IJDAR
DOI 10.1007/s10032-006-0032-y

ORIGINAL ARTICLE

Biblio: automatic meta-data extraction

Carl Staelin · Michael Elad · Darryl Greig · Oded Shmueli · Marie Vans

Received: 17 November 2004 / Revised: 4 May 2006/ Accepted: 18 September 2006

Abstract Biblio is an adaptive system that automatically extracts meta-data from semi-structured and structured scanned documents. Instead of using hand-coded templates or other methods manually customized for each given document format, it uses example-based machine learning to adapt to customer-defined document and meta-data types. We provide results from *experiments* on *the recognition of document information in* two document corpuses: a set of scanned journal articles and a set of scanned legal documents. The first set is semi-structured, as the different journals use a variety of flexible layouts. The second set is largely free-form text based on poor quality scans of FAX-quality legal documents. We demonstrate accuracy on the semi-structured document set roughly comparable to hand-coded systems, and much worse performance on the legal documents.

Keywords Document recognition · Document understanding · Neural networks · Support vector machines · Machine learning

C. Staelin (✉) · M. Vans
Hewlett-Packard Laboratories, Technion City,
Haifa 32000, Israel
e-mail: carl.staelin@hp.com

D. Greig
Hewlett-Packard Laboratories, Filton Road, Bristol
Stoke Gifford, BS34 8QZ, UK

M. Elad · O. Shmueli
The Computer Science Department,
Israel Institute of Technology, 516 Taub building,
Haifa 32000, Israel

1 Introduction

Biblio is a system for automatically extracting information from scanned documents. A scanned document is an electronic representation of a collection of pages. The electronic representation typically consists of an image component and an optical character recognized (OCR) component. Electronic documents may be viewed as images, or as text objects, and contain both representations. Documents may be annotated with additional information in an extensible format (XML). For example, documents may contain hyperlinks, tables, logos, line art, or other automatically recognized information. A document may be as small as a receipt or as large as a thesis. Users annotate documents which Biblio uses to learn how to classify documents and extract user-specified information.

The space of documents contains a range of document structure types from highly structured forms such as invoices or tax returns through partially structured business letters to unstructured text. Biblio was designed to make it easy to add the ability to automatically extract information from the full range of document types. The current version of Biblio works with highly structured and partially structured documents.

The goal is to identify the document type, and then to recognize the relevant meta-data embedded in the document. A chief design goal was maximal flexibility and adaptability with zero document-type-specific coding of any sort. This is achieved via statistical and machine learning techniques in the form of neural networks and support vector machines. Another design goal is fast operation during recognition with the assumption that training might be done during off-hours and if necessary could consume significant computational resources.

Most machine learning methods attempt to learn, or fit a model to, a $\Re^n \mapsto \Re^m$ function. Unfortunately, it is difficult to map whole documents into a fixed-size $\Re^n$ vector in a way that captures sufficient semantic information to make meta-data recognition and extraction possible. The Biblio architecture breaks documents down into fixed size blocks using sliding windows and layout information to facilitate this requirement.

Processing is done in several phases with each phase typically analyzing the document in greater detail at a finer resolution. Various alternatives are pruned at each step using a combination of support vector machines (SVMs) and neural networks to minimize the overall required computation.

Biblio operation has two major modes: training and recognition. The training system adaptively modifies the system based on user feedback and input. The recognition system classifies documents it has been trained to recognize. Biblio is intended to be an automated system with as little user interaction as possible. In addition, Biblio does not require the user to understand anything about how Biblio works; all user-interactions are in terms of items that are of interest to the user, such as defining or specifying document and document-specific types.

Both modes of operation use a cascading series of classifiers. This was done in order to increase the performance. In most documents a small proportion of the data is specific to the document type. By cascading classifiers we are able to quickly throw out large chunks of the document without needing to look at each word, sentence, etc. During recognition, if the page classifier finds no evidence of meta-data at the page level, the rest of the page need not be examined.

The user interacts with Biblio indirectly through a document browser. While viewing a document, the user can specify the type of the document (e.g., "bank statement" or "journal article") and may establish new document types. Each document type has an associated list of *meta-data* types (e.g., "account number", "author"). The browser allows the user to highlight text on the document and indicate that the highlighted text is of a particular meta-data type. Users can also define new meta-data types for a particular document type. The browser will save the user-specified document type along with the meta-data in the document. Later, Biblio can use that information during self-training to improve its recognition capabilities.

During document acquisition, as part of the scanning process, Biblio will annotate documents with the information that it recognizes. Users can over-ride the automatically generated annotations, correct mistakes, or add information using the document browser. Biblio may use the modified document during re-training to improve its performance. Recognition run-time requirements are stringent since it is expected that the user is waiting for the process to complete.

The training system requires extensive processing and is expected to run without user guidance. It utilizes the sample (user annotated) documents and the document types along with their associated meta-data types to generate the data structures and systems utilized at run-time by the recognition system.

The focus of this paper is on the *information extraction* component of Biblio. We report the results of an experiment designed for determining how well Biblio performs during meta-data extraction. The experiments were run using only two types of documents and therefore we will defer any claims to Biblio's document classification ability to a later paper.

The next section describes prior work in document analysis and classification. Section 3 discusses the machine learning and information retrieval technologies on which Biblio relies, including neural networks and support vector machines. In Sects. 4 and 5, we give a description of the system and the major software components. Section 6 delineates our experimental setup for testing Biblio while Sect. 7 reports on results of those experiments. We conclude in Sect. 8 along with a discussion of future work.

2 Prior work

A great deal of work on document analysis and classification has been done in the areas of document management systems and document recognition (e.g., page decomposition and optical character recognition) [1,8,9, 15,16,25,30]. In this article we are concerned with techniques for identifying the type of a new document from some set of known document types. This is intended for a document understanding system that puts no prior conditions on the documents presented to the system. This contrasts with much of the current work (e.g., [18] and the proceedings of ICDAR, DAS), which often focuses on only one or more well defined document types. For example, title pages of books [32], journals [11], business cards [31], business letters [6,34,35], or office documents [23]. In each of these articles, the specific semantic or structural characteristics of the particular document type under consideration is exploited to analyze the document contents. There have also been a number of more general systems proposed which deal with multiple document types. Lam [12] presents a general document understanding framework (see also Srihari [26]), which contains different processing elements for different document types. However, it appears that document types are separated by some kind of identity strings printed on the document itself and subsequently recognized by the system, rather than structure-based type detection.

In Wenzel [34], documents are represented as directed graphs with vertices given by the compounds obtained from a segmentation algorithm. This technique allows both document type identification and document meta-data classification by using graph isomorphisms to find the best match in existing document databases. Note that this technique is similar to the constraint solving approach used by Lam [12], which could equally be used to classify document types. Another scheme for processing general document types is presented in Taylor [28], but once again the problem of automatically determining the type of a new document is not broached. In Casey [4], an input document is classified as one of a number of known form types by matching the lines found on the page with the form type database. This procedure relies on the fixed structure and scale of the documents involved.

The work most similar to ours is Wnek [36], which employs automatically generated document templates using inductive learning from annotated example documents. Biblio uses similar data elements for learning and prediction, for example, attributes associated with pages, lines, words, and characters. However, Biblio works with additional elements such as paragraphs and columns. Furthermore, because we use neural networks as a learning methodology, Biblio can discover characteristics common to document types and elements of which the user may not be aware. Another differentiator between the two systems is that Biblio can work on both structured and unstructured document types while the generalized document template system works only on structured document types.

CiteSeer [13] analyzes PostScript and PDF documents to extract meta-data and citation information. It uses hand-coded parsing and recognition engines to extract both document meta-data such as author, title and date, and citations to other documents.

3 Machine learning and information retrieval

Biblio exploits techniques from both machine learning (neural networks and support vector machines) and information retrieval (term weighting methods in text retrieval) to perform document recognition.

Machine learning can be described abstractly as a black box that predicts a vector of outputs when given vector inputs. Generally, this involves a "training" phase, and an operation or prediction phase. During training, the system is exposed to a number of input vectors together with the desired output vector. The system uses these examples to "learn" how to respond to inputs. During operation the system is simply given the input vector and it uses its stored knowledge to predict the output vector.

One key element of Biblio's design goals are that it can operate independently without a skilled operator tuning the machine learning system(s). Most neural network and support vector machine implementations require the user to tune one or more parameters to obtain the optimal performance. Since this was not possible in our case, we either developed or adopted techniques to automatically control and optimize the machine learning meta-parameters.

3.1 Support vector machines

Support vector machines are a kernel-based approach to machine learning [3,5,19,29]. We used the publicly available system, LIBSVM, as the support vector machine engine. LIBSVM includes four kernel functions: linear, polynomial, radial basis function (RBF), and sigmoid. We used the RBF kernel function which is defined as:

$$\text{RBF}: \qquad k(\mathbf{u}, \mathbf{v}) = \mathrm{e}^{-\gamma|\mathbf{u}-\mathbf{v}|^2}. \tag{1}$$

During training, Biblio's SVM engine creates input vectors representing the words found in the training documents. Biblio uses the RBF kernel function with autonomous SVM parameter selection [27] to create a set of support vectors for each type of known meta-data. These vectors are then stored along with the parameter settings to be used during the recognition phase.

3.2 Evolutionary neural networks

Neural networks are an example of a machine learning technology. Biblio uses Hplinet (an in-house neural network application) for both training and operation. Hplinet is an artificial neural network package with an evolutionary mechanism for automatically configuring network architecture based on ideas from Yao [37,38]. It employs training algorithms that do not require user defined parameters, such as BFGS [2] and LBFGS [14], while using techniques from evolutionary algorithms to search for the optimal network architecture. The networks use a generalized feed-forward architecture. It also uses committees of networks in order to increase the stability of predictions.

Hplinet uses gradient-based training techniques to train networks for a given architecture and training set.

Depending on memory availability, the system switches between full quasi-Newton training (BFGS) and a limited memory version (LBFGS). The major difference between these two algorithms is that BFGS stores an approximation of the full inverse Hessian matrix of the network function with respect to the network weights (*n*Link by *n*Link elements), whereas LBFGS ignores the off-diagonal elements and stores only the diagonal of this matrix. The LBFGS algorithm pays a penalty in convergence time, but does offer a reasonable alternative in low memory situations.

The network architecture must also be chosen. This means determining the number of hidden nodes and the structure of the links connecting input, output, and hidden nodes. The size and complexity of the architecture is generally a good measure of the complexity of the output function, and it can have a dramatic effect on both the duration of training time and the accuracy of the resulting network [7].

It is well known that neural networks often converge to local minima in the error surface, and that different initial weight configurations can lead to networks of identical architectures giving quite different solutions on the same training data. In addition, neural networks often have areas of the input space where they perform very well, but they may have areas where they perform poorly because the network has not learned to represent the function accurately in that region of the input space. A more stable overall solution can be obtained by training multiple networks and combining them into *committees*.

Committees are groups of neural networks, often trained on the same data but with different initial weight settings. During runtime each network is given the data and their outputs are combined to form a final output using any of the number of possible methods, such as voting. In this way, overall accuracy of the system can be increased by factoring most of the erroneous predictions.

In general, Biblio trains ten networks for each problem. It uses a separate validation data set to rank the performance of the networks and chooses the best five networks to create a committee. The committee result is created by averaging the outputs of the five committee members. For classification problems, the real-valued network outputs are thresholded to binary values and a majority vote is used instead of a simple numeric average.

3.3 Information retrieval

Broadly speaking, information retrieval techniques operate on text streams, and they are commonly used for managing and searching large text corpuses such as the Internet or a library. They are also used for routing and filing documents such as emails into folders. The most common approach is to build a dictionary of known terms and to assign each term a unique index. Documents can then be represented as a vector, with the values typically computed as a function of the term frequency within the document. A Boolean model would simply set the value to 1 if the term appears and 0 otherwise. A more powerful and commonly used function is TFIDF [22] which is computed based on the term's frequency within the document multiplied by the inverse of the frequency with which the term appears in all documents.

Biblio uses word dictionaries to determine if the text is indicative of a particular type of meta-data. There are two types of dictionaries: meta-data type dictionaries and special dictionaries. Meta-data dictionaries are used to represent the words associated with a given meta-data type. Special dictionaries are used to represent words that may be common across meta-data types, such as names. The function of meta-data dictionaries is to give the probability that the given text stream contains text representing a particular type of meta-data. During training the system is given example text streams from the training documents. The text streams may or may not contain meta-data. Currently the dictionaries are updated manually; however, the dictionaries need to be self-updating without user intervention during training.

There are a number of models and approaches for text classification, such as Naive Bayes [21] and SVMs [17]; however, we currently use SVM with RBF kernels. The SVM engine also uses a dictionary that contains a unique ID for every word it encounters. Each time it discovers a new word, a unique ID is generated and the word is stored in the dictionary permanently. During training, the SVM engine builds input vectors using IDs from the dictionary to identify words in the document. This allows the SVM to create support vectors that identify the most probable words associated with a particular type of meta-data. During operation, the dictionary is again used by the SVM engine to build the input vectors for use with the SVM model created during training.

4 System description

There are two modes of operation for Biblio: training and recognition. Biblio uses several analysis engines for both types of processing; however, they are used in different ways. This section describes both systems, the analysis stages and how they differ between the two modes, and the major data structures used in training and recognition.

4.1 Recognition system

For recognition, Biblio is designed to eliminate many possibilities early to reduce the processing requirements.

The recognition the system is structured as follows: There are some number of user-visible document types such as “Business Letter”, “Journal Article”, or “Bank Statement”. Each document type has some meta-data types such as “Author” or “Account Number”. There are typically many example documents of a single type, which are available to the system as it trains and re-trains itself. Within a single document type, Biblio may break documents down into document classes, which are generally documents that are similar to each other within the document type. Each document class has its own set of neural networks and SVM models for detailed processing. The system itself creates the document classes from the pool of documents in the document type.

Recognition is done in stages, with possibilities filtered at each stage. Before the first stage, Biblio has no knowledge about the document. It could belong to any of the document classes and could contain any type of meta-data. To minimize the computational costs, Biblio tries to eliminate as many candidate document types as quickly and cheaply as possible.

The first stage, compound analysis, is both content and layout-based and is done for each document type. Each compound is flagged with the meta-data types that may be contained in that compound. Compounds perceived to contain no meta-data are discarded and ignored in further processing.

In the second stage, document type analysis, the system evaluates the probability that the document is of a particular type. For each document type, the system uses the information from the compound layout information to compare the document’s structure to documents of that type. Biblio chooses the closest match or rejects all candidate types. From this point forward, all processing is done with respect to a single candidate document type.

There are three following stages, paragraph analysis, line analysis, and word analysis, which iteratively analyze the compound, paragraph, and line to select or reject paragraphs, lines, and words that contain meta-data. Each paragraph, line, and word is flagged with the meta-data it is perceived to contain, and the flagged pieces are passed on for further processing. At the end, Biblio takes the remaining words and annotates the document with the recognized document type and meta-data.

4.2 Training system

The training system adaptively modifies the system based on user feedback and input. Training is designed to be run as a batch processing job, preferably during low usage times such as overnight. This is because training may consume resources at an unacceptable rate while the user is actually using the system.

Each document type has several example documents on which to learn. We use neural networks and support vector machines but training may also include actions such as updating word databases or other supporting data structures in response to the user’s feedback.

5 Analysis engines

Biblio uses a variety of analysis engines and techniques to evaluate the document at each analysis stage. Analysis engines create evidence and are called witnesses. Judges assemble witnesses, collect and collate evidence, make judgments, and propagate verdicts. Judges use special witnesses, called chairmen, to produce final evidence for each meta-data type in the document class, and this evidence is used to produce the verdict. This modular architecture makes it easy to add new analysis engines.

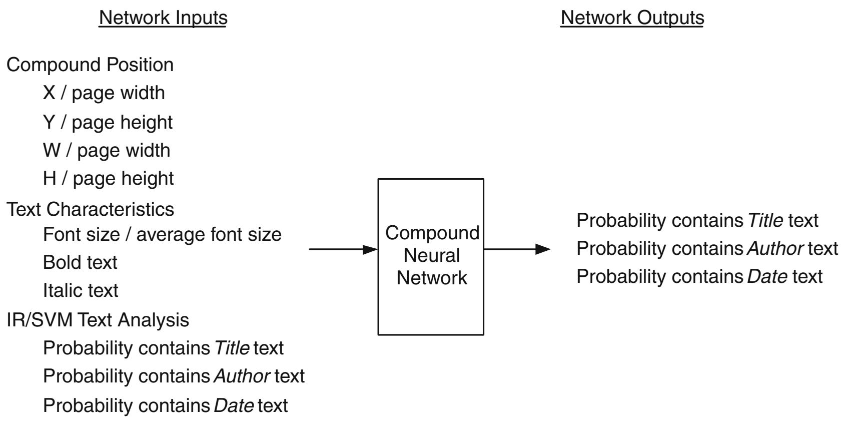
Biblio uses a layered hierarchy of analysis engines to develop and iteratively refine evidence and classification information about each information layer: compound, document, paragraph, line, and word. Each layer serves as a filter; portions that are rejected are not processed further while the remaining portions are passed on to the remaining filters. These layers correspond to the five analysis phases. Each analysis phase has a complete set of analysis engines trained exclusively for that phase and document class.

We use a cascade of neural networks and support vector machines (SVM) for learning and recognizing documents and document data. During training, the SVM engines are used for creating dictionaries at each information layer. During recognition, these dictionaries are used by the neural networks while looking for evidence of data specific to a document type. While we investigated other techniques, we decided on the SVM approach because of its efficient use of space and time.

5.1 Compound analysis

At the beginning of compound analysis during recognition, each document contains a list of compounds, which are really OCR recognized regions. The compound analysis is run separately for each candidate class, and the

Fig. 1 Compound processing


Network Inputs
Network Outputs
Compound Position
X / page width
Y / page height
W / page width
H / page height
Text Characteristics
Font size / average font size
Bold text
Italic text
IR/SVM Text Analysis
Probability contains *Title* text
Probability contains *Author* text
Probability contains *Date* text
Compound Neural Network
Probability contains *Title* text
Probability contains *Author* text
Probability contains *Date* text

output for each class is only passed to the document analysis for that class. Each document class has its own compound analysis neural network.

Compound analysis annotates each compound with the likelihood that each type of meta-data is contained in the compound. It only examines the list of meta-data types that are associated with the particular document class.

Compound analysis uses a neural network to annotate a single compound at a time. Since the number of meta-data types for each document class is fixed (at least until the user modifies the list), the neural network can output the probability for each meta-data type directly. Figure 1 is a graphical representation of the network for compound analysis. It shows the inputs and the outputs of the network.

Neural networks require a fixed number of numeric inputs, but document content is not usually fixed size and is usually not described numerically. Biblio translates the compound contents and layout into a fixed number of numeric inputs. The inputs are:

1. The bounding box of the compound
2. The font size of the text
3. The probabilities generated by the support vector machines for each meta-data
4. The proportion of all words found in the compound that are associated with known meta-types
5. The proportion of descendant words (*words contained within the compound*) containing lower case letters
6. The proportion of descendant words containing upper case letters
7. The proportion of descendant words containing upper case digits
8. The proportion of descendant words containing upper case symbols
9. The proportion of descendant words containing upper case initials

During training, the system generates the support vector machine output for each compound first. The system then trains a committee of networks to recognize compound types for each document class. It uses the probabilities produced by using the support vectors as well as the layout inputs. It creates sample inputs by feeding the system example compounds with examples containing meta-data and some that do not contain meta-data. The number of training examples is bounded and training time is generally reasonable.

5.2 Document analysis

During recognition, document analysis is done on a per-document class basis. The per-class results are compared to choose the best match. Each document class has its own document analysis neural network.

Document analysis uses a neural network to evaluate the probability that a document belongs to that class. Each meta-data type has a set of inputs to the neural network which include the same inputs as the compound analysis inputs. It uses the compound analysis results to create a pseudo compound based on the union of the compounds containing that meta-data type.

During training, the system trains a committee of networks on each document class. It takes example documents from all classes to train the network to distinguish between document classes. Any time a new class is added, all document analysis networks need to be retrained, which can be very expensive.

In the future, we would like to apply signature filtering [24] that will guarantee only documents from certain document classes will be passed. If we are successful, modifications to radically dissimilar classes should not require retraining of the document network.

5.3 Paragraph and line analysis

The paragraph and line analysis is done only for the most likely document type during recognition. It uses the information from the compound analysis to eliminate portions of the document from further processing. Each document class has a neural network for each meta-data type. The neural networks take the previous compound or paragraph analysis results and SVM probabilities as input. For each meta-data type, the networks output the likelihood that the paragraph or line contains that meta-data.

A committee of networks recognizing the paragraphs or lines that contain meta-data are trained for each document class. These networks are trained only with examples from the document class, and training time is generally reasonable. These networks only need to be retrained if there are changes in the target document class.

5.4 Word analysis

The final step for the recognition process is to examine each word in the remaining lines and identify those that are associated with a particular meta-data type. This is currently done using a neural network that takes token information for three words and outputs information about the middle word.

By giving the neural network information about three words, we give the network some context for the current word, which dramatically improves the network's accuracy. However, these networks are large, requiring more inputs and more hidden units than any other network in the system.

For each document class the system trains, a committee of networks scan the individual words in selected lines. The networks are only trained with examples from lines containing meta-data in the target document class.

Since the system creates a training example for each word, the number of training examples is huge and the training time can take up to a day (24 h) on 50 documents. This is currently the single largest consumer of CPU time in the training system. These training time problems compel us to look for alternative strategies for extracting the meta-data from each line.

6 Implementation

This section describes the major software classes and their methods during training and operation. Figures 2 and 3 are block diagrams of the major objects and their interactions.

6.1 Portfolios, hubs, and evidence

Portfolios contain document hubs, which are objects that contain information about the files on which Biblio is

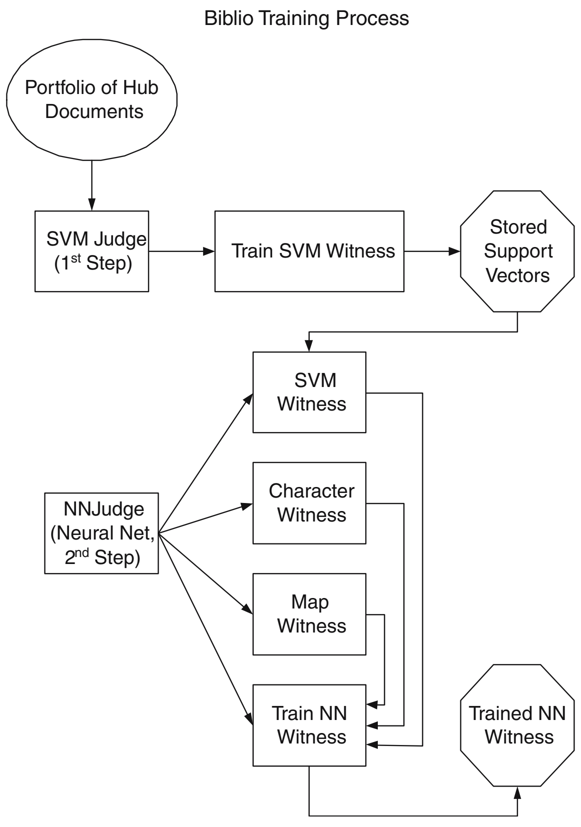


Fig. 2 Biblio training block diagram

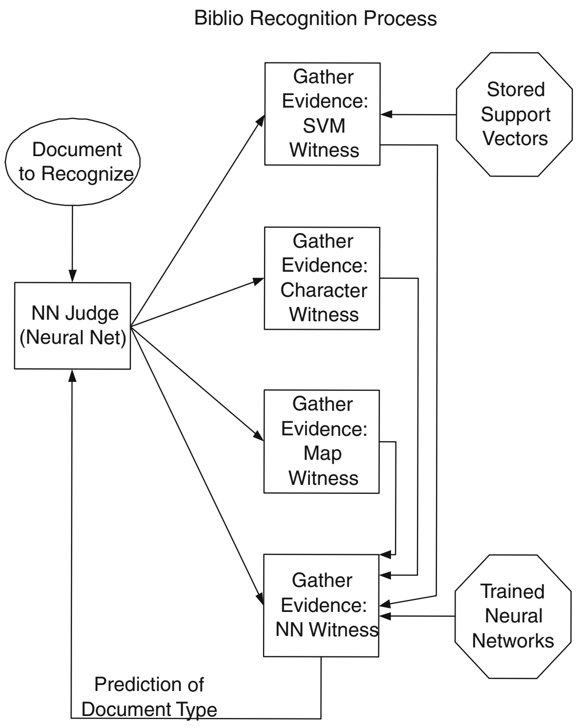

Biblio Recognition Process
Document to Recognize
NN Judge (Neural Net)
Gather Evidence: SVM Witness
Gather Evidence: Character Witness
Gather Evidence: Map Witness
Gather Evidence: NN Witness
Stored Support Vectors
Trained Neural Networks
Prediction of Document Type

Fig. 3 Biblio operation block diagram

currently working. During training, the portfolio can have as many files as is physically possible to load into memory. In our case, we used a maximum of 50. During processing, the portfolio contains a single document hub, the document it is trying to recognize. The portfolio is mainly responsible for the creation and removal of hubs.

Each hub contains information about the file it represents. The main components include an XML tree representation of the document and evidence objects. The XML representation makes it easy to quickly access any part of the document at any level. In addition, a hash table containing all the words in the document is included for fast word searches. The evidence objects work as a central repository that is used by witnesses for storing and accessing evidence. During training, the evidence object contains lists of probabilities for each type of known meta-data. Each witness is responsible for determining the probability of existence for each type of meta-data at the current level, i.e., compound, paragraph, line, and word. A special witness, the chairman, later collects all the evidence for use in training the neural networks. These networks are trained to recognize meta-data. During operation, the evidence object is again used to collect evidence by witnesses and the chairman later collects this evidence. The difference is that during recognition, the chairman uses the evidence as input to a neural net trained to recognize meta-data.

6.2 Judges and witnesses

Biblio was designed to make it easy to add functionality. This could include alternative machine learning technologies, document processing capabilities, statistical, and mathematical algorithms. This means that whenever a promising technology with application to machine learning or document understanding is invented, it can easily be incorporated into Biblio.

Currently, Biblio has two specialized *judges* whose job consists of organizing *witnesses* for specific types of *evidence*. These are the neural-net judge and the support vector judge. Both are derived from a base judge that defines the basic operations such as creating witnesses, training them, initiating prediction, and acting as a repository for evidence returned from witnesses. Another vital task of the judge is to separate data into three disjoint sets: training, test, and validation. For our experiments, the test set was given. That is, we took one file from the list of files, held it out for test, and used the remaining files for training or validation. The judge uses a random procedure that assigns roughly 75% of the remaining files to a training set and 25% to the validation set. This design makes it easy to derive any specialized judge that can start up training for specialized witnesses and collect evidence from these witnesses during the prediction phase.

6.2.1 SVM judge

The SVM judge is responsible for directing the training of the SVM witness. During training, the SVM judge creates the SVM witness and passes a portfolio of documents, a specification of the type of the documents, and the level at which the witness should train, i.e., compound, paragraph, line, or word. The SVM witness is specialized from a base witness class which contains operations that are generic to all witnesses as well as special operations related to support vector machine processing. Some of the base operations include training and prediction. The main function of this judge is to direct the SVM witness in creating support vector models. These models are stored for later use in the prediction process, a process in which the SVM judge is not currently involved.

The SVM *witness* The SVM witness is responsible for loading dictionary objects. There are two types of dictionaries used by the SVM witness. The main dictionary object contains all the words from every document ever seen by Biblio during training together with a unique ID. When a new document is included during training, the dictionary object generates a unique ID and updates the dictionary with the new ID/word pairs. This allows the dictionary to grow over time. During training, the SVM witness creates another dictionary consisting of words from the documents in the training set known to be associated with specific meta-data types. For example, for the journal article class of documents, there are files containing words *such as "journal name" and "author"* that are specific to the types of meta data found in journal articles. The main drawback of these files is that there is currently no way to automatically update them, instead they must be edited by hand.

Once both dictionaries are loaded, the SVM witness builds input vectors for the support vector machine, one set of vectors for each type of known meta data. First, the witness creates five or six support vector entries to record the existence of words from the main dictionary. Each entry in the vector contains the word ID and the proportion of the current word contained at current level, e.g., compound, paragraph, line, and word. Once the input vectors are created, a target vector is created for each word represented by the vector. The target value indicates whether the word is associated with the particular type of meta-data currently in focus.

Before sending the input/target pairs to the support vector machine to create the final model, a parameter search [27] is initiated to find the best combination of C and gamma values. Support vector machine training is executed using these values and once training is complete, prediction and cross-validation is used to determine how well the system did using those parameter values. Based on these results, new parameter values are generated which are closer to the best values from the previous run and the cycle is repeated. We iterate through the process five times, after which the values returned for gamma and C are used for actual training and model creation. Once the model is created, it is written to file where it can be accessed by other instances of the SVM witness during prediction.

6.2.2 Neural net judge

In addition to the SVM judge there is also a neural net(NN) judge which is responsible for training and prediction. Similar to the SVM judge, the neural net judge is mainly a supervisor for witnesses. The NN judge creates instances of four witnesses: a SVM, a character, a map, and a neural net witness. The character witness is responsible for collecting evidence on upper and lower case letters and the presence of symbols and digits. The map witness collects evidence on bounding boxes, average font size, the relative position of the current compound (paragraph, line, word) within the parent compound, and the relative number of tokens found in the compound. The neural net witness is responsible for collecting evidence from the other three witnesses and creating the input vectors for training the committee of networks.

The character witness The hub of each file contains a hash table of all the words in the document. Attached to each word is a bit array containing a single bit for each type of attribute the character witness knows about, i.e., upper case, lower case, symbols, digits, and initials. This information can then be used to indicate meta-data with specific formatting characteristics. For example, titles or proper names usually include upper case characters; names often include initials; digits may support the presence of a date or page numbers; URL and mathematical formulae usually include symbols. This witness first initializes the word bit arrays and fills them in for every word in the document. Later it collects evidence by going through each word in the document and recording the proportion of words containing each attribute in the document. It then stores this information in the evidence object of the hub. This process is the same for both training and recognition.

The map witness The map witnesses can supply up to seven types of evidence: the bounding box of the type of compound, i.e., compound, paragraph, line, word; the relative position of the compound within its parent, the average number of tokens in the compound, and the average font size in the compound. Boolean flags passed to the witness indicate which evidence needs to be collected. In some cases various types of evidence may not be pertinent to the type of document on which the system is being trained. The XML representation of the document makes calculating the bounding box efficient as the height and width of every component is stored as part of the compound object when it is created. The relative position is also quickly determined because the compound has access to its parent and the parent is aware of all its children. Bounding box and relative position evidence may be important, especially for highly structured documents such as telephone bills which would have the same information displayed in the same relative location on every bill. The average font size is also easy to determine as the font size *of each text component* in the compound is stored in each compound object. We can calculate an average font size for the compound by dividing it by some maximum number, in our case we used 50. This tells us whether the average font size is larger or smaller than usual. It can be very helpful to know what the average font size is when trying to identify things such as headings and titles. Finally, the average number of tokens is calculated by determining the number of words subordinate to the current compound and dividing that number by some maximum. Similar to the average font size measure, this gives us an idea of whether the number of tokens is small or large.

The SVM witness The SVM witness behaves differently depending on the judge that creates it. If the judge is a SVM judge, the witness creates the SVM model as described above. If the judge is a neural net judge, then the SVM witness uses the model it previously created to collect evidence for both the training and recognition processes. As with model creation, the SVM witness must create input for the support vector machine from the document. It does this in exactly the same way as when building the model. However, during evidence collection, the SVM witnesses loads the model, calls the support vector machine prediction module, and passes it the input vector. The SVM witness uses this model, along with the previously determined parameters to predict whether each of the words represented by the input vectors contain meta-data. Prediction returns probabilities for each word in the compound. These probabilities are then put into the evidence object of the corresponding hub.

The neural net witness The neural net witness is the chairman. This means it is responsible for collecting all the evidence produced by the other witnesses and using it to either train a committee of neural networks or to use as input for recognition. During training the neural net builds a single training object containing all the evidence from each hub evidence object in the training set. It also builds a similar object using the validation set of hubs. Target arrays are built that associate the true value to each piece of evidence collected. It then creates a committee of evolutionary neural networks, passing in the evidence and target arrays. The number of networks created is configurable. In our experiments we used committees of ten networks. Once the network has found the optimal architecture and configuration, the witness stores the network for use during prediction.

During prediction, the neural network witness collects all the evidence from the hub evidence object and creates a single input vector. The witness then loads the network architecture from the previously stored file and passes it to the input vector. Output from the committee consists of probabilities for each type of meta-data. This is stored back into the hub's evidence object and passed back to the judge. The judge will then be able to use the evidence to decide on the type of document based on the probabilities of meta-data found.

7 Experimental setup

We used a corpus consisting of a few hundred documents. These documents were either articles from various computer science journals or legal documents recording transfer of properties, i.e., deeds. All were scanned into the system and stored in an HP proprietary file format consisting of both images and text recognized by the scanner. Meta data describing the document and identifying key words (i.e., words that are unique to the type of the document trained on) is embedded within the XML.

We trained Biblio on 50 documents at a time, as the memory requirements for larger document sets was prohibitive. During training, the documents are decomposed into regions consisting of major regions, paragraphs, lines, and words. The system considers each type of component separately and determines the support vectors based on known meta-data types for each type of component. Once the support vectors are identified the system uses these vectors to generate probabilities that each component contains a specific type of meta-data. The resultant probabilities are fed into the neural network as evidence for training the neural network. The network takes the evidence and builds a parsimonious architecture based on the best results. This means the final network consists of the architecture with the fewest nodes and is close to the absolute best architecture, which may actually have many more nodes. Biblio then saves this network for use during operation.

During operation, or prediction, the neural network acts as the driver for the entire process. In general, evidence is gathered on a previously unseen document and the trained network uses this evidence to determine the type of the document. We tested the meta-data recognition aspect of Biblio in these experiments by giving Biblio the document type with which to begin. The evidence gathering process starts the SVM process. This process takes the stored support vectors for each type of known meta-data and determines probabilities that are subsequently fed into the trained network. Once the network has identified the document type and the meta-data contained within it, the system annotates the original file by adding the document type and the recognized meta-data.

To test the system, we use leave-one-out cross fold validation for our experiments [33]. For each of the 50 documents we hold one document out, train on the remaining documents, and then predict whether the document held for testing contained meta-data appropriate to the type of document we trained on. During prediction we collect confusion matrix data to report the results.

7.1 Documents and meta-types

We ran our experiments on two types of documents: journal articles and grant deeds. The journal articles are an example of structured documents while the grant deed documents are an example of unstructured documents. Within the journal article type there are two classes of documents: articles from IEEE Transactions journals and articles from the HP Journal. For grant deeds there are six registered types of meta-data: Date of Recording - Meta0, County - Meta1, Document Number - Meta2, Return Address - Meta3, Grantor - Meta4, and Grantee - Meta5.

For journal articles there are five registered types of meta-data: Title - Meta0, Author - Meta1, Journal Name - Meta2, Pages - Meta3, and Date - Meta4.

8 Results

The results consist of statistics generated from the confusion matrices collected during our experiments on three sets of data.

8.1 Confusion matrix

Table 1 contains the confusion matrix information for all documents. The *Total* column is the total of all the words in the associated document class. For comparison, the total number of actual words containing

Table 1 Confusion matrix

Type	Total	TrueNeg	FalseNeg	FalsePos	TruePos	PosCount	NegCount
GRDE-Meta0	48,629	48,496	121	0	12	133	48,496
GRDE-Meta1	48,629	48,368	143	0	118	261	48,368
GRDE-Meta2	48,629	48,477	98	0	54	152	48,477
GRDE-Meta3	48,629	47,051	869	11	698	1,567	47,062
GRDE-Meta4	48,629	47,076	1,192	14	347	1,539	47,090
GRDE-Meta5	48,629	47,106	1,152	35	336	1,488	47,141
JA00-Meta0	19,208	18,991	28	0	189	217	18,991
JA00-Meta1	19,208	19,048	32	16	112	144	19,064
JA00-Meta2	19,208	19,044	11	0	153	164	19,044
JA00-Meta3	19,208	19,184	15	0	9	24	19,184
JA00-Meta4	19,208	19,161	3	4	40	43	19,165
JA01-Meta0	19,703	19,463	6	0	234	240	19,463
JA01-Meta1	19,703	19,521	27	17	138	165	19,538
JA01-Meta2	19,703	19,638	31	0	34	65	19,638
JA01-Meta3	19,703	19,670	15	0	18	33	19,670
JA01-Meta4	19,703	19,632	33	0	38	71	19,632

Table 2 Word recall

Type	WordPosCount	WordRecall
GRDE-Meta0	133	[0.04 – 0.14]
GRDE-Meta1	261	[0.39 – 0.51]
GRDE-Meta2	152	[0.28 – 0.43]
GRDE-Meta3	1,567	[0.42 – 0.47]
GRDE-Meta4	1,539	[0.20 – 0.25]
GRDE-Meta5	1,488	[0.20 – 0.25]
JA00-Meta0	217	[0.83 – 0.92]
JA00-Meta1	144	[0.71 – 0.85]
JA00-Meta2	164	[0.89 – 0.97]
JA00-Meta3	24	[0.18 – 0.57]
JA00-Meta4	43	[0.85 – 1.00]
JA01-Meta0	240	[0.96 – 0.99]
JA01-Meta1	165	[0.78 – 0.89]
JA01-Meta2	65	[0.40 – 0.64]
JA01-Meta3	33	[0.38 – 0.72]
JA01-Meta4	71	[0.42 – 0.65]

meta-data (column labeled *PosCount*) and the total number of actual words not associated with meta-data (column labeled *NegCount*) are included. Obviously, the majority of the words are not associated with meta-data. During testing, the neural networks used a threshold of 0.50 when deciding whether or not a particular word was meta-data. There are two types of errors: False Negative (prediction of false when the meta-data is actually there) and False Positive (prediction of true when no meta-data exists). Table 1 shows that Biblio does predict incorrectly. However, as Tables 3 and 4 below show, Biblio does not incorrectly label data very often.

8.2 Recall

Table 2 reports the Word recall of Biblio. Recall is defined as:

$$\text{Recall} = \frac{\text{TruePositiveCount}}{\text{TruePositiveCount} + \text{FalseNegativeCount}}. \quad (2)$$

In our case, recall is a measure of how well Biblio can detect real meta-data in the document when it exists. Results were divided into positive and negative predictions with positive predictions shown in the *WordPosCount* and *Count* column. These values reflect the actual number of meta-data words, that is, number of true positive plus the number of false negative predictions during testing. *WordPosCount* is the sum of all true positive and false negative predictions for all files.

One thing should be noted here about the sample populations. We had approximately three times more samples of the GRDE group than each of the Journal Article group samples. The GRDE group consists of three separate runs of 50 documents each, while the Journal Article groups consisted of one run each with approximately 50 documents. We combined the GRDE runs because there is essentially no difference between the three sets of data. However, there are differences between the Journal Article groups, for example, type of journal and layout.

The *WordRecall* column is the 95% confidence intervals for the recall measure.

8.2.1 Analysis

From Table 2 we can see a large difference between the GRDE and Journal Article document classes. In general, Biblio performed poorly in recognition of most meta-data types for GRDE. This is somewhat expected since GRDE falls into the unstructured document

Table 3 Positive word prediction rate

Type	FalsePos	TruePos	WordRate
GRDE-Meta0	0	12	[0.000 – 0.000]
GRDE-Meta1	0	118	[0.000 – 0.000]
GRDE-Meta2	0	54	[0.000 – 0.000]
GRDE-Meta3	11	698	[0.006 – 0.024]
GRDE-Meta4	14	347	[0.019 – 0.059]
GRDE-Meta5	35	336	[0.065 – 0.124]
JA00-Meta0	0	189	[0.000 – 0.000]
JA00-Meta1	16	112	[0.068 – 0.182]
JA00-Meta2	0	153	[0.000 – 0.000]
JA00-Meta3	0	9	[0.000 – 0.000]
JA00-Meta4	4	44	[0.006 – 0.176]
JA01-Meta0	0	234	[0.000 – 0.000]
JA01-Meta1	17	138	[0.060 – 0.159]
JA01-Meta2	0	34	[0.000 – 0.000]
JA01-Meta3	0	18	[0.000 – 0.000]
JA01-Meta4	0	38	[0.000 – 0.000]

Table 4 Negative word prediction rate

Type	FalseNeg	TrueNeg	WordRate
GRDE-Meta0	121	48,496	[0.002 – 0.003]
GRDE-Meta1	143	48,368	[0.002 – 0.003]
GRDE-Meta2	98	48,477	[0.001 – 0.002]
GRDE-Meta3	869	47,051	[0.017 – 0.019]
GRDE-Meta4	1,195	47,076	[0.023 – 0.026]
GRDE-Meta5	1,152	47,106	[0.023 – 0.025]
JA00-Meta0	28	18,991	[0.001 – 0.002]
JA00-Meta1	32	19,048	[0.001 – 0.002]
JA00-Meta2	11	19,044	[0.000 – 0.001]
JA00-Meta3	15	19,184	[0.000 – 0.001]
JA00-Meta4	3	19,161	[0.000 – 0.000]
JA01-Meta0	6	19,463	[0.000 – 0.001]
JA01-Meta1	27	19,521	[0.000 – 0.002]
JA01-Meta2	31	19,638	[0.001 – 0.002]
JA01-Meta3	15	19,670	[0.000 – 0.001]
JA01-Meta4	33	19,632	[0.001 – 0.002]

category, which we assumed would be harder to recognize. In addition, the GRDE files have numerous OCR errors. If meta-data words contain OCR errors, it is difficult for Biblio to recognize them as meta-data. While we did not track the OCR error rate as part of the experiment, we assume that lower OCR error rates would lead to better performance on the part of Biblio for those documents containing known meta-data. It is also possible that a different threshold value would allow better separation.

On the other hand, Biblio did quite well in recognizing meta-data in the journal articles. This is in spite of the fact that Biblio had less data on which to learn than the GRDE category. The main difference in results between the two types of journal Articles are with meta-data type 3 (page numbers). We theorize that this is because the first type, the HP Journal Articles, are much less structured than the IEEE articles. The format of HP Journals can vary widely. Meta data types 2, 3, and 4 for the HP Journal articles (Journal Name, Pages, Date) are not consistently placed on every page or between different articles. Another factor affecting the results could be related to the number of each meta-data type found in the test file. The table shows significantly better results for meta-data types with many examples. For example, both Meta0 and Meta1 (title and author) appear hundreds of times. The associated confidence intervals for both types of journal articles have upper ranges close to 1.0. The same is true for Meta2 (Journal name) for the IEEE journal articles. This would make sense if we can assume that the number of positive examples of each type of meta-data is representative of what we would find in a "typical" IEEE or HP Journal Article. The fact that Biblio did very well for Meta4 on the IEEE journal articles in spite of the relatively small number of samples could be related to the fact that IEEE articles have consistent formats.

8.3 Specificity

The vast majority of the predictions were negative, i.e., words that were not meta-data. This is the value containing the true negative plus the false positive predictions. These are the number of words and words in files containing meta-data, respectively, that did not actually contain any of the associated type of meta-data labeled on the row. Specificity is defined as:

$$\text{Specificity} = \frac{\text{TrueNegativeCount}}{\text{TrueNegativeCount} + \text{FalsePositiveCount}}. \tag{3}$$

or how well Biblio classifies non meta-data. We do not present a table for specificity because all specificities were either 1.00 or 0.99, with 95% confidence intervals between [0.99 – 1.00]. In general, this means that our prediction system is very good at identifying words that do not contain meta-data.

8.4 False positive ratios

Another interesting statistic shows the proportion of meta-data that Biblio predicted as being present that is not really there. Table 3 shows these results. The *WordRate* column reports the 95% confidence interval for the statistic. The ratio is computed using:

$$\text{Negative word rate} = \frac{\text{FalsePositiveCount}}{\text{TruePositiveCount} + \text{FalsePositiveCount}}. \quad (4)$$

This is an important result for potential users of Biblio. Because the majority of the words in any document will most likely not be meta-data, if Biblio reports too many false positives, users will spend time correcting errors. While it is not clear whether this problem is as severe as missing meta-data that does exist, it is an annoyance likely to keep users from using the system. As the table shows, *misclassifications* occur with *Meta1* (*Author*) *data* for both journal article classes. On average, when Biblio predicted Meta1 type for the JA00 class, 12% of the time it was incorrect. For JA01 the rate for Meta1 positive misclassification is 0.11. It is encouraging, however, that for 10 of the 16 meta-types, Biblio made no false positive predictions. The rest of the positive misclassifications occurred at a rate of 9% or less.

8.5 False negative ratios

The proportion of words that Biblio predicted as non meta-data words when they were actually meta-data words is reported in Table 4. The *WordRate* column reports the 95% confidence interval for the statistic and the ratio is computed using:

$$\text{Negative word rate} = \frac{\text{FalseNegativeCount}}{\text{TrueNegativeCount} + \text{FalseNegativeCount}}. \quad (5)$$

The table shows that Biblio does better in terms of predicting false negatives. The highest rate occurs for GRDE-Meta3 and GRDE-Meta4 and the average for these meta-types is around 2%. On the other hand, since the majority of the words in a given document will be negative, the number of false negatives translates into the number of corrections the user would have to make. Because the number of meta types in a given document is small compared to the number of non meta- data words, failing to identify even a small amount of the existing meta data could cause Biblio to incorrectly classify the document as a whole.

9 Conclusions and future work

Biblio is a meta-data extraction and document type recognition engine that utilizes a combination of machine learning technologies to adapt to user-defined document types and meta-data fields. While our experiments demonstrated the system's ability to recognize meta-data for certain document types, we need to test Biblio's document recognition capabilities using a much larger collection of document types. The data set we used consisted of only two types of documents. Additionally, there are a number of areas that we should like to improve and test.

We did not analyze run time for recognition and prediction; however, we are aware of the unacceptable training time needed by the word analysis engine. Training time for other components such as compounds, paragraphs, and lines were more reasonable, on the order of seconds. We need to investigate and implement a more efficient method for processing words.

Currently, all documents within a single user-defined class are treated as a unitary class. However, it is likely that a single class, such as "Journal Article" or "Invoice", would have sub-classes such as "phone bill" and "gas bill" which are far more regular. However, we would like to automatically discover clusters of similar documents within a user-defined class so as to take advantage of similarities without burdening the user.

Another area that we should like to improve is the ability of the system to recognize when a document is not a member of any known document class. While the system does this today, it is not sufficiently selective so documents can be annotated with spurious meta-data.

We are examining the possibility of using techniques similar to the methods used by Koller and Sahami [10] to identify key words in document clusters. This would involve analyzing the statistical patterns of text containing meta-data versus text that does not contain meta-data to give reasonable probabilities for unseen strings. The Naïve Bayes classification algorithm may be suitable for classifying text as containing/not containing meta-data [20]. The word dictionaries could also be automatically enhanced using the SONIA clustering method.

Additionally, we would like to improve the ability of the system to make use of *tagged* fields, such as fields commonly pr-/post-fixed with identifying strings such as "To:" or "Subject:". This requires analyzing neighboring regions to see if they indicate the presence or absence of meta-data in a given region, and this evidence can then be passed to the region analysis.

Finally, Biblio is currently a single-page analysis system, meaning that each page is analyzed independently. Sometimes documents contain important meta-data on more than just the first page. For example, some journals only put the journal name, volume, and number on subsequent pages of an article. We should like to extend the system so that it can analyze a whole document and extract any relevant meta-data regardless of page.

References

1. Baumann, S., Malburg, M., Hein, H.-G., Hoch, R., Kieninger, T., Kuhn, N.: Document analysis at DFKI, part 2: information extraction, German Research Center for Artificial Intelligence (DFKI). In: DFKI Research Report, no. RR-95-03, Kaiserslautern, Germany (1995)
2. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press, Oxford (1995)
3. Burges, C.J.C.: A tutorial on Support Vector Machines for pattern recognition. Data Mining Knowl. Discov. **2**(2), 121–167 (1998)
4. Casey, R., Ferguson, D., Mohiuddin, K., Walach, E.: Intelligent forms processing system. Mach. Vis. Appl. **5**, 143–155 (1992)
5. Cristianini, N., John Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods. Cambridge University Press, Cambridge (2000)
6. Dengel, A., Bleisinger, R., Fein, F., Hoch, R., Hones, F., Malburg, M.: OfficeMAID - a system for office mail analysis, interpretation, and delivery. In: Proceedings of the First International Conference on Document Analysis and Recognition, Kaiserslautern, Germany, pp. 253-275 (1994)
7. Gori, M., Scarselli, F.: Are multilayer perceptrons adequate for pattern recognition and verification. IEEE Trans. Pattern Anal. Mach. Intell. **20**(11), 1121–1132 (1998)
8. Hull, J.J., Hart, P.: The infinite memory multifunction machine (IM3). In: Proceedings of the 3rd IAPR Workshop on Document Analysis Systems, Nagano, Japan, pp. 49–58 (1998)
9. Hull, J.L., Taylor, S.L., (eds.) Document Analysis Systems (II). World Scientific Publications, Singapore. (1998)
10. Koller, D., Sahami, M.: Hierarchically classifying documents using very few words. In: Proceedings on the 14th Conference on Machine Learning (1997)
11. Krishnamoorthy, M., Nagy, G., Seth, S., Viswanathan, M.: Syntactic segmentation and labeling of digitized pages and technical journals. IEEE Trans Pattern Anal. Mach. Intell. **15**(7), 737–747 (1993)
12. Lam, S.W., Spitz, A.L., Dengel, A.: An adaptive approach to document classification and understanding. In: Proceedings IAPR Workshop on Document Analysis Systems, World Scientific, Kaiserlautern, Germany, pp. 114–134 (1994)
13. Lawrence, S., Giles, C.L., Kurt Bollacker, K.: Digital libraries and autonomous citation indexing. IEEE Comput. **32**(6) 67–71 (1999)
14. Liu, D.C., Nocedal, J.: On the limited memory method for large scale optimization. Math. Program. B, **45**(3), 503–528 (1989)
15. Lopresti, D.P., Hu, J., Kashi, R.: Document analysis systems *V*. In: Lecture Notes in Computer Science, vol. 2423. Springer, Berlin Heidelberg New York (2002)
16. Maarek, Y.: Automatically organizing bookmarks per contents. In: Proceedings WWW5 (1996)
17. Manevitz, L.M., Yousef, M.: One-class SVMs for document classification. J. Mach. Learn. Res. **2**, 139–154 (2001)
18. O'Gorman, L., Kasturi, R.: Document image analysis. IEEE Computer Society Press, USA (1995)
19. Platt, J.: Fast training of Support Vector Machines using sequential minimal optimization. In: Scholkopf, B., Burges, C., Smola, A. (Eds.) Advances in Kernel Methods–Support Vector Learning. MIT Press, Cambridge (1998)
20. Sahami, M., Yusufali, S., Baldonado, M.Q.W.: SONIA: a service for organizing networked information autonomously. In: Proceedings of the 3rd ACM Conference on Digital Libraries (1998)
21. Sahami, M.: Using machine learning to improve information access. In: Thesis Computer Science Department, Stanford University, Stanford, CA. (1998)
22. Salton, G., Buckley, C.: Term weighting approaches in automatic text retrieval. Inf. Process. Manage. **24**(5), 513–523 (1988)
23. Savic, D.: Automatic classification of office documents: review of available methods and techniques. Reco. Manag. Q. 3–18 (1995)
24. Shmueli, O., Staelin, C., Greig, D., Elad, M.: Classifying Semi-Structured Documents Using Image Signatures. Hewlett-Packard Laboratories, HPL-1999-65, Palo Alto, CA (1999)
25. Spitz, A.L., Dengel, A. (eds.): Document Analysis Systems. World Scientific Publishing, Singapore (1995)
26. Srihari, S.N., Lam, S.W., Govindaraju, V., Srihari, R.K., Hull, J.J.: Document image understanding: research directions. In: Center of Excellence for Document Analysis and Recognition, CEDAR-TR-92-1 (1992)
27. Staelin, C.: Parameter Selection for Support Vector Machines. Hewlett-Packard Laboratories, HPL-2002-354R1, Palo Alto, CA (2002)
28. Taylor, S.L., Lipshutz, M.: Document understanding system for multiple document representations. In: Document Analysis Systems II, pp. 283–300. World Scientific, Singapore (1998)
29. Vapnik, V.: The nature of statistical learning theory. In: Statistics for Engineering and Information Science, 2nd edn. Springer, Berlin Heidelberg New York (2000)
30. Walischewski, H.: Automatic knowledge acquisition for spatial document interpretation. In: Proceedings of the 4th International Conference on Document Analysis and Recognition, Ulm, Germany, pp. 243–247 (1997)
31. Watanabe, T., Huang, X.: Automatic acquisition of layout knowledge for understanding business cards. In: Proceedings of the 4th International Conference on Document Analysis and Recognition, Ulm, Germany, pp. 216–220 (1997)
32. Weibel, S., Oskins, M., Vizine-Goetz, D.: Automated title page cataloging: a feasibility study. Inf. Process. Manag. **25**(2), 187–203 (1989)
33. Weiss, S., Kulikowski, C.: Computer Systems that Learn– Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems. Morgan Kaufman, New York (1991)
34. Wenzel, C., Baumann, S., Jager, T.: Advances in document classification by voting of competitive approaches. In: Document Analysis Systems II, pp. 385–405 World Scientific, Singapore (1998)
35. Wenzel, C.: Supporting information extraction from printed document by lexico-semantic pattern matching. In: Proceedings of the 4th International Conference on Document Analysis and Recognition, pp. 732–735 (1997)
36. Wnek, J.: Machine learning of generalized document templates for data extraction. In: Document Analysis Systems V. Lecture Notes in Computer Science, vol. 2423, pp. 457–468. Springer, Berlin Heidelberg New York (2002)
37. Yao, X., Liu, Y.: A new evolutionary system for evolving artificial neural networks. IEEE Transa. Neural Netw. **8**(3), 694–713 (1997)
38. Yao, X., Liu, Y.: Making use of population information in evolutionary artificial neural networks. IEEE Tran. Syst. Man Cybern. **28**(3), 417–425 (1998)