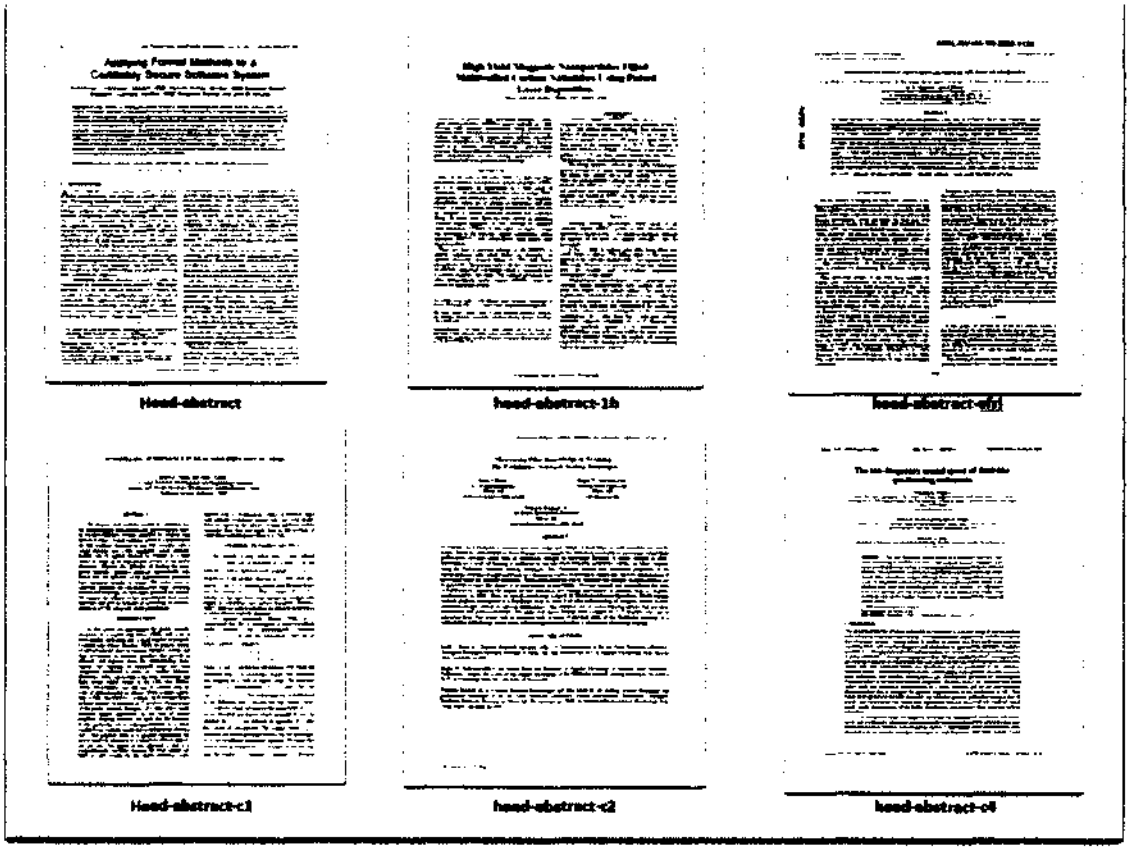


Fig. 31. Samples of "head-abstract" Class Group

7.2.3 Evaluation of Post Hoc classification

The next step was to evaluate extraction results to determine the accuracy in selecting the appropriate template. The documents were processed through the extraction system, which was configured with 152 non-form templates. The extraction resulted in

finding 1112 documents with metadata, 4 documents failed extraction, and 563 generated no metadata. I examined each extraction result on 5 common fields, title, abstract, authors, report date and report number and compared these to metadata retrieved from the DTIC online database. The template used for extraction was rated as correct, incorrect or variant. A correct rating indicates that the fields which were attempted by the template were taken from the appropriate place. Note that because I was evaluating the correctness of the selection of the template the fields may contain various typographic or OCR errors and still be marked as correct. A template variant is characterized by the majority of the fields being correct but at least one has a boundary condition error. The most common variant was where the abstract was extracted but included the words "Introduction" or "Abstract" indicating that the *begin* rule would need adjustment.

TABLE 15
Extraction Classification Results

Resulted in extraction			**1112**	
	Manually classified			1267
		Correct template	873	69%
		Variant of template	70	6%
		Incorrect template	93	7%
	Unclassfied			408
		Found a good template	42	10%
		Found a variant template	8	2%
		Incorrect template	26	6%
No extraction			**563**	
	Manually classified		325	26%
	Unclassified		234	57%
Failed during processing			**4**	

Table 15 shows the results of this evaluation, 915 / 1112 (83%) were judged to be correct and 78 / 1112 (7%) were variants. Of the manually classified documents,

873/1267 (69%) resulted in correct classifications. The system also found 42 good classifications from the previously unclassified documents. I conducted a detailed review, Table 16, of the candidate template extractions of the 119 /1112 (11%) incorrect template selections to determine if the system selected the incorrect template in preference to a correct template. There is only minimal evidence (2.5%) that the system is selecting the wrong template in preference of the correct one. 20 of the documents had problems correctly recognizing the personal author names. The template correctly identified the personal author block but the names were not extracted correctly. Improvements in the name recognition system would provide minor improvement in the overall performance. There was a large group of documents with more than 5 candidate templates which were part of the "head –abstract" type documents.

TABLE 16
Overview of Incorrect Template Characteristics

Single candidates	46	35.3%
Correct candidate not selected	3	2.5%
More than 5 candidates	46	35.3%
Name recognition problem	20	16.8%

Table 17 shows the breakdown of the incorrectly selected templates. 78/115 (67%) of the incorrect templates were from the "head-abstract" group of templates. "head-abstract-c1" and "head-abstract-c2" count for a large number of documents primarily because when multiple templates score the same, the first template in alphabetic order is selected. The "head-abstract-typewritten" class scores poorly because the documents contain many typographic errors.

TABLE 17
Breakdown of Incorrect Template Selections

Class	Documents	Avg Of Confidence
Acgsc	1	0
afrl4	4	0.49
annualReport	2	0
arl_2	1	0
crs_7	1	1
Disam	1	1
erdc2	2	0.80
erdc4	1	0.815
finalReport_3	10	0.0255
Gao	3	0.42
head-abstract	8	0.52
head-abstract-1b	7	0.29
head-abstract-AFRL	1	0
head-abstract-c1	11	0.38
head-abstract-c2	14	0.37
head-abstract-objective	1	0.04
head-abstract-physicsJournal	1	0.42
head-abstract-pttiMeeting	4	0.31
head-abstract-season	2	0.66
head-abstract-typewritten	15	0.26
head-NOabstract	13	0.20
head-NOabstract-received	1	0.58
inspectorGeneral	2	0.35
nps_proc	1	0.71
nps_report	1	0
nps_thesis3	2	0.50
NSWCCD	1	0.71
rand_arroyo_1	4	0.24
status_change_1	1	1
technicalReport_3	1	0
Usaec	2	0.5

A review of the manual classification for the 563 documents with no results showed that 89 documents in 34 classes were in classes that had templates developed.

These represent probable variants of current templates. There were also 22 classes (98 documents) for which no templates were developed.

7.3 Feature Set Construction

I used a variety of different feature sets during these experiments, the basic feature set types are described below.

7.3.1 Block Layout Signature

This method entails measuring the page similarity with a selected set of documents called the *signature set.* The features of the feature set are defined as the similarity distance measure(s) from each document in the signature set to the instance document. Thus, total number of features is equal to the number of documents in the signature set times the number of similarity measures used. I used two similarity measures for each signature document, one was the block layout distance and the other was the MxN block similarity (100x200 block grid).

I conducted a series of tests to determine the best way to select the signature documents. I initially attempted to define "orthogonal" documents to attempt to maximize separation between the signature set documents. The "orthogonal" signature set was built by iteratively adding a document from collection which has the highest average distance from the signature set. Fig. 32 shows the process for constructing the "orthogonal" set. We start with a single seed document and then find the document in the collection which has the greatest distance (or dissimilarity) from the original document. The new document is added to the signature set. We repeat the process (N-1) times until we have N documents in the signature set at each step we add the document which has

the greatest average distance from the current signature set. Fig. 33 shows a visual depiction of the blocks in a signature set size 50. Note that the set is composed of documents that have either a large number of small blocks or a small number of blocks. This demonstrates a limitation with this method of selecting the signature. As the set is built, the document added during the addition phase will oscillate between groups of sparse and dense documents (e.g. the fourth document added in Fig. 32 is sparse after adding 3 dense documents.) The essential problem with this method was that it was measuring the extremes.

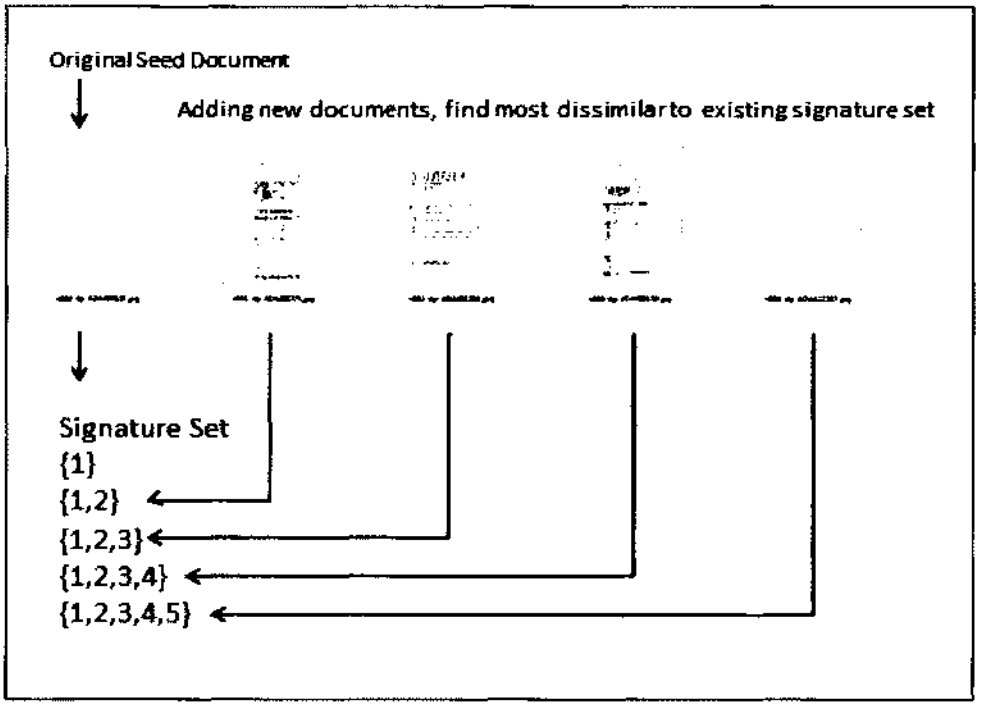


Fig. 32. Process for Building "orthogonal" Signature Set

Further experimentation showed that a better method of selecting the signature is to randomly select a large number (100-500) of signature documents and use dimensionality reduction to reduce the total number used.

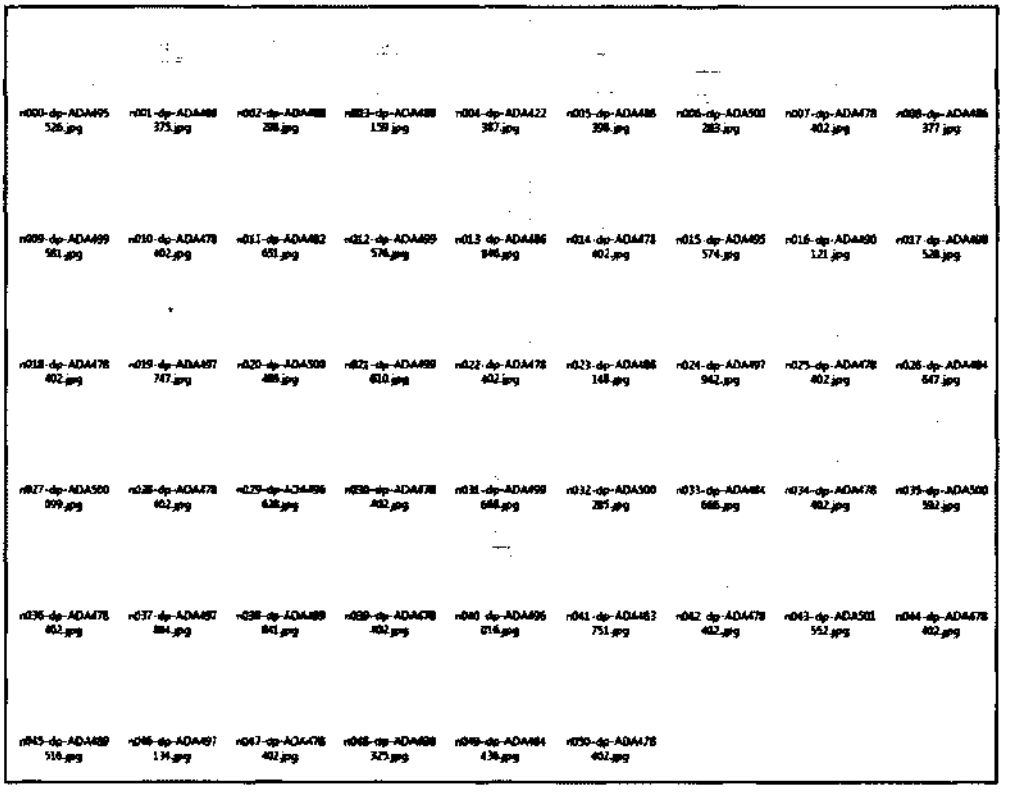


Fig. 33. Example Orthogonal Signature Set Size(N=50)

7.3.2 Document Statistics Feature Set

This feature set used the following statistical features extracted from the first page and the reduced (first 5 and last 5 pages) document. (avgFontSize, fontModePct, fontSizeMode, block, line, letter, word, digitline, digitend, avgwordline, avglinesize, capline, shortline5, shortline7, shortline9) This feature set was influenced by the

dramatic difference in fontsizes reported in textPDF pages and OCR'ed pages. The textPdf module typically reports fontsizes in the 200-400 range while OCR is 20-45. Document level measurements are influenced by the truncation of the number of pages of the original document.

7.3.3 Contextual Blocks

In order to mirror more of the features a template writer uses to produce an extraction template, I tried to define contextual blocks. Each block reports the set of features listed below:

lines: number of lines

fontsize: point value of font size reported at the box level

fontmode: most common font point size reported at the word level

modepct: percent of total words using mode size

short5: number of lines with less than 5 words

short7: number of line with less than 7 words

short9: number of lines with less than 9 words

linecase: overall case of the block (Upper, Title, Mix, Normal)

style: overall style of the block (Bold, Italic, Normal, Mix)

block position : scaled to percent of page dimensions

block area: calculated using scaled dimensions

Note: A value of "Mix" on linecase and style indicates that there are multiple lines and they do not all use the same scheme.

I conducted a series of experiments to evaluate the best mixture of blocks to use, varying the number of blocks (3, 5, 7, and 10) and combination of both beginning and

ending blocks. I also tested adding syntax features and checked the blocks for the presence of the following:

Date: found a possible date

Email: found a possible email address

Phone: found a possible phone number

Zip: found a possible zip code

7.3.4 Common Vocab

In order to generate the feature set, for each class, I found the list of common words found on page 1 of each instance document. These would represent the signature words for that particular class. I then combined all of the class level word lists into a single list and eliminated duplicates. The resulting list consisted of 1211 words shown in Appendix C. The dataset was created by checking each instance document for the presence of each word in the feature set.

7.4 Experiment: Comparing Classifiers

This experiment attempted to test as many of the classifiers available in the WEKA system against the same feature set.

7.4.1 Setup and definition

The feature set was a contextual block feature set called "fullset1" that uses the following statistical features extracted from the first page and the reduced (first 5 and last 5 pages) document: avgFontSize, fontModePct, fontSizeMode, block, line, letter, word, digitline, digitend, avgwordline, avglinesize, capline, shortline5, shortline7, and shortline9.

TABLE 18
Results Comparing Classifiers Against Single Feature Set

Classifier	%correct	Precision	Recall	F-Measure
weka.classifiers.bayes.BayesNet	73.58	0.71	0.74	0.71
weka.classifiers.lazy.KStar	73.50	0.72	0.74	0.72
weka.classifiers.meta.MultiBoostAB/J48	71.12	0.68	0.71	0.69
weka.classifiers.lazy.IB1	68.17	0.67	0.68	0.67
weka.classifiers.lazy.IBk	68.17	0.67	0.68	0.67
weka.classifiers.functions.MultilayerPerceptron	68.09	0.67	0.68	0.67
weka.classifiers.bayes.NaiveBayes	67.92	0.69	0.68	0.67
weka.classifiers.meta.Bagging/REPTree	67.84	0.62	0.68	0.64
weka.classifiers.trees.FT	67.84	0.67	0.68	0.67
weka.classifiers.rules.FURIA	65.63	0.60	0.66	0.62
weka.classifiers.trees.J48	64.73	0.63	0.65	0.64
weka.classifiers.rules.JRip	63.33	0.63	0.63	0.61
weka.classifiers.trees.LADTree	56.93	0.48	0.57	0.51
weka.classifiers.functions.SMO	49.30	0.34	0.49	0.38
weka.classifiers.lazy.LWL	22.89	0.12	0.23	0.11

7.4.2 Results

As shown in the results listed in Table 18, the BayesNet neural net classifier performed best against this particular feature set. The KStar instance-based classifier and the Boosted J48 tree-based classifier performed nearly as well as BayesNet.

7.5 Experiment: Block distance signatures

In these experiments, I investigated the block distance signature methods and the use of dimensionality reduction to improve performance.

7.5.1 Setup and definition

Data Generation: For this experiment I used four slightly different feature sets generated using the block layout signature methods. All feature sets represent the block

distances from a selected set of signature documents, the block distance measurement and the MN block similarity. The "orthogonal" feature sets are constructed using the process described in section 7.3.1. The four feature sets are:

Ortho40-noname – orthogonal feature set. The signature set has 40 documents.

Ortho50-noname – orthogonal feature set. The signature set has 50 documents.

Rand-dist-1-noname – random feature set. The signature set contains 250 randomly selected documents and the blocks used for layout are based on paragraph blocks.

Rand-dist2-1-noname – random feature set. The signature set contains 50 randomly selected documents and the blocks are based on IDM regions containing only paragraphs, paragraphs and image blocks.

Algorithm: I used the J48 and the Bayesnet classifiers in native mode as well as using an Attribute Reduction filter. The results of each filter and dataset are compared using the WEKA Paired Corrected Tester.

7.5.2 Results

Fig. 34 shows the results of this experiment. The "V" mark next to a column result indicates that the result is a statistically significant improvement over the result in column one, conversely an asterisk indicates degraded performance. The best result is found for the Bayesnet classifier using 250 random documents with attribute selection. This is a reasonable result since having 250 sample documents provides a greater range of values to use for the dimensionality reduction.

```
Tester:     weka.experiment.PairedCorrectedTTester
Analysing:  Percent_correct
Datasets:   4
Resultsets: 4
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       11/9/10 10:42 PM


Dataset                   (1) trees.J4 | (2) bayes (3) meta. (4) meta.
------------------------------------------------------------------------
ortho40-noname            (100)   54.16 |   53.21     53.64     55.85
ortho50-noname            (100)   53.88 |   53.27     53.83     56.74 v
rand-dist-1-noname        (100)   59.16 |   60.76     62.28 v   65.00 v
rand-dist2-1-noname       (100)   58.67 |   55.25 *   58.72     60.49
------------------------------------------------------------------------
                                (v/ /*) |   (0/3/1)   (1/3/0)   (2/2/0)


Key:
(1) trees.J48 '-C 0.25 -M 2' -2.17733168393644448E17
(2) bayes.BayesNet '-D -Q bayes.net.search.local.K2 -- -P 1 -S BAYES -E
bayes.net.estimate.SimpleEstimator -- -A 0.5' 7.4603744325877594E17
(3) meta.FilteredClassifier '-F \"supervised.attribute.AttributeSelection -E
\\\"CfsSubsetEval \\\" -S \\\"BestFirst -D 1 -N 5\\\"\" -W trees.J48 -- -C 0.25 -M 2' -
4.52345061853871728+18
(4) meta.FilteredClassifier '-F \"supervised.attribute.AttributeSelection -E
\\\"CfsSubsetEval \\\" -S \\\"BestFirst -D 1 -N 5\\\"\" -W bayes.BayesNet -- -D -Q
bayes.net.search.local.K2 -- -P 1 -S BAYES -E bayes.net.estimate.SimpleEstimator -- -A 0.5'
-4.52345061853871728+18
```

Fig. 34. WEKA Results Comparing Block Layout Distance Measures

7.6 Experiment: Testing multiple pages

The experiment described in section 7.2 investigated using different classifiers for a context block feature set. This experiment tests the effect of using context blocks over multiple pages.

7.6.1 Setup and definition:

Four datasets were generated using the context block feature set. The datasets represent the number of pages from 1 to 4. Each block reports the set of features listed below:

-lines: number of lines

-fontsize: point value of font size reported at the box level

-fontmode: most common font point size reported at the word level

-modepct: percent of total words using mode size

-short5: number of lines with less than 5 words

-short7: number of line with less than 7 words

-short9: number of lines with less than 9 words

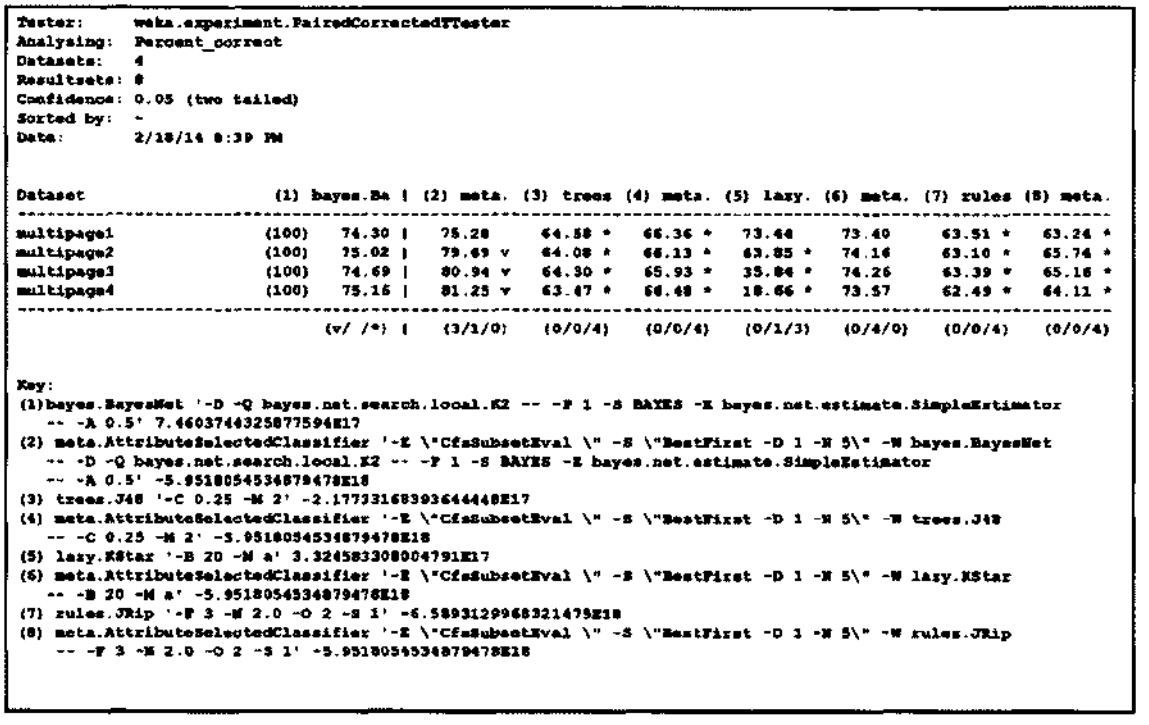
-linecase: overall case of the block (Upper, Title, Mix, Normal)

-style: overall style of the block (Bold, Italic, Normal, Mix)

-block position: x and y values

Note: "Mix" on linecase and style indicates that there are multiple lines in the block and they do not all use the same scheme.

For these experiments, the first 5 paragraphs on each of the first 4 pages are the blocks used.



```
Tester:     weka.experiment.PairedCorrectedTTester
Analysing:  Percent_correct
Datasets:   4
Resultsets: 8
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       2/18/14 8:39 PM


Dataset                   (1) bayes.Ba | (2) meta. (3) trees (4) meta. (5) lazy. (6) meta. (7) rules (8) meta.
-------------------------------------------------------------------------------------------------------------
multipage1                (100)   74.30 |   75.28     64.58 *   66.36 *   73.44     73.40     63.51 *   63.24 *
multipage2                (100)   75.02 |   79.69 v   64.08 *   66.13 *   63.85 *   74.16     63.10 *   65.74 *
multipage3                (100)   74.69 |   80.94 v   64.30 *   65.93 *   35.84 *   74.26     63.39 *   65.16 *
multipage4                (100)   75.16 |   81.25 v   63.47 *   66.48 *   18.66 *   73.57     62.49 *   64.11 *
-------------------------------------------------------------------------------------------------------------
                                (v/ /*) |   (3/1/0)   (0/0/4)   (0/0/4)   (0/1/3)   (0/4/0)   (0/0/4)   (0/0/4)


Key:
(1)bayes.BayesNet '-D -Q bayes.net.search.local.K2 -- -P 1 -S BAYES -E bayes.net.estimate.SimpleEstimator
   -- -A 0.5' 7.4603744325877594E17
(2) meta.AttributeSelectedClassifier '-E \"CfsSubsetEval \" -S \"BestFirst -D 1 -N 5\" -W bayes.BayesNet
   -- -D -Q bayes.net.search.local.K2 -- -P 1 -S BAYES -E bayes.net.estimate.SimpleEstimator
   -- -A 0.5' -5.951805453487947SE18
(3) trees.J48 '-C 0.25 -M 2' -2.1773316839364444SE17
(4) meta.AttributeSelectedClassifier '-E \"CfsSubsetEval \" -S \"BestFirst -D 1 -N 5\" -W trees.J48
   -- -C 0.25 -M 2' -5.951805453487947SE18
(5) lazy.KStar '-B 20 -M a' 3.3245833080047791E17
(6) meta.AttributeSelectedClassifier '-E \"CfsSubsetEval \" -S \"BestFirst -D 1 -N 5\" -W lazy.KStar
   -- -B 20 -M a' -5.951805453487947SE18
(7) rules.JRip '-F 3 -N 2.0 -O 2 -S 1' -6.5893129968321479E18
(8) meta.AttributeSelectedClassifier '-E \"CfsSubsetEval \" -S \"BestFirst -D 1 -N 5\" -W rules.JRip
    -- -F 3 -N 2.0 -O 2 -S 1' -5.951805453487947SE18
```

Fig. 35. Results of Adding Multiple Pages to Feature Set

7.6.2 Results

Fig. 35 shows the results of adding the additional pages to the feature set. I compared a total of eight different classifiers against the BayesNet classifier. The best performing is the BayesNet with attribute selection having 4 pages. Under the normal BayesNet, the addition of extra data pages does not improve performance. Selecting the best attributes using attribute selection results in a statistically significant improvement in performance. The BayesNet classifier with attribute selection is the only one tested that shows improvement with the addition of extra pages. The 81% correct identification rate is comparable to the performance found in post hoc classification.

7.7 Experiment: Adding syntax features

The purpose of this experiment was to test the effect of adding syntax features to the feature set by checking the blocks for the presence of Dates, Emails, Phones or Zip Codes. I compared the performance against two of the multi-page feature sets used in the experiment detailed in section 7.6

7.7.1 Setup and definition

Four feature sets were generated using the context feature blocks. Additional features are added to various sets as noted in the dataset description. Each block reports the set of features listed below:

-lines: number of lines

-fontsize: point value of font size reported at the box level

-fontmode: most common font point size reported at the word level

-modepct: percent of total words using mode size

-short5: number of lines with less than 5 words

-short7: number of line with less than 7 words

-short9: number of lines with less than 9 words

-linecase: overall case of the block (Upper, Title, Mix, Normal)

-style: overall style of the block (Bold, Italic, Normal, Mix)

-block position: x and y values

For these experiments I used Paragraphs for the block level. The blocks selected are the first 5 blocks on a page.

7.7.2 Feature sets

multipage1 - extracts the above features on only page 1

multipage4 - extracts the above features from the first 4 pages

singlepage - extracts the above feature on only page 1 and for each block adds the value for the area of the block.

Singlepage_pluslines - extracts the above feature on only page 1 and for each block adds the value for the area of the block and the syntax features.

```
Tester: weka.experiment.PairedCorrectedTTester
Analysing: Percent_correct
Datasets: 4
Resultsets: 2
Confidence: 0.05 (two tailed)


Dataset          (1)Bayesnet | (2) Meta-Bayesnet
-----------------------------------------------
multipage1             74.33 | 75.01
multipage4             75.17 | 81.21  v
singlepage             73.19 | 75.48  v
singlepage_pluslines   71.10 | 75.48  v
-----------------------------------------------
Key
bayes.BayesNet '-D -Q bayes.net.search.local.K2 -- -P 1 -S BAYES
      -E bayes.net.estimate.SimpleEstimator
      -- -A 0.5' 7.4603744325877594E17
meta.AttributeSelectedClassifier '-E \"CfsSubsetEval \" -S \"BestFirst
      -D 1 -N 5\" -W bayes.BayesNet
      -- -D -Q bayes.net.search.local.K2 -- -P 1 -S BAYES
      -E bayes.net.estimate.SimpleEstimator
      -- -A 0.5' -5.9518054534879478E18
```

Fig. 36. Results for Testing Adding Syntax Features to the Feature Set

7.7.3 Results

Fig. 36 shows the results of adding syntax features to the feature set. The best performance is found in the multiple page features set using an Attribute Selected

Bayesnet classifier. Note that the syntax features are dropped during the attribute selection and so the performance of the two singlepage feature sets becomes identical.

7.8 Experiment: Comparing performance of different feature sets

The purpose of this experiment was to compare results of different feature sets used thus far using the same classification algorithm (Bayesnet w/Attribute filtering).

7.8.1 Setup and definition

For this experiment I generated 16 separate feature sets using the four basic methods details in section 7.3 I then ran the Bayesnet Classifier with the attribute selection filter.

classif_manual_10lines - Based on context blocks for the first and last five blocks on the page.

firstlast5nblocks - Based on context blocks for the first and last five blocks on the page plus the total number of blocks on the first five pages.

fullset1 - Based on document statistics feature set plus the statistics for the first page.

multipage1 - Based on context blocks for the first five blocks on the first page.

multipage2 - Based on context blocks for the first five blocks on the first two pages.

multipage3 - Based on context blocks for the first five blocks on the first three pages.

multipage4 - Based on context blocks for the first five blocks on the first four pages.

ortho05 - Based on block layout distance without MxN blocks. The signature set includes 5 documents

ortho50 - Based on block layout distance without MxN blocks. The signature set includes 50 documents

orthodist50 - Based on block layout distance without MxN blocks. The signature set includes 50 documents using a different first seed document.

rand-dist-0 - Based on block layout distance with MxN blocks. The signature set contains 250 randomly selected documents. The blocks used for layout are based on paragraph blocks, includes MxN blocks.

rand-dist-2 - Based on block layout distance with MxN blocks. The signature set contains 250 randomly selected documents. The blocks used for layout are based on paragraph blocks, includes MxN blocks.

rand-dist2-4 - Based on block layout distance with MxN blocks. The signature set contains 250 randomly selected documents. The blocks used for layout are based on paragraph blocks, includes MxN blocks.

rand-dist-400 - Based on block layout distance with MxN blocks. The signature set contains 400 randomly selected documents. The blocks used for layout are based on paragraph blocks, includes MxN blocks..

random-mn20b - Based on block layout distance with MxN blocks. The signature set contains 20 randomly selected documents. The blocks used for layout are based on paragraph blocks, includes MxN blocks.

vocab-common - Based on the vocabulary features described above.

TABLE 19
Feature Set Comparisons Using BayesNet with Attribute Selection

Type	Feature set	%Correct
C	multipage4	81.3
C	multipage3	81.0
C	firstlast5nblocks	80.7
C	classif_manual_10lines	80.4
C	multipage2	79.9
C	multipage1	74.8
C	fullset1	73.6
D	rand-dist-400	66.9
D	rand-dist-0	66.1
D	rand-dist-2	66.0
V	vocab-common	65.4
D	rand-dist2-4	59.5
D	random-mn20b	58.9
D	orthodist50	57.9
D	ortho50	57.1
D	ortho05	43.8

7.8.2 Results

Table 19 shows the results of comparing the feature sets using the same classifier. The context blocks (labeled Type C) provide the best results and when there is more data available to the filter the classifiers function better. The block layout signatures (labeled Type D) do not perform particularly well with the Bayesnet classifier. Fig. 37 shows a comparison with multiple classifiers against the same datasets. Each of the classifiers was defined using attribute selection. With the exception of SVM, the context blocks perform better than the geometric based blocks.

```
Tester: weka.experiment.PairedCorrectedTTester
Analysing: Percent_correct
Datasets: 17
Resultsets: 7
Confidence: 0.05 (two tailed)

                                                        Decision          |
Dataset                  LibSVM  J48  NaiveBayes  Jrip   Table     Kstar  | BayesNet
                           (1)   (2)     (3)      (4)     (5)       (6)   |  (7)
------------------------------------------------------------------------- |----------
multipage4                12.2  66.5    67.9     64.1    54.8      73.6   |  81.3
multipage3                15.2  65.9    68.5     65.2    54.9      74.3   |  80.9
classif_manual_10lines    24.8  68.0    70.8     68.1    57.3      76.1   |  80.1
multipage2                16.8  66.1    68.2     65.7    54.8      74.2   |  79.7
multipage1                19.7  66.4    65.2     63.2    54.8      73.4   |  75.3
fullset1                   9.4  65.3    68.0     61.8    52.9      73.5   |  73.7
rand-dist-400             52.1  62.1    47.0     58.7    47.9      70.1   |  65.9
rand-dist-0               51.6  60.4    49.7     59.8    47.7      68.6   |  65.3
vocab-common              57.2  62.1    66.1     56.4    52.8      65.1   |  65.2
rand-dist-2               38.2  60.9    48.3     57.7    49.7      69.3   |  65.2
rand-dist2-4              56.3  58.4    38.6     54.1    45.7      66.3   |  59.0
random-mn20b              54.4  52.3    51.5     50.9    41.8      63.5   |  58.5
orthodist50               53.6  54.6    41.0     51.3    44.0      61.4   |  56.8
ortho50                   53.4  53.8    35.6     49.8    43.5      60.7   |  56.7
ortho05                   46.4  46.9    27.0     42.8    41.8      50.7   |  44.3
------------------------------------------------------------------------- |----------
Classifier Definitions:
-1  meta.FilteredClassifier '-F \"supervised.attribute.AttributeSelection -E \\\"CfsSubsetEval \\\"
       -S \\\"BestFirst -D 1 -N 5\\\"\" -W functions.LibSVM -- -S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5
       -M 40.0 -C 1.0 -E 0.0010 -P 0.1 -model D:\\libs\\weka-64\\Weka-3-7' -4.5234506185387172E18
-2  meta.FilteredClassifier '-F \"supervised.attribute.AttributeSelection -E \\\"CfsSubsetEval \\\"
       -S \\\"BestFirst -D 1 -N 5\\\"\" -W trees.J48 -- -C 0.25 -M 2' -4.5234506185387172E18
-3  meta.FilteredClassifier '-F \"supervised.attribute.AttributeSelection -E \\\"CfsSubsetEval \\\"
       -S \\\"BestFirst -D 1 -N 5\\\"\" -W bayes.NaiveBayes --' -4.5234506185387172E18
-4  meta.FilteredClassifier '-F \"supervised.attribute.AttributeSelection -E \\\"CfsSubsetEval \\\"
       -S \\\"BestFirst -D 1 -N 5\\\"\" -W rules.JRip -- -F 3 -N 2.0 -O 2 -S 1' -4.5234506185387172E18
-5  meta.FilteredClassifier '-F \"supervised.attribute.AttributeSelection -E \\\"CfsSubsetEval \\\"
       -S \\\"BestFirst -D 1 -N 5\\\"\" -W rules.DecisionTable -- -X 1 -S \"BestFirst -D 1 -N 5\"'
       -4.5234506185387172E18
-6  meta.FilteredClassifier '-F \"supervised.attribute.AttributeSelection -E \\\"CfsSubsetEval \\\"
       -S \\\"BestFirst -D 1 -N 5\\\"\" -W lazy.KStar -- -B 20 -M a' -4.5234506185387172E18
-7  meta.FilteredClassifier '-F \"supervised.attribute.AttributeSelection -E \\\"CfsSubsetEval \\\"
       -S \\\"BestFirst -D 1 -N 5\\\"\" -W bayes.BayesNet -- -D -Q bayes.net.search.local.K2 -- -P 1
       -S BAYES -E bayes.net.estimate.SimpleEstimator -- -A 0.5' -4.5234506185387172E18
```

Fig. 37. Results from Comparing Multiple Feature Sets with Multiple Classifiers

7.9 Experiment: Per class performance

The results from the previous experiments can be used to attempt to identify the best performing classifier and feature set combination. In this experiment, I attempted to evaluate the performance for each of the classes. The ideal result would be to see equivalent performance across all of the classes.

7.9.1 Setup and definition:

The dataset used is the multipage4 context block dataset identified in previous experiments. The classifier used is the Bayesnet with attribute selection. Since I wanted to try and identify the per class performance, I used 1000-fold cross-validation instead of the usual 10-fold with the goal that small classes will appear in at least two folds.

```
Scheme:     weka.classifiers.meta.AttributeSelectedClassifier -E
                weka.attributeSelection.CfsSubsetEval  -S
                weka.attributeSelection.BestFirst -D 1 -N 5 -W
                weka.classifiers.bayes.BayesNet -- -D -Q
                weka.classifiers.bayes.net.search.local.K2
                 -- -P 1 -S BAYES -E
                weka.classifiers.bayes.net.estimate.SimpleEstimator
                  -- -A 0.5

Relation:     multipage4-weka.filters.unsupervised.attribute.Remove-R1
Instances:    1588
Attributes:   261
              [list of attributes omitted]
Test mode:    1000-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances        1055      80.8429 %
Incorrectly Classified Instances      250       19.1571 %
Kappa statistic                                  0.7999
Mean absolute error                              0.0039
Root mean squared error                          0.0592
Relative absolute error                         19.8487 %
Root relative squared error                     59.7666 %
Coverage of cases (0.95 level)                  83.6782 %
Mean rel. region size (0.95 level)               1.1572 %
Total Number of Instances             1305
Ignored Class Unknown Instances        283
```

Fig. 38. Results for Bayesnet 1000-fold Cross-validation

TABLE 20
Detailed Accuracy by Class

Count	TP Rate	FP Rate	Precision	Recall	F-Measure	Class
116	**0.81**	**0.04**	**65%**	**81%**	**0.72**	**HEAD-ABSTR-1COL**
111	0.97	0.00	100%	97%	0.99	NPS
103	1.00	0.00	98%	100%	0.99	AMARAC
80	**0.87**	**0.02**	**74%**	**87%**	**0.80**	**HEAD-2COL**
77	0.73	0.00	95%	73%	0.82	CRS
64	0.98	0.00	100%	98%	0.99	AFIT
57	**0.66**	**0.02**	**65%**	**66%**	**0.66**	**head_abstr+1col_top_margin**
52	0.65	0.02	62%	65%	0.64	glossy
48	0.96	0.00	100%	96%	0.98	AWC
43	0.86	0.01	74%	86%	0.80	horiz
38	1.00	0.02	67%	100%	0.80	CRS_7
35	0.89	0.01	78%	89%	0.83	AFRL
32	1.00	0.00	100%	100%	1.00	MSMR
30	1.00	0.00	97%	100%	0.98	ARL
27	0.74	0.01	71%	74%	0.73	PAGE1-FORM
22	0.96	0.00	96%	96%	0.96	RAND
21	0.91	0.00	100%	91%	0.95	GAO
19	**0.25**	**0.01**	**27%**	**25%**	**0.26**	**head_abstr_cntr_2col**
19	**0.33**	**0.01**	**36%**	**33%**	**0.35**	**head_abstr_cntr_2col_top_margin**
16	1.00	0.00	100%	100%	1.00	INSPECTORGENERAL
14	0.64	0.01	43%	64%	0.51	diasam
14	0.93	0.00	100%	93%	0.96	ESTSC
13	1.00	0.00	100%	100%	1.00	ACGSC
13	0.77	0.00	91%	77%	0.83	COAJ
12	0.67	0.00	80%	67%	0.73	RTO
11	1.00	0.00	100%	100%	1.00	nhrc
10	1.00	0.00	100%	100%	1.00	EAGLE
10	1.00	0.01	59%	100%	0.74	nwc
10	0.40	0.00	50%	40%	0.44	RDECOM
9	1.00	0.00	90%	100%	0.95	ERDCCERL
9	0.89	0.00	67%	89%	0.76	ews
9	**0.43**	**0.00**	**60%**	**43%**	**0.50**	**head_2col_top_margin**
7	1.00	0.00	88%	100%	0.93	FAA
7	0.71	0.00	100%	71%	0.83	jsom
7	0.14	0.00	33%	14%	0.20	MORSS_1
6	1.00	0.00	100%	100%	1.00	DMDC

TABLE 20 Continued

Count	TP Rate	FP Rate	Precision	Recall	F-Measure	Class
6	1.00	0.00	86%	100%	0.92	ESTCP
5	0.60	0.00	75%	60%	0.67	coaj2
5	0.80	0.00	100%	80%	0.89	dsto
5	1.00	0.00	71%	100%	0.83	ecpc
5	**0.25**	**0.00**	**50%**	**25%**	**0.33**	**HEAD-ABSTR-3COL**
5	0.00	0.00	0%	0%	0.00	HELO3
5	1.00	0.00	83%	100%	0.91	jsou
5	1.00	0.00	63%	100%	0.77	NRL
5	1.00	0.00	83%	100%	0.91	PARAMETERS
5	1.00	0.00	100%	100%	1.00	TRAUMA
5	0.80	0.00	100%	80%	0.89	walker
Weighted Avg	0.81	0.01	79%	81%	0.79	
Classes with less than 5 instances: AFRL1, AFRL2, AFRL3_1, AFRL3_2, AFRL4, AFRL_1, ARTICLE, CARDERICKDIV, CRM, CROSSTALK, CRS_1, CRS_1B, CRS_1C, CSIREPORT1, DEPT_DEFENCE, DRDC, ENDPROJECTNUMBER, HELLO3_3, LETTERREPORT1, NAVLWARCOLLEGE, NOMETADATA, NPSRT, NSWCCD, PATENT, R-NSF, SCIENCEDIRECT, STATUSFORM, TECHREPORTNO_1, USARIEM, USARMYJOURNAL, apss, arq, au, awc-thesis, casos, checo, civileng, cp3e, defensesci, ida, ima, imast, longtermgoal, marshall, mcu, nps_arp, nswccd_2, nwcnp, sdfj, signature, whoi						

7.9.2 Results

Fig. 38 shows the overall results for the cross-validation. Table 20 shows the results for the per class performance for classes with more than 5 instance documents. As expected, the performance of the individual classes varies with the best performance being found in classes which are visually distinctive. Of particular note is the poor performance of the head-abstract report type of documents, which are highlighted in the table. Given the large variance in per class performance, I would not recommend using

this method as the primary document classification but rather as a supporting method for the post hoc classification.

7.10 Experiment: Detecting success

I originally attempted to design this experiment to examine conditions during the validation phase which would indicate the need to develop template variants. While that effort was not successful I was able to determine that the average of the validation fields will give a strong indication on the likelihood that the extracted metadata is valid.

7.10.1 Setup and definition

I added additional instrumentation code to the extraction system to monitor several new factors during the classification and validation phases of the extraction process. From the classification phase, I captured the number of candidate classes and the respective confidence scores. From the validation phase, I altered the validation script to capture the average score of fields (avg), the sum of all fields (sum), and the sum of all fields except for the average score of personal authors (avgsum).

I joined these instrumentation fields with the Baseline evaluation values (good, bad, variant) for each file to generate the following features as a machine learning dataset for Weka. The features used were:

min : original validation value

avg : average score of fields

sum : sum of all fields

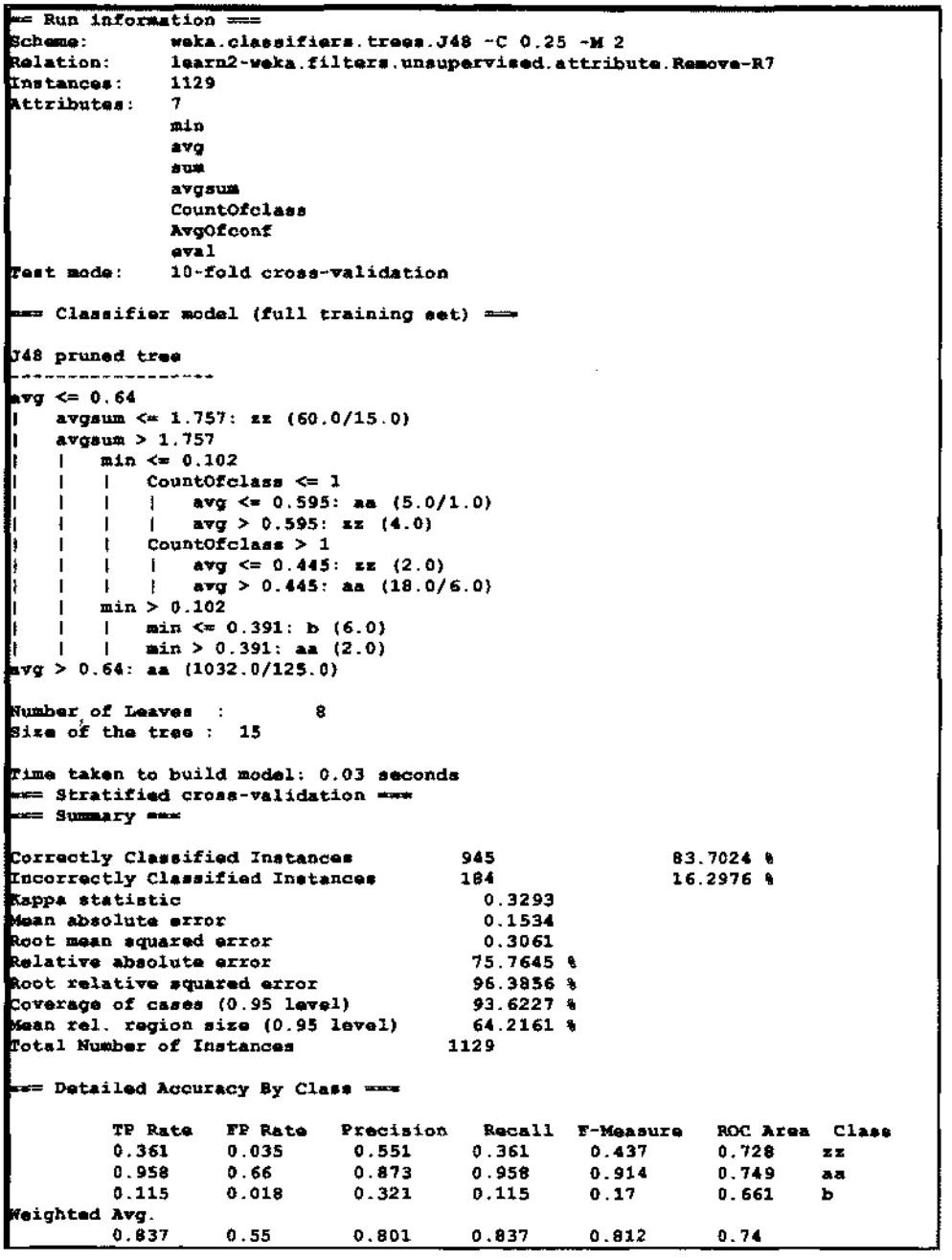
avgsum : sum of all fields plus average of personal author

countOfClass : number of classification candidate classes

avgOfConf : average of classification candidate classes scores

eval : baseline eval, good, bad, variant

The instances were labeled as "aa" for correct evals, "b" for variants, and "zz" for incorrect evals.



```
=== Run information ===
Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     learn2-weka.filters.unsupervised.attribute.Remove-R7
Instances:    1129
Attributes:   7
              min
              avg
              sum
              avgsum
              CountOfclass
              AvgOfconf
              eval
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------
avg <= 0.64
|   avgsum <= 1.757: zz (60.0/15.0)
|   avgsum > 1.757
|   |   min <= 0.102
|   |   |   CountOfclass <= 1
|   |   |   |   avg <= 0.595: aa (5.0/1.0)
|   |   |   |   avg > 0.595: zz (4.0)
|   |   |   CountOfclass > 1
|   |   |   |   avg <= 0.445: zz (2.0)
|   |   |   |   avg > 0.445: aa (18.0/6.0)
|   |   min > 0.102
|   |   |   min <= 0.391: b (6.0)
|   |   |   min > 0.391: aa (2.0)
avg > 0.64: aa (1032.0/125.0)

Number of Leaves  :       8
Size of the tree :  15

Time taken to build model: 0.03 seconds
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         945               83.7024 %
Incorrectly Classified Instances       184               16.2976 %
Kappa statistic                          0.3293
Mean absolute error                      0.1534
Root mean squared error                  0.3061
Relative absolute error                 75.7645 %
Root relative squared error             96.3856 %
Coverage of cases (0.95 level)          93.6227 %
Mean rel. region size (0.95 level)      64.2161 %
Total Number of Instances             1129

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
               0.361     0.035      0.551     0.361     0.437      0.728    zz
               0.958     0.66       0.873     0.958     0.914      0.749    aa
               0.115     0.018      0.321     0.115     0.17       0.661    b
Weighted Avg.
               0.837     0.55       0.801     0.837     0.812      0.74
```

Fig. 39. Examination of Validation and Classification Factors

7.10.2 Results

Fig. 39 shows the machine learning results for WEKA using a tree based learner (J48). The first branch in the tree is the average validation value of greater than or equal to 0.64. This particular feature is very effective at indicating that the extraction will be good. As shown in the model diagram in Fig. 39, the average attribute has more influence on the success of the output than the minimum attribute. The minimum is dominated by a single low scoring field, (which may in fact be a good extraction but not meet the validation rules), so the minimum attributes can lose information about the other fields.

This result warranted further investigation. Table 21 shows the class evaluation using the 0.64 validation average as a separation point between acceptable and unacceptable classifications. The Correct Assessment Rate is a measurement of the True Positive or True Negative assessment. The comparisons are made against the baseline evaluations for good, bad and variant classifications. The low average is also good at predicting the failures, 76% true negative rate. This would be expected since a low average is most likely by matched by a validation failure caused by the minimum being less than 0.5.

TABLE 21
Examining Effects of Using 0.64 Validation Average

	Correct	Incorrect	Correct Assessment Rate
Avg Validation > 0.64	907	126	0.88
Avg Validation < 0.64	71	23	0.76

7.11 Conclusions from experiments

I found that the BayesNet classifier combined with attribute selection provides good document classification performance using contextual features which can be simply generated from the IDM document model. As noted in the per class experimental results, the performance is not homogenous across the collection. Document classes which are visually distinctive are well suited to this method with most of the precision and recall rates in excess of 90%. However, the problems noted in the head-abstract type of document are a concern when looking at the machine learning as a replacement classification method. The head abstract type of documents often possess only minor differences between classes, such as the order of metadata fields or the presence of a keyword like "Introduction" or "Abstract". The feature set used for the BayesNet classifier does not provide features which would allow the classifier to discern those types of differences. While the post hoc classifier does much better at this type of problem, it also has difficulty with the head abstract group as shown in the analysis of the incorrect classification.

I also discovered that by adding a minor additional factor to the final validation script to calculate the average of the field confidences, we can provide the system with additional information to information the user of the need for human interaction. Specifcally, an average confidence of less than 0.64 would indicate the need for human confirmation even if no individual confidence generated a warning.

CHAPTER 8

CONTRIBUTIONS AND FUTURE WORK

8.1 Conclusions

Earlier research by the Old Dominion University Digital Library Group [4] explored the feasibility of using a template-based approach to extract metadata automatically from large heterogenous legacy collections. Each template is a script that describes a set of rules for locating desired metadata from a group of documents or class that share a similar structure. The success or failure of the template approach is directly tied to document classification which is the ability to match the document to the proper template correctly and consistently. This dissertation examined the evolution and all the major components of an automated metadata extraction system. It described the development and structure of the IDM schema, which de-couples the input from the internal representation of a document. Through our research we have developed a robust scripting language for the templates, to address a wide variety of layout complexities. The cost of the ability to handle these layout complexities is an increased complexity in writing and validating templates. I helped to reduce this cost by developing a template creation program that provides the user with immediate feedback of about a proposed template.

The primary focus of this thesis was document classification. I examined the post hoc classification methodology in detail. The strength of post hoc classification is that the templates define the contextual meaning of blocks as well as the geometric relationship relative to other blocks as well as the page. In chapter 7, I provided analysis of the

performance of the post hoc classification and it showed that when the post hoc classifier makes a decision it is correct 83% of the time. But just as significantly, the False Positive rate where it incorrectly assigns a classification to documents not represented in the template collection is only 8%. I identified the main weakness of the system is that it is sensitive to templates with poorly defined or overly general rules.

As described in Chapter 1 the objectives of this thesis were:

- Detection of problem templates;
- Reducing sensitivity to template changes;
- Document classification to aid post hoc classification;
- Description of evolution of a large software system.

This research has met these objectives. As noted above, I described evolution of our system in Chapters 3 and 4, including the major components like input processing, the IDM and field normalization. The techniques examined for handling greedy templates address not only the detection of problem templates but also methods for reducing the sensitivity to template changes. Specifically, the use of the required attribute and boosted rules are effective in reducing sensitivity.

I conducted extensive experimentation in using machine learning algorithms to conduct or supplement document classification in our system. I found that, although using the BayesNet classifier on a context block feature set drawn from multiple pages of a document provides document classification performance in excess of 80% correct, the performance can vary dramatically among different classes.

8.2 Future Work

The incorporation of machine learning to supplement post hoc classification would require more research into how it can best integrate into the system. One area not addressed directly in this thesis is the use of trained machine learning classifiers in a clustering role which could be used to identify new potential classes from unclassified documents or new incoming documents.

REFERENCES

1. Crystal, A. and P. Land, *Metadata and Search: Global Corporate Circle* in *DCMI 2003 Workshop*. 2003: Seattle WA.
2. *Bibliographic Control of Web Resources: A Library of Congress Action Plan*, L.o. Congress, Editor. 2005: Washington DC.
3. Greenberg, J., K. Spurgin, and A. Crystal, *Final Report for the AMeGA (Automatic Metadata Generation Applications) Project*. 2005.
4. Tang, J., *Template-based Metadata Extraction for Heterogeneous Collection*, in *Computer Science*. 2006, Old Dominion University: Norfolk ,VA. p. 209.
5. Han, H., et al. *Rule-based word clustering for document metadata extraction.* in *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*. 2005. ACM.
6. Han, H., et al. *Automatic document metadata extraction using support vector machines*. in *JCDL '03: Proceedings of the 3rd ACM/IEEE-CS joint conference on Digital libraries*. 2003. IEEE.
7. Seymore, K., A. McCallum, and R. Rosenfeld. *Learning Hidden Markov Model Structure for Information Extraction.* in *In AAAI 99 Workshop on Machine Learning for Information Extraction*. 1999.
8. Tang, J., et al. *Automated Building of OAI Compliant Repository from Legacy Collection*. in *ELPUB*. 2006.
9. Mao, S., J. Kim, and G. Thoma. *A Dynamic Feature Generation System for Automated Metadata Extraction in Preservation of Digital Materials*. in *DIAL '04: Proceedings of the First International Workshop on Document Image Analysis for Libraries (DIAL'04)*. 2004. IEEE.
10. Bergmark, D., *Automatic Extraction of Reference Linking Information from Online Documents*. 2000, Cornell University.
11. Klink, S., A. Dengel, and T. Kieninger. *Document Structure Analysis Based on Layout and Textual Features*. in *Proc. of International Workshop on Document Analysis Systems, DAS2000*. 2000. IAPR.
12. Marciniak, J.J., *Encyclopedia of Software Engineering*. 2002: John Wiley & Sons, Inc.
13. Debnath, S., P. Mitra, and L. Giles. *Automatic extraction of informative blocks from webpages*. in *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*. 2005. ACM.
14. Wick, M., A. Culotta, and A. McCallum. *Learning Field Compatibilities to Extract Database Records from Unstructured Text.* in *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*. 2006.
15. Sigletos, G., et al., *Mining Web sites using wrapper induction, named entities and post-processing*, in *Web Mining: FromWeb to SemanticWeb*. 2004, Springer Berlin / Heidelberg. p. 97-112.

16. Mohapatra, R., K. Rajaraman, and S.S. Yuan. *Efficient Wrapper Reinduction from Dynamic Web Sources*. in *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence* 2004. IEEE Computer Society
17. Miled, Z.B., et al., *A Wrapper Induction Application with Knowledge Base Support: A Use Case for Initiation and Maintenance of Wrappers*. Proceedings of the 5th IEEE Symposium on Bioinformatics and Bioengineering, 2005.
18. McCallum, A., *Extraction: Distilling Structured Data from Unstructured Text*, in *ACN Queue*. 2005.
19. Lerman, K., S.N. Minton, and C.A. Knoblock, *Wrapper Maintenance: A Machine Learning Approach*. Journal of Artificial Intelligence Research, 2003. **18**: p. 149-181.
20. Knoblock, C.A., et al., *Accurately and reliably extracting data from the Web: a machine learning approach*, in *Intelligent exploration of the web*. 2003, Physica-Verlag GmbH. p. 275-287.
21. Muslea, I., S. Minton, and C. Knoblock. *Wrapper Induction for Semistructured, Web-based Information Sources*. in *Proceedings of the Conference on Automatic Learning and Discovery* 1998. Pittsburgh.
22. Lerman, K., C. Knoblock, and S. Minton. *Automatic Data Extraction from Lists and Tables in Web Sources*. in *In Proceedings of the workshop on Advances in Text Extraction and Mining* 2001. Menlo Park: AAAI Press.
23. Kushmerick, N., *Wrapper induction for information extraction*. 1997, University of Washington.
24. Laender, A., B. Neto, and A. da Silva, *DEByE - Date extraction by example*. Data Knowl. Eng., 2002. **40**(2): p. 121-154.
25. Altamura, O., F. Esposito, and D. Malerba. *WISDOM++: An Interactive and Adaptive Document Analysis System*. in *ICDAR*. 1999.
26. Malerba, D., et al. *Automated Discovery of Dependencies Between Logical Components in Document Image Understanding*. in *ICDAR*. 2001. IEEE.
27. Malerba, D., F. Esposito, and O. Altamura. *Adaptive Layout Analysis of Document Images*. in *ISMIS*. 2002. Springer.
28. Aumann, Y., et al., *Visual information extraction*. Knowledge and Information Systems, 2006. **10**(1): p. 1-15.
29. Mao, S., A. Rosenfeld, and T. Kanungo. *Document structure analysis algorithms: a literature survey*. in *Document Recognition and Retrieval X*. 2003. SPIE.
30. Chen, N. and D. Blostein, *A survey of document image classification: problem statement, classifier architecture and performance evaluation*. Int'l Journal on Document Analysis and Recognition, 2007.
31. Reis, D.C., et al. *Automatic web news extraction using tree edit distance*. in *WWW '04: Proceedings of the 13th international conference on World Wide Web*. 2004. ACM.
32. Laven, K., S. Leishman, and S. Roweis. *A statistical learning approach to document image analysis*. in *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*. 2005.

33. Baldi, S., S. Marinai, and G. Soda. *Using tree-grammars for training set expansion in page classification.* in *ICDAR '03: Proceedings of the Seventh International Conference on Document Analysis and Recognition.* 2003. IEEE.
34. Shasha, D., et al. *ATreeGrep: approximate searching in unordered trees.* in *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on.* 2002.
35. Shasha, D., et al., *Exact and approximate algorithms for unordered tree matching.* Systems, Man and Cybernetics, IEEE Transactions on, 1994. **24**(4): p. 668-678.
36. Marinai, S., et al. *A general system for the retrieval of document images from digital libraries.* in *Document Image Analysis for Libraries, 2004. Proceedings. First International Workshop on.* 2004.
37. Marinai, S., E. Marino, and G. Soda. *Tree clustering for layout-based document image retrieval.* in *Document Image Analysis for Libraries, 2006. DIAL '06. Second International Conference on.* 2006.
38. Marinai, S., M. Gori, and G. Soda, *Artificial neural networks for document analysis and recognition.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2005. **27**(1): p. 23-35.
39. Marinai, S., E. Marino, and G. Soda. *Layout based document image retrieval by means of XY tree reduction.* in *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on.* 2005.
40. Cesarini, F., et al. *Encoding of modified X-Y trees for document classification.* in *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on.* 2001.
41. Cesarini, F., et al. *Structured document segmentation and representation by the modified X-Y tree.* in *Document Analysis and Recognition, 1999. ICDAR '99. Proceedings of the Fifth International Conference on.* 1999.
42. Nattee, C. and M. Numao. *Geometric Method for Document Understanding and Classification Using On-line Machine Learning.* in *International Conference on Document Analysis and Recognition.* 2001.
43. Appiani, E., et al., *Automatic document classification and indexing in high-volume applications.* International Journal on Document Analysis and Recognition, 2001. **4**(2): p. 69-83.
44. Appiani, E., et al. *"STRETCH": A System for Document Storage and Retrieval by Content.* in *DEXA '99: Proceedings of the 10th International Workshop on Database \& Expert Systems Applications.* 1999. IEEE.
45. Hu, J., R. Kashi, and G. Wilfong. *Document classification using layout analysis.* in *Database and Expert Systems Applications, 1999. Proceedings. Tenth International Workshop on.* 1999.
46. Hu, J., R. Kashi, and G. Wilfong. *Document image layout comparison and classification.* in *Document Analysis and Recognition, 1999. ICDAR '99. Proceedings of the Fifth International Conference on.* 1999.
47. Hu, J., R. Kashi, and G. Wilfong, *Comparison and Classification of Documents Based on Layout Similarity.* Inf. Retr., 2000. **2**(2-3): p. 227-243.

48. van Beusekom, J., et al. *Distance measures for layout-based document image retrieval.* in *Document Image Analysis for Libraries, 2006. DIAL '06. Second International Conference on.* 2006.
49. Le, D.X. and G.R. Thoma. *Page Layout Classification Technique for Biomedical Documents.* in *Proc. World Multiconference on Systems, Cybernetics and Informatics (SCI).* 2000.
50. Shin, C.K. and D.S. Doerrmann. *Classification of document page images based on visual similarity of layout structures.* in *In Proceedings of the SPIE Document Recognition and Retrieval VII* 2000.
51. Eglin, V. and S. Bres. *Document page similarity based on layout visual saliency: application to query by example and document classification.* in *Seventh International Conference on Document Analysis and Recognition.* 2003.
52. Briem, G.J., J.A. Benediktsson, and J.R. Sveinsson. *Use of multiple classifiers in classification of data from multiple data sources.* in *Geoscience and Remote Sensing Symposium, 2001. IGARSS '01. IEEE 2001 International.* 2001.
53. Cheeseman, P. and J. Stutz, *Bayesian classification (AutoClass): theory and results.* 1996: p. 153-180.
54. Duch, W., et al., *Competent undemocratic committees*, in *Neural Networks and Soft Computing.* 2002, Physica. p. 412-417.
55. Rahman, A., H. Alam, and M. Fairhurst, *Multiple Classifier Combination for Character Recognition: Revisiting the Majority Voting System and Its Variations*, in *Document Analysis Systems V.* 2002. p. 167-178.
56. Ting, K.M. and Z. Zheng, *A Study of AdaBoost with Naive Bayesian Classifiers: Weakness and Improvement.* Computational Intelligence, 2003. **19**(2): p. 186-200.
57. Duin, R.P.W. and D.M.J. Tax, *Experiments with Classifier Combining Rules*, in *Proceedings of the First International Workshop on Multiple Classifier Systems.* 2000, Springer-Verlag.
58. Bauer, E. and R. Kohavi, *An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants.* Mach. Learn., 1999. **36**(1-2): p. 105-139.
59. Ho, T.K., *Multiple classifier combination: lessons and next steps.* Hybrid Methods in Pattern Recognition, 2002: p. 171-198.
60. Wenzel, C., S. Baumann, and T. Jäger, *Advances in Document Classification by Voting of Competitive Approaches.* Advances in Document Analysis Systems, 1997.
61. Maly, K., S. Zeil, and M. Zubair, *Exploiting Dynamic Validation for Document Layout Classification During Metadata Extraction.* www/Internet 2007, 2007.
62. *MARCXML: Marc 21 XML Schema Official Web Site.* 2009; Available from: http://www.loc.gov/standards/marcxml///.
63. Steward, S., *pdftk --the PDF toolkit.* 2006.
64. *Apache PDFBox.* 2006, The Apache Software Foundation.
65. *Extract Program Customization for the DTIC Collection.* 2010 June 29, 2010; Available from: http://www.cs.odu.edu/~extract/developersWiki/doku.php?id=dtic:extract_program_customization_for_the_dtic_collection.

66. Digital Library Research Group, O.D.U., *Metadata Extraction Software Operation Manual Version 4.0.* 2010.
67. Zubair, M., K.J. Maly, and S. Zeil, *GPO Document Characterization and Feasibility Study of Congressional Documents*. 2008, Old Dominion University: Norfolk, VA.
68. Zubair, M., K. Maly, and S. Zeil, *GPO Document Characterization and Feasibility Study of EPA Documents*. 2008, Old Dominion University: Norfolk VA.
69. Knuth, D.E., *The Stanford GraphBase: A Platform for Combinatorial Computing*. 1994, Reading, M.A.: Addison-Wesley.
70. Nedas, K.A., *Munkres-Kuhn (Hungarian) Algorithm Clean Version: 0.11*. 2005, Department of Spatial Information Science & Engineering: Orono, ME.
71. Hall, M., et al., *The WEKA Data Mining Software: An Update*. SIGKDD Explorations, 2009. **11**(1).
72. Alpaydin, E., *Introduction to machine learning*. 2nd ed. 2010: Massachusetts Institute of Technology. 538.

APPENDIX A

IDM VERSION 2 SCHEMA

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<!--
This is the schema for the Independent Document Model (IDM) used by our
metadata extraction process.

OCR raw XML documents are converted to this schema by use of an XSLT
2.0  stylesheet
-->
<!--
The root element is <doc>  it ooptionally contains statistcis about the
enitre document under the <docInfo> element these statistics are
caclulated during the XSL transformation
-->
<xs:element name="doc">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="docInfo" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="data" type="dataType"
maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="page" type="pageType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<!--
Documents are composed of page elements

Attributes:
width - measurement of the page width
height - measurement of the page height
x-res - number of pixels per inch
y-res - number of pixels per inch
orientation - page orientation   0,90,180,270
pgno - page number
pageInfo element contains the statisticaldata for the page
-->
<xs:complexType name="pageType">
  <xs:sequence>
    <xs:element name="pageInfo" type="pageInfoType" minOccurs="0"/>
```

```
    <xs:element name="region" type="regionType" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="width" type="xs:int" use="required"/>
  <xs:attribute name="height" type="xs:int" use="required"/>
  <xs:attribute name="x-res" type="xs:int" use="required"/>
  <xs:attribute name="y-res" type="xs:int" use="required"/>
  <xs:attribute name="orientation" type="xs:string" use="optional"/>
  <xs:attribute name="pgno" type="xs:int" use="required"/>
</xs:complexType>

<xs:complexType name="pageInfoType">
  <xs:sequence>
    <xs:element name="fontinfo" type="fontinfoType" minOccurs="0"/>
    <xs:element name="parametrics" type="parametricsType"
minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<!--
Page is composed of region elements

Attributes - left, top, right, bottom - bounding box of the region
region composed of  vet-white-sapce, para, rowcol-table, image
-->
<xs:complexType name="regionType">
  <xs:sequence>
    <xs:element name="regInfo" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="data" type="dataType"
maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:choice maxOccurs="unbounded">
      <xs:element name="vert-white-space" type="vert-white-spaceType"/>
      <xs:element name="para" type="paraType"/>
      <xs:element name="rowcol-table" type="rowcol-tableType"/>
      <xs:element name="image">
        <xs:complexType>
          <xs:attribute name="l" type="xs:int"/>
          <xs:attribute name="t" type="xs:int"/>
          <xs:attribute name="r" type="xs:int"/>
          <xs:attribute name="b" type="xs:int"/>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="left" type="xs:int"/>
  <xs:attribute name="top" type="xs:int"/>
  <xs:attribute name="right" type="xs:int"/>
  <xs:attribute name="bottom" type="xs:int"/>
</xs:complexType>
<!--
para elements contain individual lines
```

```
Attributes:
align - alignment of the entire paragraph
line-spacing - space between lines
li - left indent
ri - right indent
lsp - linespaicing measurement
 l , t, b, r, -Bounding rectangle positions
-->
<xs:complexType name="paraType">
  <xs:sequence>
    <xs:element name="paraInfo" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="data" type="dataType"
maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="line" type="lineType" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="align" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="centered"/>
        <xs:enumeration value="justified"/>
        <xs:enumeration value="left"/>
        <xs:enumeration value="right"/>
        <xs:enumeration value="decimal"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="line-spacing" type="xs:int" use="optional"/>
  <!--left indent -->
  <xs:attribute name="li" type="xs:int" use="optional"/>
  <!--right indent -->
  <xs:attribute name="ri" type="xs:int" use="optional"/>
  <xs:attribute name="lsp" use="optional">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="single"/>
        <xs:enumeration value="oneAndHalf"/>
        <xs:enumeration value="double"/>
        <xs:enumeration value="exactly"/>
        <xs:enumeration value="atLeast"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <!-- Bounding rectangle positions  -->
  <xs:attribute name="l" type="xs:int"/>
  <xs:attribute name="t" type="xs:int"/>
  <xs:attribute name="r" type="xs:int"/>
  <xs:attribute name="b" type="xs:int"/>
</xs:complexType>
```

```
<!--
Statistics about font types and sizes used on a page and the document
-->
<xs:complexType name="fontinfoType">
  <xs:sequence>
    <xs:element name="fontsz" type="fontszType" minOccurs="0"
maxOccurs="unbounded"/>
    <xs:element name="fontnm" type="fontnmType" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<!--
This is the table type composed of cells.  cells contain positioning to
reconstruct the table

Attributes:
rows - number of rows to place cells into
columns - number of columns to place cells into
 l , t, b, r, -Bounding rectangle positions
-->
<xs:complexType name="rowcol-tableType">
  <xs:sequence>
    <xs:element name="cell" type="cellType" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="rows" type="xs:int" use="required"/>
  <xs:attribute name="cols" type="xs:int" use="required"/>
  <!-- Bounding rectangle positions  -->
  <xs:attribute name="l" type="xs:int"/>
  <xs:attribute name="t" type="xs:int"/>
  <xs:attribute name="r" type="xs:int"/>
  <xs:attribute name="b" type="xs:int"/>
</xs:complexType>
<!--

This is any individual cell in a table. it can span multiple rows and
columns
A cells can contain zero or more para elements or vert-white-space
elements

Attributes:
x-  left position of cell
y - top poisiotn of cell
 rowspan - number of rows this cell covers
colspan - number of cols this cell covers
 l , t, b, r, -Bounding rectangle positions
-->
<xs:complexType name="cellType">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="para" type="paraType"/>
    <xs:element name="vert-white-space" type="vert-white-spaceType"/>
  </xs:choice>
  <xs:attribute name="x" type="xs:int" use="required"/>
  <xs:attribute name="y" type="xs:int" use="required"/>
  <xs:attribute name="colspan" type="xs:int" use="required"/>
  <xs:attribute name="rowspan" type="xs:int" use="required"/>
```

```
  <!-- Bounding rectangle positions  -->
  <xs:attribute name="l" type="xs:int" use="optional"/>
  <xs:attribute name="t" type="xs:int" use="optional"/>
  <xs:attribute name="r" type="xs:int" use="optional"/>
  <xs:attribute name="b" type="xs:int" use="optional"/>
</xs:complexType>
<!--
Count of the number of words under a specified font name
-->
<xs:complexType name="fontnmType">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="wds" type="xs:int" use="required"/>
</xs:complexType>
<!--
Count of the number of words under a specified font size
-->
<xs:complexType name="fontszType">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="wds" type="xs:int" use="required"/>
</xs:complexType>
<!--
Statistics data entry
-->
<xs:complexType name="dataType">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="value" type="xs:string" use="required"/>
</xs:complexType>
<!--
line type a single line of text

Attirbutes:
ff - font face name
fs - font size in pixel measurement
style - bold, italic, non-bold etc.
 l , t, b, r, -Bounding rectangle positions
-->
<xs:complexType name="lineType">
  <xs:sequence minOccurs="0">
    <xs:element name="lineInfo" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="data" type="dataType"
maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="wd" type="wdType"/>
    </xs:choice>
  </xs:sequence>
  <!-- l , t, b, r, -Bounding rectangle positions  -->
  <xs:attribute name="l" type="xs:int"/>
  <xs:attribute name="t" type="xs:int"/>
  <xs:attribute name="r" type="xs:int"/>
  <xs:attribute name="b" type="xs:int"/>
```

```
  <xs:attribute name="ff" type="xs:string" use="optional"/>
  <xs:attribute name="fs" type="xs:string" use="optional"/>
  <xs:attribute name="style" type="xs:string" use="optional"/>
</xs:complexType>
<!--
Statistics data entry element
-->
<xs:complexType name="parametricsType">
  <xs:sequence>
    <xs:element name="data" type="dataType" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<!--
Empty whitespace inside of a region

Attributes-
t - top position
b- bottom position
pct - amount of whitespace based on percent of page measurement
loc- top or bottom of region
unit - line count or px count
v- measurenemt of size
-->
<xs:complexType name="vert-white-spaceType">
  <xs:attribute name="t" type="xs:int"/>
  <xs:attribute name="b" type="xs:int"/>
  <xs:attribute name="pct" type="xs:double"/>
  <xs:attribute name="loc">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="bottom"/>
        <xs:enumeration value="top"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="unit" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="line"/>
        <xs:enumeration value="px"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="v" type="xs:int"/>
</xs:complexType>
<!--
wd - a single word , includes trailing punctuation

Attributes:
ff - font face name
fs - font size in pixel measurement
style - bold, italic, non-bold etc.
 l , t, b, r, -Bounding rectangle positions
-->
<xs:complexType name="wdType" mixed="true">
```

```
  <!-- Bounding rectangle positions  -->
  <xs:attribute name="l" type="xs:int" use="optional"/>
  <xs:attribute name="t" type="xs:int" use="optional"/>
  <xs:attribute name="r" type="xs:int" use="optional"/>
  <xs:attribute name="b" type="xs:int" use="optional"/>
  <xs:attribute name="ff" type="xs:string" use="optional"/>
  <xs:attribute name="fs" type="xs:string" use="optional"/>
  <xs:attribute name="style" type="xs:string" use="optional"/>
</xs:complexType>
</xs:schema>
```

APPENDIX B

IDM XSLT STYLESHEET

XSLT stylesheet for converting Luratech ABBY format into IDM

```
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fn="http://www.w3.org/2004/07/xpath-functions"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:my="paul.com"
xmlns:lura="java:edu.odu.cs.extract.utils.LuratechTableUtil" exclude-
result-prefixes="fn my xs lura" xpath-default-
namespace="http://www.abbyy.com/FineReader_xml/FineReader6-schema-
v1.xml" xmlns:tns="http://www.abbyy.com/FineReader_xml/FineReader6-
schema-v1.xml">
  <!--
Supports version 2 of style sheet which requires calculation of
geometry into a normalized measurement
-->
  <xsl:output method="xml" omit-xml-declaration="no" indent="yes"/>
  <xsl:strip-space elements="doc page region para line"/>
  <!-- luratech uses dots per inch as measurement so inches per meter -
->
  <xsl:param name="dpi-meter"  select="39.3700787"></xsl:param>
<xsl:variable name="points-meter" select="2835" ></xsl:variable>
  <!-- -->
  <xsl:template match="/">
    <doc>
      <xsl:apply-templates select="//page"/>
    </doc>
  </xsl:template>
  <!-- This function will be replaced with an external java function to
calculate the column number
-->
<xsl:function name="my:getNextColNumb">
    <xsl:param name="luraObj"/>
    <xsl:param name="therow"/>
    <xsl:param name="thecol"/>
    <xsl:param name="therowspan"/>
    <xsl:param name="thecolspan"/>
    <xsl:variable name="rtn" select="$thecol+$thecolspan "
xmlns:lura="java:edu.odu.cs.extract.utils.LuratechTableUtil" use-
when="not(function-available('lura:isAvail',0))"/>
    <xsl:variable name="rtn" select="0"
xmlns:lura="java:edu.odu.cs.extract.utils.LuratechTableUtil" use-
when="function-available('lura:isAvail',0)"/>
    <xsl:value-of select="$rtn"/>
  </xsl:function>
<!--
x is dots
meter-conv is dots per meter
-->
<xsl:function name="my:scaleFS">
```

```
<xsl:param name="x" ></xsl:param>
<xsl:param name="meter-conv" ></xsl:param>
<xsl:value-of select="floor((number($x)  *number($meter-conv) div
$points-meter))"></xsl:value-of>
</xsl:function>
  <!-- -->
  <xsl:template match="page">
  <!-- meter is dots per meter -->
  <xsl:variable name="meter" select=" floor(@resolution * $dpi-meter)
"></xsl:variable>
    <xsl:element name="page">
      <xsl:copy-of select="@width"/>
      <xsl:copy-of select="@height"/>
      <xsl:attribute name="x-res" select="@resolution"/>
      <xsl:attribute name="y-res" select="@resolution"/>
      <xsl:attribute name="meter" select="$meter"></xsl:attribute>
      <xsl:attribute name="pgno"><xsl:number
count="page"/></xsl:attribute>
      <xsl:apply-templates select=".//block">
      <xsl:with-param name="meter" select="$meter"></xsl:with-param>
        <xsl:sort select="@t" order="ascending" data-type="number"/>
        <xsl:sort select="@l" data-type="number"/>
      </xsl:apply-templates>
    </xsl:element>
  </xsl:template>
  <!-- -->
  <xsl:template match="block">
  <xsl:param name="meter"></xsl:param>
    <xsl:element name="region">
      <xsl:attribute name="l" select=" @l"/>
      <xsl:attribute name="t" select=" @t"/>
      <xsl:attribute name="r" select=" @r"/>
      <xsl:attribute name="b" select=" @b"/>
      <xsl:if test="@blockType='Picture'">
        <image>
      <xsl:attribute name="l" select=" @l"/>
      <xsl:attribute name="t" select=" @t"/>
      <xsl:attribute name="r" select=" @r"/>
      <xsl:attribute name="b" select=" @b"/>
        </image>
      </xsl:if>
      <xsl:if test="@blockType='Table' or exists(row)">
        <xsl:call-template name="rowcol">
          <xsl:with-param name="meter" select="$meter"></xsl:with-
param>
          <xsl:with-param name="reg" select="."/>
        </xsl:call-template>
      </xsl:if>
      <xsl:if test="@blockType='Text'">
        <xsl:apply-templates select="text">
           <xsl:with-param name="meter" select="$meter"></xsl:with-
param>
        </xsl:apply-templates>
      </xsl:if>
    </xsl:element>
```

```
  </xsl:template>
  <!-- -->
  <xsl:template name="rowcol">
    <xsl:param name="reg"/>
  <xsl:param name="meter"></xsl:param>
    <rowcol-table rows="{count($reg/row)}" cols="{my:maxCols($reg)}">
      <xsl:attribute name="l" select=" $reg/@l"/>
      <xsl:attribute name="t" select=" $reg/@t"/>
      <xsl:attribute name="r" select=" $reg/@r"/>
      <xsl:attribute name="b" select=" $reg/@b"/>
      <xsl:variable name="luraObj" select="'text'"/>
      <xsl:for-each select="$reg/row">
        <xsl:call-template name="doRow">
          <xsl:with-param name="theluraObj" select="$luraObj"/>
          <xsl:with-param name="therow" select="."/>
          <xsl:with-param name="rownumb" select="position()-1"/>
          <xsl:with-param name="meter" select="$meter"></xsl:with-
param>
        </xsl:call-template>
      </xsl:for-each>
    </rowcol-table>
  </xsl:template>
  <!-- -->
  <xsl:function name="my:countCols">
    <xsl:param name="arow"/>
    <xsl:value-of select="sum(for $x in $arow/cell return if
(exists($x/@colSpan)) then $x/colSpan else 1)"/>
  </xsl:function>
  <!-- -->
  <xsl:function name="my:maxCols">
    <xsl:param name="atable"/>
    <xsl:value-of select="max(for $x in $atable/row return
my:countCols($x))"/>
  </xsl:function>
  <!-- -->
  <xsl:template name="doRow">
    <xsl:param name="theluraObj"/>
    <xsl:param name="therow"/>
    <xsl:param name="rownumb"/>
    <xsl:param name="meter"></xsl:param>
    <xsl:variable name="firstcol" select="0"/>
    <xsl:call-template name="doCells">
      <xsl:with-param name="theluraObj" select="$theluraObj"/>
      <xsl:with-param name="thecells" select="$therow/cell"/>
      <xsl:with-param name="rownumb" select="$rownumb"/>
      <xsl:with-param name="colnumb" select="$firstcol"/>
      <xsl:with-param name="meter" select="$meter"></xsl:with-param>
    </xsl:call-template>
  </xsl:template>
  <!-- -->
  <xsl:template name="doCells">
    <xsl:param name="theluraObj"/>
    <xsl:param name="thecells"/>
    <xsl:param name="rownumb"/>
    <xsl:param name="colnumb"/>
```

```
    <xsl:param name="meter"></xsl:param>
    <xsl:if test="exists($thecells)">
      <xsl:variable name="colspan" select="if(
exists($thecells[1]/@colSpan)) then $thecells[1]/@colSpan else 1 "/>
      <xsl:variable name="rowspan" select="if(
exists($thecells[1]/@rowSpan)) then $thecells[1]/@rowSpan else 1 "/>
      <xsl:variable name="nextcol" select="0"/>
      <xsl:call-template name="doOneCell">
        <xsl:with-param name="cell" select="$thecells[1]"/>
        <xsl:with-param name="rownumb" select="$rownumb"/>
        <xsl:with-param name="colnumb" select="$colnumb"/>
        <xsl:with-param name="meter" select="$meter"></xsl:with-param>
      </xsl:call-template>
      <xsl:call-template name="doCells">
        <xsl:with-param name="theluraObj" select="$theluraObj"/>
        <xsl:with-param name="thecells"
select="subsequence($thecells,2)"/>
        <xsl:with-param name="rownumb" select="$rownumb"/>
        <xsl:with-param name="colnumb" select="$nextcol"/>
        <xsl:with-param name="meter" select="$meter"></xsl:with-param>
      </xsl:call-template>
    </xsl:if>
  </xsl:template>
  <!-- -->
  <xsl:template name="doOneCell">
    <xsl:param name="cell"/>
    <xsl:param name="rownumb"/>
    <xsl:param name="colnumb"/>
    <xsl:param name="meter"></xsl:param>
    <xsl:element name="cell">
      <xsl:attribute name="x" select="$colnumb"/>
      <xsl:attribute name="y" select="$rownumb"/>
      <xsl:attribute name="colspan" select="if( exists($cell/@colSpan))
then $cell/@colSpan else 1 "/>
      <xsl:attribute name="rowspan" select="if( exists($cell/@rowSpan))
then $cell/@rowSpan else 1 "/>
      <xsl:if test="exists(.//line)">
        <xsl:attribute name="l" select=" min(.//line/@l)"/>
        <xsl:attribute name="t" select=" min(.//line/@t)"/>
        <xsl:attribute name="r" select=" max(.//line/@r)"/>
        <xsl:attribute name="b" select=" max(.//line/@b)"/>
      </xsl:if>
      <xsl:apply-templates select="$cell/text">
      <xsl:with-param name="meter" select="$meter"></xsl:with-param>
</xsl:apply-templates>
 </xsl:element>
</xsl:template>
  <!-- -->
  <xsl:template match="text">
  <xsl:param name="meter"></xsl:param>
    <xsl:apply-templates select="par">
    <xsl:with-param name="meter" select="$meter"></xsl:with-param>
</xsl:apply-templates>
  </xsl:template>
  <!-- -->
```

```
  <xsl:template match="par">
  <xsl:param name="meter"></xsl:param>
  <xsl:choose>
   <xsl:when test="exists(./node())">
    <para>
      <xsl:attribute name="t" select=" min(*/@t)"/>
      <xsl:attribute name="l" select=" min(*/@l)"/>
      <xsl:attribute name="r" select=" max(*/@r)"/>
      <xsl:attribute name="b" select=" max(*/@b)"/>
      <xsl:if test="exists(@leftIndent)">
        <xsl:attribute name="li" select="  @leftIndent"/>
      </xsl:if>
      <xsl:if test="exists(@rightIndent)">
        <xsl:attribute name="ri" select="  @rightIndent"/>
      </xsl:if>
      <xsl:choose>
        <xsl:when test="exists(@align)">
          <xsl:attribute name="align" select="my:alignLookup(@align)"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:attribute name="align" select="'left'"/>
        </xsl:otherwise>
      </xsl:choose>
      <xsl:if test="exists(@lineSpacing)">
        <xsl:attribute name="line-spacing" select="@lineSpacing"/>
      </xsl:if>
      <xsl:call-template name="doLines2">
        <xsl:with-param name="lines" select="line"/>
        <xsl:with-param name="meter" select="$meter"></xsl:with-param>
      </xsl:call-template>
    </para>
    </xsl:when>
    <xsl:otherwise><xsl:element
name="emptyblock"></xsl:element></xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <!-- -->
  <xsl:function name="my:alignLookup">
    <xsl:param name="a"/>
    <xsl:value-of select=" if($a='Center') then 'centered' else
if($a='Left') then 'left' else if($a='Right') then 'right' else
if($a='Justified') then 'justified' else 'left'  "/>
  </xsl:function>
  <!-- -->
  <xsl:template name="doLines2">
    <xsl:param name="lines"/>
  <xsl:param name="meter"></xsl:param>
    <xsl:choose>
      <xsl:when test="exists($lines)">
        <xsl:variable name="openface" select="if
(empty($lines[1]/formatting[1]/@ff)  ) then ()   else
$lines[1]/formatting[1]/@ff "/>
        <xsl:variable name="openfs" select="if
(empty($lines[1]/formatting[1]/@fs)  ) then ()   else
$lines[1]/formatting[1]/@fs "/>
```

```
        <xsl:variable name="pos" as="xs:integer">
          <xsl:call-template name="findLineMatch">
            <xsl:with-param name="lines"
select="subsequence($lines,2)"/>
            <xsl:with-param name="face" select="$openface"/>
            <xsl:with-param name="fontsize" select="$openfs"/>
            <xsl:with-param name="pos" select="1"/>
            <xsl:with-param name="meter" select="$meter"></xsl:with-
param>
          </xsl:call-template>
        </xsl:variable>
        <xsl:call-template name="doStyleLines">
          <xsl:with-param name="lines" select="$lines[position() le
$pos]"/>
          <xsl:with-param name="ff" select="$openface"/>
          <xsl:with-param name="fs" select="$openfs"/>
          <xsl:with-param name="meter" select="$meter"></xsl:with-
param>
        </xsl:call-template>
        <xsl:call-template name="doLines2">
          <xsl:with-param name="lines"
select="subsequence($lines,$pos+1)"/>
          <xsl:with-param name="meter" select="$meter"></xsl:with-
param>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <!-- -->
  <xsl:template name="doStyleLines">
    <xsl:param name="lines"/>
    <xsl:param name="ff"/>
    <xsl:param name="fs"/>
    <xsl:param name="meter"></xsl:param>
    <xsl:choose>
      <xsl:when test="exists($lines)">
        <xsl:variable name="openstyle"
select="my:style($lines[1]/formatting) "/>
        <xsl:variable name="pos" as="xs:integer">
          <xsl:call-template name="findStyleMatch">
            <xsl:with-param name="lines"
select="subsequence($lines,2)"/>
            <xsl:with-param name="attr" select="$openstyle"/>
            <xsl:with-param name="pos" select="1"/>
            <xsl:with-param name="meter" select="$meter"></xsl:with-
param>
          </xsl:call-template>
        </xsl:variable>
        <xsl:choose>
          <xsl:when test="exists($openstyle)">
            <xsl:apply-templates select="$lines[position() le $pos]">
              <xsl:with-param name="ff" select="$ff"/>
              <xsl:with-param name="fs" select="$fs"/>
```

```
              <xsl:with-param name="style" select="$openstyle"/>
              <xsl:with-param name="meter" select="$meter"></xsl:with-
param>
            </xsl:apply-templates>
          </xsl:when>
          <xsl:otherwise>
            <xsl:message>style2=<xsl:value-of select="$openstyle">
</xsl:value-of>
            </xsl:message>
            <xsl:apply-templates select="$lines[position() le $pos]">
              <xsl:with-param name="ff" select="$ff"/>
              <xsl:with-param name="fs" select="$fs"/>
              <xsl:with-param name="meter" select="$meter"></xsl:with-
param>
            </xsl:apply-templates>
          </xsl:otherwise>
        </xsl:choose>
        <xsl:call-template name="doStyleLines">
          <xsl:with-param name="lines"
select="subsequence($lines,$pos+1)"/>
          <xsl:with-param name="ff" select="$ff"/>
          <xsl:with-param name="fs" select="$fs"/>
          <xsl:with-param name="meter" select="$meter"></xsl:with-
param>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <!-- -->
  <xsl:template name="findLineMatch">
    <xsl:param name="lines"/>
    <xsl:param name="face"/>
    <xsl:param name="fontsize"/>
    <xsl:param name="pos"/>
    <xsl:param name="meter"></xsl:param>
    <!-- find first matching fonts and size -->
    <xsl:choose>
      <xsl:when test="exists($lines)">
        <xsl:choose>
          <xsl:when test="($lines[1]/formatting/@ff = $face) and
($lines[1]/formatting/@fs = $fontsize) ">
            <xsl:call-template name="findLineMatch">
              <xsl:with-param name="lines"
select="subsequence($lines,2)"/>
              <xsl:with-param name="face" select="$face"/>
              <xsl:with-param name="fontsize" select="$fontsize"/>
              <xsl:with-param name="pos" select="$pos +1"/>
              <xsl:with-param name="meter" select="$meter"></xsl:with-
param>
            </xsl:call-template>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="$pos"/>
```

```
          </xsl:otherwise>
        </xsl:choose>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$pos"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <!-- -->
  <xsl:template name="findStyleMatch">
    <xsl:param name="lines"/>
    <xsl:param name="attr"/>
    <xsl:param name="pos"/>
    <xsl:param name="meter"></xsl:param>
    <!-- find first matching style -->
    <xsl:choose>
      <xsl:when test="exists($lines)">
        <xsl:choose>
          <xsl:when test="(my:style($lines[1]/formatting) = $attr)  ">
            <xsl:call-template name="findStyleMatch">
              <xsl:with-param name="lines"
select="subsequence($lines,2)"/>
              <xsl:with-param name="attr" select="$attr"/>
              <xsl:with-param name="pos" select="$pos +1"/>
              <xsl:with-param name="meter" select="$meter"></xsl:with-
param>
            </xsl:call-template>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="$pos"/>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$pos"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <!-- -->
  <xsl:template match="line">
    <xsl:param name="ff" as="xs:string"/>
    <xsl:param name="fs" as="xs:string"/>
    <xsl:param name="style"/>
    <xsl:param name="meter"></xsl:param>
    <xsl:element name="line">
      <xsl:attribute name="l" select=" @l"/>
      <xsl:attribute name="t" select=" @t"/>
      <xsl:attribute name="r" select=" @r"/>
      <xsl:attribute name="b" select=" @b"/>
      <xsl:attribute name="base" select=" @baseline"/>
      <xsl:if test="exists($ff) and string-length($ff) gt 0">
        <xsl:attribute name="ff" select="$ff"/>
      </xsl:if>
      <xsl:if test="exists($fs) and string-length($fs) gt 0">
```

```
      <xsl:variable name="currfs" select="(if (contains($fs, '.')) then
substring-before($fs,'.') else $fs)"></xsl:variable>
        <!-- truncate font point sizes -->
        <xsl:attribute name="fs"  select="my:scaleFS($currfs,$meter)"/>
      </xsl:if>
      <xsl:if test="exists($style) and string-length($style) gt 0">
        <xsl:attribute name="style" select="$style"/>
      </xsl:if>
      <xsl:apply-templates select="formatting" mode="split">
        <xsl:with-param name="face" select="$ff"/>
        <xsl:with-param name="fontsize" select="$fs"/>
        <xsl:with-param name="style" select="$style"/>
          <xsl:with-param name="l" select="@l"/>
          <xsl:with-param name="t" select="@t"/>
          <xsl:with-param name="r" select="@r"/>
          <xsl:with-param name="b" select="@b"/>
          <xsl:with-param name="meter" select="$meter"></xsl:with-
param>
      </xsl:apply-templates>
    </xsl:element>
  </xsl:template>

  <xsl:template match="formatting" mode="split">
    <xsl:param name="face"/>
    <xsl:param name="fontsize"/>
    <xsl:param name="style"/>
    <xsl:param name="l"/>
    <xsl:param name="t"/>
    <xsl:param name="r"/>
    <xsl:param name="b"/>
    <xsl:param name="meter"></xsl:param>
    <xsl:choose>
      <xsl:when test="exists(charParams)">
        <xsl:call-template name="assembleWd">
          <xsl:with-param name="charSeq"
select="subsequence(charParams,2)"/>
          <xsl:with-param name="wd" select="charParams[1]"/>
          <xsl:with-param name="l" select="charParams[1]/@l"/>
          <xsl:with-param name="t" select="charParams[1]/@t"/>
          <xsl:with-param name="r" select="charParams[1]/@r"/>
          <xsl:with-param name="b" select="charParams[1]/@b"/>
          <xsl:with-param name="meter" select="$meter"></xsl:with-
param>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
      <xsl:for-each select="tokenize(., '\s')">
        <wd ><xsl:value-of select="normalize-space(.)"/>
        </wd>
        </xsl:for-each>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <!-- -->
  <xsl:template name="assembleWd">
```

```
    <xsl:param name="charSeq"/>
    <xsl:param name="wd"/>
    <xsl:param name="l"/>
    <xsl:param name="t"/>
    <xsl:param name="r"/>
    <xsl:param name="b"/>
    <xsl:param name="meter"></xsl:param>
    <xsl:choose>
      <xsl:when test="exists($charSeq)">
        <xsl:choose>
          <xsl:when test="$charSeq[1]/@wordStart='true' ">
            <wd t="{$t}" l="{$l}" r="{$r}" b="{$b}"><xsl:value-of
select="normalize-space($wd)"/>
            </wd>
            <xsl:call-template name="assembleWd">
              <xsl:with-param name="charSeq"
select="subsequence($charSeq,2)"/>
              <xsl:with-param name="wd" select="$charSeq[1]"/>
              <xsl:with-param name="l" select="$charSeq[1]/@l"/>
              <xsl:with-param name="t" select="$charSeq[1]/@t"/>
              <xsl:with-param name="r" select="$charSeq[1]/@r"/>
              <xsl:with-param name="b" select="$charSeq[1]/@b"/>
              <xsl:with-param name="meter" select="$meter"></xsl:with-
param>
            </xsl:call-template>
          </xsl:when>
          <xsl:otherwise>
            <xsl:call-template name="assembleWd">
              <xsl:with-param name="charSeq"
select="subsequence($charSeq,2)"/>
              <xsl:with-param name="wd" select="concat($wd,
$charSeq[1])"/>
              <xsl:with-param name="l" select="min(($l,
$charSeq[1]/@l))"/>
              <xsl:with-param name="t" select="min(($t,
$charSeq[1]/@t))"/>
              <xsl:with-param name="r" select="max(($r,
$charSeq[1]/@r))"/>
              <xsl:with-param name="b" select="max(($b,
$charSeq[1]/@b))"/>
              <xsl:with-param name="meter" select="$meter"></xsl:with-
param>
            </xsl:call-template>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:when>
      <xsl:otherwise>
        <wd t="{ $t}" l="{ $l}" r="{ $r}" b="{ $b}"><xsl:value-of
select="normalize-space($wd)"/>
        </wd>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <!-- -->
  <xsl:function name="my:style">
```

```
    <xsl:param name="format"/>
    <xsl:variable name="temp">
      <xsl:if test="$format/@bold  ">
        <xsl:value-of select="' bold'"/>
      </xsl:if>
      <xsl:if test="$format/@italic">
        <xsl:value-of select="' italic'"/>
      </xsl:if>

<xsl:if test="$format/@subscript"><xsl:value-of select="'
subscript'"></xsl:value-of></xsl:if>
<xsl:if test="$format/@superscript"><xsl:value-of select="'
superscript'"></xsl:value-of></xsl:if>
<xsl:if test="$format/@smallcaps"><xsl:value-of select="'
smallcaps'"></xsl:value-of></xsl:if>
<xsl:if test="$format/@underline"><xsl:value-of select="'
underline'"></xsl:value-of></xsl:if>
<xsl:if test="$format/@strikeout"><xsl:value-of select="'
strikeout'"></xsl:value-of></xsl:if>
    </xsl:variable>
    <xsl:value-of select="if (string-length(my:trim($temp))=0) then  ()
else my:trim($temp) "/>
  </xsl:function>
  <!-- -->
  <xsl:function name="my:capline">
    <xsl:param name="line"/>
  </xsl:function>
  <!-- -->
  <xsl:function name="my:trim">
    <xsl:param name="str"/>
    <xsl:variable name="s" select="my:ltrim($str)"/>
    <xsl:sequence select="my:rtrim($s)"/>
  </xsl:function>
  <!-- -->
  <xsl:function name="my:ltrim">
    <xsl:param name="str"/>
    <xsl:value-of select="replace($str,'^\s+', '')"/>
  </xsl:function>
  <!-- -->
  <xsl:function name="my:rtrim">
    <xsl:param name="str"/>
    <xsl:value-of select="replace($str, '\s+$' , '')"/>
  </xsl:function>
  <!-- -->
  <xsl:function name="my:stddev">
    <xsl:param name="nodes"/>
    <xsl:choose>
      <xsl:when test="empty($nodes)">
        <xsl:value-of select="0"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:variable name="mean" select="avg($nodes)"/>
        <xsl:variable name="meandif2" select="for $x in $nodes return
($mean - $x)*($mean - $x) "/>
        <xsl:variable name="summeandif" select="sum($meandif2)"/>
```

```
        <xsl:value-of select="my:sqrt(($summeandif div (count($nodes) -
1)))"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:function>
  <!-- -->
  <xsl:function name="my:sqrt">
    <!-- The number you want to find the square root of -->
    <xsl:param name="numb"/>
    <xsl:variable name="number" select="if (empty($numb) or $numb le 0)
then 0 else $numb "/>
    <!-- The current 'try'.  This is used internally. -->
    <xsl:variable name="try" select="number($number &lt; 100) +
                  number($number >= 100 and $number &lt; 1000) * 10 +
                  number($number >= 1000 and $number &lt; 10000) * 31
+
                  number($number >= 10000) * 100"/>
    <!-- The current iteration, checked against maxiter to limit loop
count -->
    <xsl:variable name="iter" select="1"/>
    <!-- Set this up to ensure against infinite loops -->
    <xsl:variable name="maxiter" select="10"/>
    <!-- This template was written by Nate Austin using Sir Isaac
Newton's
      method of finding roots -->
    <xsl:value-of select="my:sqrt2($number,  $try,  $iter, $maxiter)"/>
  </xsl:function>
  <!-- -->
  <xsl:function name="my:sqrt2">
    <!-- The number you want to find the square root of -->
    <xsl:param name="number"/>
    <!-- The current 'try'.  This is used internally. -->
    <xsl:param name="try"/>
    <!-- The current iteration, checked against maxiter to limit loop
count -->
    <xsl:param name="iter"/>
    <!-- Set this up to ensure against infinite loops -->
    <xsl:param name="maxiter"/>
    <!-- This template was written by Nate Austin using Sir Isaac
Newton's
      method of finding roots -->
    <xsl:choose>
      <xsl:when test="$try * $try = $number or $iter > $maxiter">
        <xsl:value-of select="$try"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="my:sqrt2($number,
         $try - (($try * $try - $number) div (2 * $try)),
         $iter + 1,
         $maxiter)"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:function>
</xsl:stylesheet>
```

APPENDIX C

COMMON VOCABULARY WORD LIST

word	instructors	maximize	ryan	express
president	asked	case	machining	decision
encompasses	lindsey	ltd	hussein's	sayyid
ralph	casos	powers	institution	acquisition
managing	monitoring	lavian	top	fullerton
ethane	engine	deconflict	conflicts	accomplishments
participants	number	australian	defense	tests
rochester	keystroke	aberdeen	crucial	unless
academic	requested	oceanography	macedo	trademarks
lambda	independently	child	provides	hile
time	origins	jsou	demonstration	construction
materials	warranted	subscription	variety	note
deputy	wrightpatterson	riemenschneider	needed	hole
florida	air	sampling	jordanian	held
declared	yemeni	performance	school	seventeenth
movement	view	multiple	politicization	gaines
advanced	introduction	placed	views	sciences
systematic	computer	interactions	syracuse	ethical
substantial	ocean	profile	bozeman	officials
intelligence	embraces	crs	service	oswald
these	attempted	materiel	subsystem	retains
marie	modification	regular	provided	reserved
millington	division	counter	hand"	studying
newport	feasibility	norms	carnegie	anomaly
sheryl	could	community	operations	master
partial	minneapolis	you	our	hurlburt
reorganization	intervention	adult	out	reasoning
overton	debate	sacred	technoloqy	improvement
evolution	domain	soon	muslim	capabilities
few	current	defects	tal	force
conditions	barnes	general	diaz	durip
project	logistics	sensors	technical	mortimer
radnor	materially	representative	maintaining	manufacturing
wwwncdmmorg	vernon	furnished	planners	pla
biological	azzam	marcia	city	palestinian
saudis	embassy	sons	center	entered
lisowski	compliment	pre	course	where

quality	lieutenant	authentication	open	type
opening	web	registered	authors	reproduction
codes	whatsoever	prohibited	simultaneously	ffoorr
bin	canal	infrastructures	aviv	stockton
attacks	represent	csi	page	surveillance
faculty	paula	bethesda	initial	experiment
array	history	very	vehicles	primary
enemies	lundgren	equipment's	groups	request
knowledge	background	doors	artillery	action
communications	modelbased	trapezoid	wwwnprstnavymil	useful
incorporating	volume	qaeda	islamist	kentkendall
dependent	detrick	automated	information	moteff
prospect	methods	surface	others	applications
john	quantico	house	form	medicine
made	abroad	monograph	national	required
mathematics	award	people	art	cycle
training	norman	corporation	fort	joint
authority	dear	nps	table	occupation
rostow	transportation	accepted	director	workshop
patient	completion	drive	sawka	maintainer's
concerned	conduct	shotgun	shall	contract
experiments	affairs	focus	cite	scroll
turbulent	any	brewton	soldier	philosophy
foundation	ann	industry	inputs	designated
elizabeth	light	grove	dissertation	by[montana
experts	key	million	david	sectors
airborne	claims	contributors	dedicating	ews
increasing	keywords	charitable	position	embedded
taking	packages	elk	noncommercial	cooper
electronic	informa	operational	budget	paul
labor	afghanistan	many	myatt	whole
expertise	likely	not	shama	proceedings
new	begins	logical	institutions	leslie
including	forces	san	ship	dougherty
days	interim	covert	traces	chairman
library	prognostic	now	dale	security
base	oral	start	muhammad	defensive
testing	contemporary	satisfaction	separated	requirements
garner	example	years	ground	automatically
resources	reprints	warranty	king	ideology
physical	wall	some	contact	grid
congress	does	intregration	commandant	east

payoff	set	link	costs	drug
root	department	purposes	effective	hershkowitz
change	combatting	line	private	building
pdf	milestone	lind	title	fichtner
kimscna	carlisle	believed	accurate	political
tool	met	indirectly	laden's	reuse
lead	organization	excellence	montana	buildup
architecture	england	weber	command	task
qutb	area	estcp	expressly	agreement
policies	defence	district	commandand	problem
terrorists	members	just	thomas	doctoral
founded	kuwaiti	authorized	aging	environmental
software	ledlow	sseeccuurriittyy	united	manufacturinq
medical	diego	conquest	rely	goodman
personnel	asiapacific	process	desert	implementation
walters	alison	encrypted	delays	advisor
component	sea	lee	signed	accrue
updated	still	degree	claus	navy
invasion	see	oldfield	access	also
derived	mission	injury	should	forbidden
overall	michael	brother	research	energy
motion	ida	usaf	galie	changes
had	numerical	preprint	secretary	microbiology
primarily	william	govt	valuable	berg
jihad	daniel	phone	stinfo	professionals
fax	instructor	restrictions	industrial	limited
tekcom	kursk	herein	article	appear
fulfillment	street	negroponte	cirderock	itr
vemork	rights	villanova	fall	unclassified
indicating	iii	used	walker	maintain
code	nicira	heat	appendix	ambiguity
rising	operation	concerns	based	flow
hussein	abuse	been	wander	countries
names	waging	sources	muslims	scripps
revised	correct	reproduce	grant	feedback
guillermo	verified	sent	solutions	practices
supplemental	minimizing	sherman	tel	movements
opinions	only	llc	little	year
spar	central	sapiro	impact	secondly
recruiting	commercial	ablaze	requests	jorge
hamas	branches	kelly	tools	contents
ideologue	structure	abdullah	were	california

make	monterey	algorithms	please	indicated
guidance	church	robert	camphouse	log
justice	ecological	specializes	compromise	colonel
measure	maybe	glantz	martin	system
msmr	lippincott	connection	cooling	weakened
seabasing	submitted	intellectual	notoriously	leavenworth
analysis	clift	agencies	maryland	referred
education	questions	enterprise	power	denis
business	board	mark	combat	press
response	monthly	permission	reduction	complex
author	dzakowic	heimbuch	university]	territory
qaeda's	rulebases	limitation	directly	monica
findings	kathleen	jump	korea	oklahoma
possible	sof	toolkits	ave	aassiiaappaacciiffiicc
especially	islam	presentation	administration's	glauser
name	committee	representation	systems	use
statements	postgraduate	policy	work	states
nswc	attn	suite	expressed	downloaded
asia	delay	mary	brainstorming	difficult
infection	comment	russell	class	usa
implied	evaluations	administration	pacific	graduation
population	gao	increased	property	cni
law	warfare	paper	jeddah	substance
buchman	region	issue	market	americas
trauma	expert	virus	environment	life
cable	series	state	gainesville	significant
congressional	scott	committees	isolating	cna
trafficking	experiences	outlining	program	site
layer	development	another	company	reselling
respectfully	mosquito	drdc	jerome	documents
laboratory	rohan	details	wave	liable
baghdad	subject	full	management	tennessee
mcdevitt	most	leader	prepared	pennsylvania
usawc	usariem	southeast	west	wwwcrsgov
aziz	publisher	construed	other	seen
expected	remaining	exist	davis	material
which	mavsea	corps	intention	works
rome	appropriation	jolla	interests	michigan
relations	closed	availability	sunni	directorate
take	redistribution	saddam	contained	south
anyone	organisation	university	vvoolluummee	down
box	pages	beginning	routines	nonmuslim

make	monterey	algorithms	please	indicated
guidance	church	robert	camphouse	log
justice	ecological	specializes	compromise	colonel
measure	maybe	glantz	martin	system
msmr	lippincott	connection	cooling	weakened
seabasing	submitted	intellectual	notoriously	leavenworth
analysis	clift	agencies	maryland	referred
education	questions	enterprise	power	denis
business	board	mark	combat	press
response	monthly	permission	reduction	complex
author	dzakowic	heimbuch	university]	territory
qaeda's	rulebases	limitation	directly	monica
findings	kathleen	jump	korea	oklahoma
possible	sof	toolkits	ave	aassiiaappaacciiffiicc
especially	islam	presentation	administration's	glauser
name	committee	representation	systems	use
statements	postgraduate	policy	work	states
nswc	attn	suite	expressed	downloaded
asia	delay	mary	brainstorming	difficult
infection	comment	russell	class	usa
implied	evaluations	administration	pacific	graduation
population	gao	increased	property	cni
law	warfare	paper	jeddah	substance
buchman	region	issue	market	americas
trauma	expert	virus	environment	life
cable	series	state	gainesville	significant
congressional	scott	committees	isolating	cna
trafficking	experiences	outlining	program	site
layer	development	another	company	reselling
respectfully	mosquito	drdc	jerome	documents
laboratory	rohan	details	wave	liable
baghdad	subject	full	management	tennessee
mcdevitt	most	leader	prepared	pennsylvania
usawc	usariem	southeast	west	wwwcrsgov
aziz	publisher	construed	other	seen
expected	remaining	exist	davis	material
which	mavsea	corps	intention	works
rome	appropriation	jolla	interests	michigan
relations	closed	availability	sunni	directorate
take	redistribution	saddam	contained	south
anyone	organisation	university	vvoolluummee	down
box	pages	beginning	routines	nonmuslim

assessment	developments	funding	recent	reflect
terrorism	appearing	leadership	coast	world
thermal	barry	ultimately	morss	realizing
formulae	teaching	potential	vol	emergency
layers	additional	came	protected	taylor
simulationbased	item	gates	order	perform
relived	attached	addition	officer	identified
brotherhood	kansas	egypt	vector	science
sensor	accuracy	enacted	monument	appropriations
katzman	design	"boxes"	york	necessarily
minnesota	described	address	unrestricted	architectures
marlyn	technology	sacramentoyolo	deane	robin
recently	second	seoul	find	adopted
report	individual	reaerosolization	epstein	randy
phm	wales	facing	office	care
rand	being	special	aerospace	surfaced
lancaster	integrity	benefits	marine	bush
nations	investigator	group	jinhwa	presented
itself	study	first	directing	demand
diagnostic	computing	user	rice	control
final	accountability	need	matter	condition
reported	studies	predicting	naturally	manageable
transfer	middle	cadarette	andino	diplomatic
support	procedures	mountain	origin	evolving
kent	military	benefit	precipitate	preliminary
numerous	prior	threat	proving	tactics
health	caused	tyndall	howsoever	cost
prone	sustainment	college	woods	moments
damages	more	higher	berkeley	economic
armys	level	visit	publication	url
publications	eminent	mellon	conclusions	thesis
spawned	inc	storm	technologies	failure
decisions	results	principal	safety	joseph
airbase	gulf	brou	term	cast
will	official	militant	applied	ncdmm
branch	faults	natick	learns	email
scharschan	protection	endorsed	largely	several
columbia	insecticides	chemical	continue	barracks
broad	challenges	contribution	claimed	london
abdul	appropriate	government	anne	floor
schleier	nnuummbbeerr	wwwrandorg	iraq	oceanographic
objective	assertions	impingers	filling	own

such
wilkins
mors
guarding
lantos
studied
copies
plethora
sympathizers
eastern
included
causes
carley
[subscription
concept
jack
loss
mindset
arabia
becomes
alabama
improve
includes
war
was
online
oceangoing
army
risk
real
naval
manifests
staff
selected
santa
blvd
shapiro
driven

conservative
student
browse
gunaratna
soviet
arising
taken
conference
timing
networks
edmond
however
associates
maintenance
frederick
effort
providing
reduce
architect
james
assets
surgical
efforts
foreign
natural
doses
model
trade
land
saudi
well
valentine
engineering
appears
rokusn
journal
mitigation

data

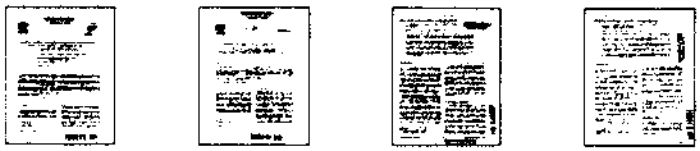
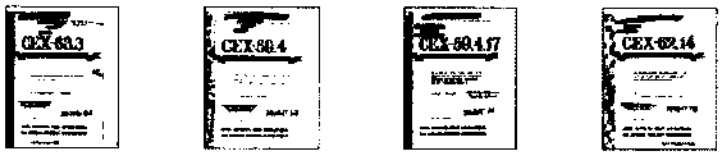
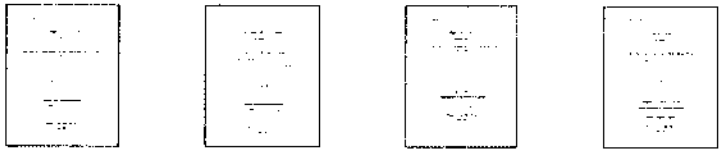
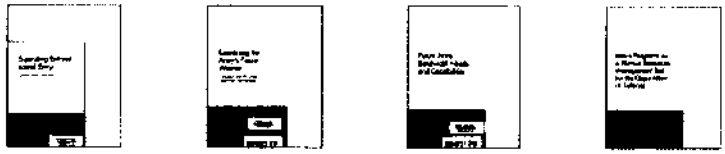
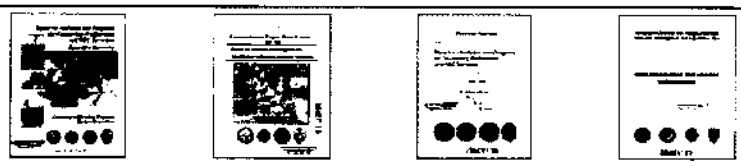
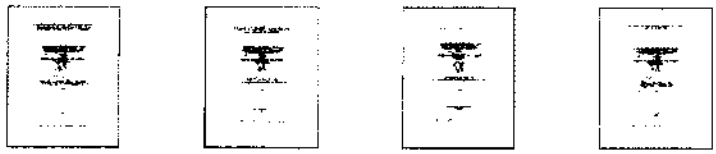
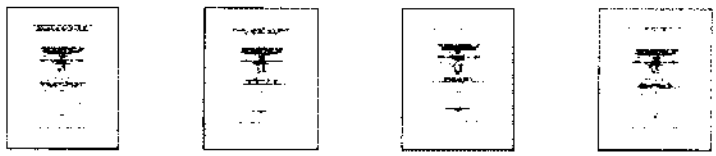
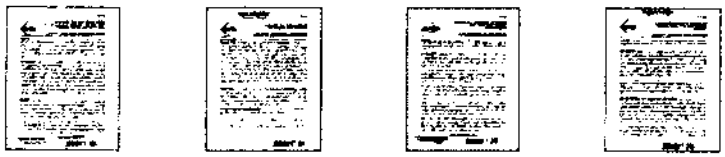
date
reasons
estimate
christobal
laden
critical
personal
document
figure
schmit
civil
notice
direct
ecent
maxwell
previous
later
front
philadelphia
wallace
combined
field
institute
plots
documentation
ericson
berlin
peterson
part
changyu
infrastructure
agree
doi
cceenntteerr
actions

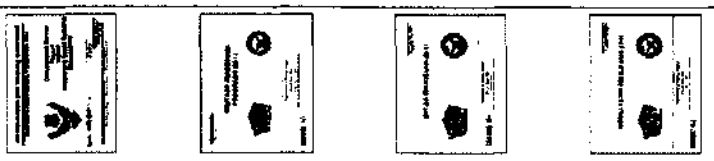
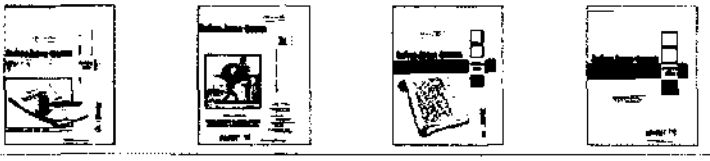
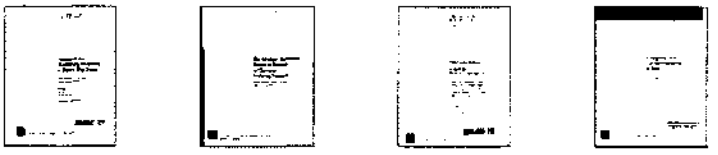
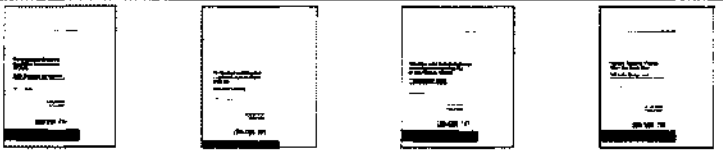
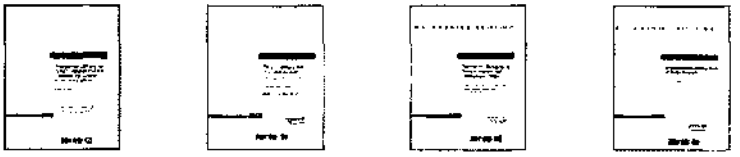
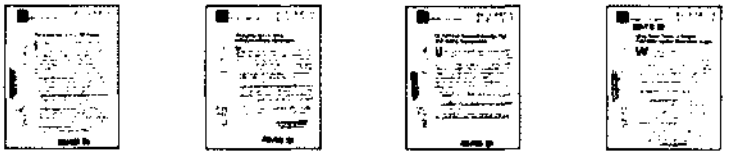
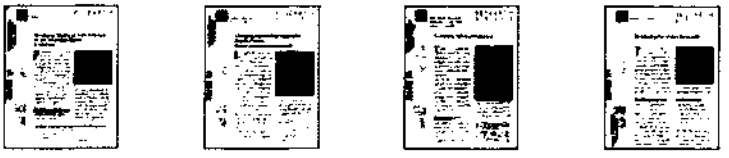
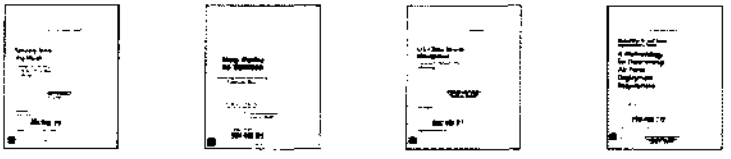
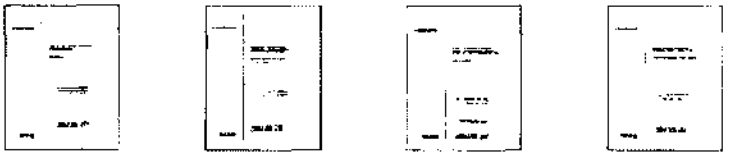
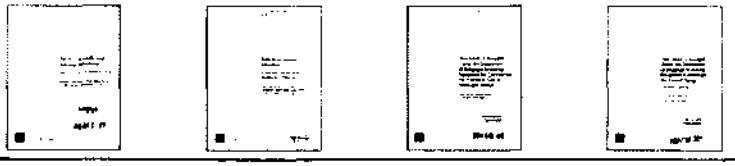
human
ima
two
electrical
twenty
shear
breaking
hughes
summary
"what
polychronicity
don
passed
specialist
characterization
received
optimal
washington
randolph
ssttuuddiieess
reduced
magnate
navys
virginia
hsingwang
signature
homeland
piece
arlington
mentor
books
deficiencies
francis
roots
hydromechanics
williams
international

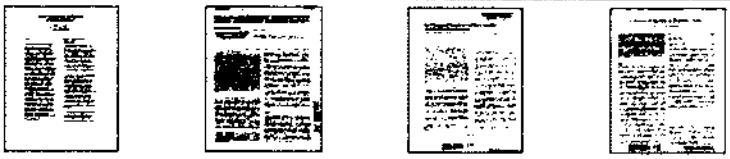
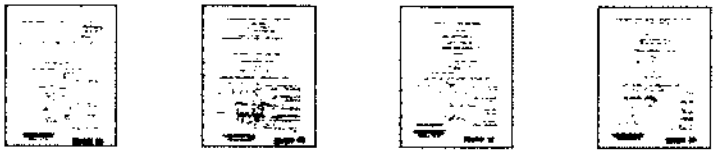
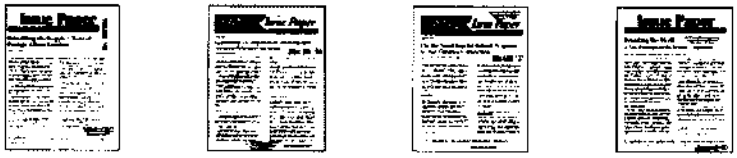
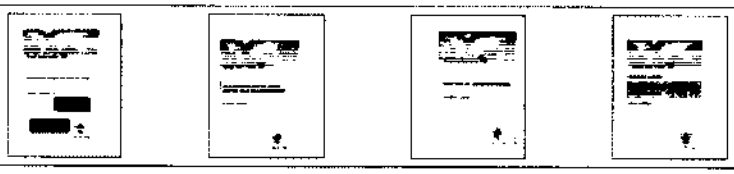
nonbreaking
realized
osama
born
give
progression
prescription
instructions
brian
toronto
terms
robinson
poorly
elsevier
cooperation
arms
abstract
carderock
those
bruce
susan
massachusetts
builder
copyright
loan
poposki
kenneth
ssg
hall
equine
complete
tom
supply
four
contracting
scalable
sublicensing

APPENDIX D

SIMILARITY EXPERIMENTS SAMPLE DOCUMENTS

ABSTRACT1-2COL	
ATOM	
AU	
BOTTOM-BLOCK	
CPRC	
EAGLE-IMAGE	
EAGLE-TEXT	
ERDC	

HORIZ	
LOGI	
RAND-ARC	
RAND-ARROYO	
RAND-ARROYO2	
RAND-BRIEF1	
RAND-BRIEF2	
RAND-LEFT	
RAND-NOTE	
RANDTECH	

RESEARCH	
SIGNATUR	
TOPLOG-2COL	
WARCOLLEGE	

VITA

Paul K. Flynn

Computer Science Department

Old Dominion University

Norfolk, VA 23539

EDUCATION

B.S. Computer and Information Engineering Sciences, December 1989, University of Florida

M.S. Computer Science, December 1998, Old Dominion University

Paul K. Flynn was born in Philadelphia, PA in 1961. After briefly attending Drexel University, he enlisted in the United States Marine Corps in March 1981. He served as an Arabic Cryptologic linguist in a variety of field locations and as a Traffic Analyst at the National Security Agency before being selected for a commissioning program. Upon graduation from the University of Florida, he was commissioned as a 2nd Lieutenant in 1989 and was assigned as an Intelligence Officer with the 1st Marine Brigade in Kaneohe Bay Hawaii.

In 1995, then Captain Flynn joined the initial cadre of the Joint Targeting School. He developed coursework in intelligence, collections, geospatial information and weapons of mass destruction effects. Over the next six years, he served as a primary or alternate instructor for virtually every course offered at the school. During this time he also earned an M.S. in Computer Science from Old Dominion University.

After retiring from the Marine Corps in March 2001, Mr. Flynn served as the Instrumentation Engineer at the Joint Battle Damage Assessment, Joint Test and Evaluation Program. He was hired by Applied Research Associates, Inc. in April 2004, where he worked on the development team for the Defense Threat Reduction Agency's Integrated Weapons of Mass Destruction Toolkit and was the Lead Developer for the Simulation Subsystem.