Learning Non-Generative Grammatical Models for Document Analysis

Michael Shilman, Percy Liang, and Paul Viola
Microsoft Research
Redmond, WA, 98033

Abstract

We present a general approach for the hierarchical segmentation and labeling of document layout structures. This approach models document layout as a grammar and performs a global search for the optimal parse based on a grammatical cost function. Our contribution is to utilize machine learning to discriminatively select features and set all parameters in the parsing process. Therefore, and unlike many other approaches for layout analysis, ours can easily adapt itself to a variety of document analysis problems. One need only specify the page grammar and provide a set of correctly labeled pages. We apply this technique to two document image analysis tasks: page layout structure extraction and mathematical expression interpretation. Experiments demonstrate that the learned grammars can be used to extract the document structure in 57 files from the UWIII document image database. We also show that the same framework can be used to automatically interpret printed mathematical expressions so as to recreate the original LaTeX.

1 Introduction

A 2003 review of document structure analysis lists seventeen distinct approaches for the problem [15]. Perhaps the greatest difference between the published approaches is in the definition of the problem itself. One approach may extract the title, author, and abstract of a research paper [13, 11]. Another approach may extract the articles from a newspaper [18]. The seventeen approaches (and others published since the review) use widely varying algorithms as well. Absent from most of these papers is the sense that progress made in solving one task is directly transferable to another task.

Our goal is to create a single framework which can be applied to new domains rapidly, with high confidence that the resulting system will be efficient and reliable. This is in contrast to a number of previous systems, where retargeting requires hand tuning many parameters and the selection of features for local distinctions. Our approach uses machine learning to set all parameters and to select a key subset of a features from a large generic library of features. While the features selected for two different tasks will be different, the library itself is can be used for a wide variety of tasks.

Some previous approaches have used local algorithms which group characters/words/lines in a bottom up process. Bottom up algorithms are very fast, but are often brittle. The approach taken in this paper is to build a global hierarchical and recursive description of all observations on the page (text or pixels or connected components). The set of all possible hierarchical structures is described compactly as a grammar. Dynamic programming is used to find the globally optimal parse tree for the page. Global optimization provides a principled technique for handling local ambiguity: the algorithm selects the local interpretations that maximize the global score.

Related earlier work on grammatical modeling of documents includes [3, 5, 13, 17, 22, 9]. It is not surprising that these efforts have adopted the state of the art in parsing at the time of publication. So for example the work of Krishnamoorthy et. al. uses the grammatical and parsing tools available from the programming language community [13] (see also [5, 1]). Similarly the work by Hull uses probabilistic context free grammars [8] (see also [3, 17, 16]). In the last few years there has been a rapid progress in research on grammars in the natural language community. Advances include powerful discriminative models that can be learned directly from data [14, 21]. Such models are strictly more powerful than the probabilistic context free grammars used in previous document analysis research. Simultaneously progress has been made on accelerating the parsing process [2, 12]. Motivated by these results a new wave of research on grammatical parsing for documents is likely to result.

The challenges of grammatical approaches include computational complexity, grammar design, feature selection, and parameter estimation. The focus of this paper is on feature selection and parameter estimation. We present an algorithm which can learn to parse effectively from labeled examples. The key difference from earlier published work is that a *discriminative* grammar is learned, rather than

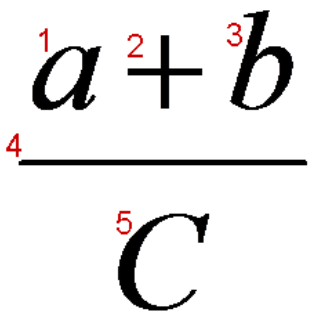


Figure 1. A simple mathematical expression containing 5 terminals. The terminals are assigned arbitrary IDs shown as small red digits.

```
(Expr → Row)
(Row → Row Item)
(Row → Item)
(Item → SubItem )
(Item → FracItem )
(Item → RawItem )
(Item → SupItem)
(FracItem → Row FracItem1)
(FracItem1 → BAR Row)
(SubItem → SupItem Row )
(SubItem → RawItem Row)
(SupItem → RawItem Row)
```

Table 1. A grammar for mathematical expressions. Note that the terminals of the grammar are not included and are referred to by the non-terminal `RawItem`.

a generative grammar. The advantages of discriminative Markov models are now well appreciated [14]. The advantages of a discriminative grammar are similarly significant. Many new types of features can be used. Additionally the grammar itself can often be radically simplified.

2 Document Grammars

One simple example examined in detail may yield some intuitions regarding the algorithms presented below. Figure 1 shows a simple printed mathematical expression with 5 terminal characters (in other applications the terminals could be connected components, pen stokes, text lines, etc). A simple grammar that can model this type of expression is shown in Table 1.

Consider the following parse for this document:

```
(Expr (Row (Item
            (FracItem
               (Row (Item (RawItem 1))
                    (Row (Item (RawItem 2))
                         (Item (RawItem 3))))
               (FracItem1 (BAR 4)
                    (Row (Item (RawItem 5))))))))
```

A printed (or scanned) document contains only the spatial arrangement of the visible symbols, and lacks the information necessary to recreate the original typographic description (as is contained in LaTeX, MathML, or Word). In contrast the parse tree of this document captures information about the document structure: that the expression is a fraction whose numerator contains a row of 3 terminals 'a', '+', and 'b', and whose denominator contains 'C'. Such a parse tree can be used to reconstruct an equivalent LaTeX (or MathML) expression. The grammatical approach can be adopted for many types of document analysis tasks, including the parsing of document structure, mathematical expressions, text information extraction, and table extraction.

This paper will restrict attention to grammars in Chomsky normal form[1], which contains productions such as `(A → B C)` and `(B → b)`. This first states that the non-terminal symbol `A` can be replaced by the non-terminal `B` followed by the non-terminal `C`. The second states that the non-terminal `B` can be replaced by terminal symbol `b`. A simple weighted grammar, or equivalently a Probabilistic Context Free Grammar (PCFG), additionally assigns a cost (or negative log probability) to every production.

While there are a number of competing parsing algorithms, one simple yet generic framework is called Chart Parsing [10]. Chart parsing attempts to fill in the entries of a chart $C(\mathtt{A}, R)$. Each entry stores the best score of a non-terminal `A` as an interpretation of the sub-sequence of terminals R. The cost of any non-terminal can be expressed as the following recurrence:

$$C(\mathtt{A}, R_0) = \min_{\substack{\mathtt{A}\rightarrow\mathtt{BC} \\ R_1 \cap R_2 = \emptyset \\ R_1 \cup R_2 = R_0}} C(\mathtt{B}, R_1) + C(\mathtt{C}, R_2) + l(\mathtt{A} \rightarrow \mathtt{BC}) \tag{1}$$

where $\{\mathtt{BC}\}$ ranges over all productions for `A`, and R_0 is a subsequence of terminals (what we will call a region), and R_1 and R_2 are subsequences which are disjoint and whose union is R_0 (i.e. they form a partition). Essentially the recurrence states that the score for `A` is computed by finding a low cost decomposition of the terminals into two disjoint sets. Each production is assigned a cost (or loss or negative log probability) in a table, $l(\mathtt{A} \rightarrow \mathtt{BC})$. The entries in the chart (sometimes called edges) can be filled in any order, either top down or bottom up. The complexity of the parsing

[1] Any more general grammar can be easily converted to a CNF grammar.

process arises from the number of chart entries that must be filled and the work required to fill each entry. The chart constructed while parsing a linear sequence of N terminals using a grammar including P non-terminals has $O(PN^2)$ entries (there are $\frac{1}{2}\binom{N}{2} \in O(N^2)$ contiguous subsequences, $\{i, j\}$ such that $0 \leq i < j$ and $j < N$). Since the work required to fill each entry is $O(N)$, the overall complexity is $O(PN^3)$.

2.1 Geometric Parsing Is Exponential

In this paper we will study algorithms for parsing terminals arranged on a two dimensional page. Unfortunately a direct application of chart parsing to two dimensional arrangements of terminals requires exponential time. The key problem is that the terminals no longer have a linear sequential order. Returning to (1), the region R_0 is now a subset, and R_1 and R_2 are subsets which are disjoint and whose union is R_0 (i.e. they form a partition). As before we can analyze the size of the chart, which is $O(P|\mathcal{P}(N)|)$ where $\mathcal{P}(N)$ is set of all subsets of N terminals. Since there are an exponential number of subsets the algorithm is exponential.

Hull introduced a geometric criteria which prunes the search in cases where the geometric component of the cost is too high [8]. Miller and Viola introduced a heuristic based on convex hulls which rejects regions R_1, R_2 that violate $\text{CHULL}(R_1) \cap R_2 = \emptyset$ or $\text{CHULL}(R_2) \cap R_1 = \emptyset$ [17]. Calling these sets *regions* is now appropriate, since each set lies within a convex region of the page. It is worth noting that if the terminals lie along a line (and therefore have a strict linear ordering) the convex hull criterion yields the $O(N^2)$ regions and is equivalent to the linear sequence used in conventional parsing.

Experiments in this paper make use of the convex hull constraint, as well as other geometric constraints that significantly reduce the set of subsets considered during parsing. These constraints combine to yield near $O(N^3)$ complexity on most types of printed documents. A set of novel and efficient geometric constraints are described elsewhere [6].

2.2 The Limitations of Generative Grammars

The basic parsing framework described in (1) provides a modest set of parameters which can be adapted using standard machine learning techniques. In essence there is one parameter for each production in the grammar and additionally a set of parameters associated with each terminal type. Models such as these are essentially PCFGs, and they lack the expressive power to model many key properties of documents. Stated in another way, the terminals of these models are statistically independent given the parse tree structure (much in the same way the observation of a Markov chain model are independent given the hidden states). For a simple grammar where a paragraph is a collection of lines ((`Par` → `Line Par`) and (`Par` → `Line`)) the appearance of the lines in a paragraph are independent of each other. Clearly the lines in a particular paragraph are far from independent, since they share many properties; for example the lines often have the same margins, or they may all be center justified, or they may have the same inter-line spacing.

This severe limitation was addressed by researchers in the document structure analysis [3, 8, 24]. They replaced the pure PCFG grammar with an attributed grammar. This is essentially equivalent to an expansion of the set of non-terminals. So rather than a grammar where a paragraph is a set of lines (all independent), the paragraph non-terminal is replaced by a `paragraph(lMargin, rMargin, lineSpace, justification)`. The terminal `line` is then rendered with respect to these attributes. When the attributes are discrete (like paragraph justification), this is exactly equivalent to duplicating the production in the grammar. The result is several types of paragraph non-terminals, for example left, right, and center justified. An explosion in grammar complexity results, with many more productions and much more ambiguity.

Continuous attributes are more problematic still. The only tractable models are those which assume that the attributes of the right hand side (non-)terminals are a simple function of those on the left hand non-terminals. For example, that the margins of the lines are equal to the margins of the paragraph plus Gaussian noise.

The main, and almost unavoidable, problem with PCFGs is that they are generative. The grammar is an attempt to accurately model the details of the printed page. This includes margin locations, lines spacing, font sizes, etc. Generative models have dominated both in natural language and in related areas such as speech (where the generative Hidden Markov Model is almost universal [20]). In the last few years related non-generative discriminative models have arisen. Discriminative grammars allow for much more powerful models of terminal dependencies without the increase in grammar complexity that arises from attributes.

3 Non-generative Grammatical Models

The first highly successful non-generative grammatical model was the Conditional Random Field[2] [14]. Recently similar insights have been applied to more complex grammatical models [21]. The basic insight is actually quite simple; the production cost in (1) can be generalized considerably without changing the complexity of the parsing process. The cost function can be expressed more generally as:

[2]The focus of the referenced paper is on Markov chain models, which are equivalent to a very simple grammar.

$l(\texttt{A} \rightarrow \texttt{BC}, R_0, R_1, R_2, \text{DOC})$ which allows the cost to depend on the the regions R_0, R_1 and R_2, and even the entire document DOC. The main restriction on $l()$ is that it cannot depend on the structure of the parse tree used to construct B and C (for this would violate the dynamic programming assumption underlying chart parsing).

This radically extended form for the cost function provides a great deal of flexibility. So for example, low cost could be assigned to paragraph hypotheses where the lines all have the same left margin (or the same right margin, or where all lines are centered on the same vertical line). This is quite different from conditioning the line attributes on the paragraph attributes. For example, one need not assign any cost function to the lines themselves. The entire cost of the paragraph hypothesis can fall to the paragraph cost function. The possibilities for cost functions are extremely broad. The features defined below include many types of region measurements and many types of statistics on the arrangements of the terminals (including non-Gaussian statistics).

One possible extension includes cost functions which are computed using pattern recognition techniques that directly analyze the *visual appearance* of the terminals or regions. Almost all previous document analysis systems process input in two stages: an image processing and pattern recognition stage which finds and recognizes characters/symbols and a geometry stage which measures geometric information about the relationships between components. This new approach directly combines geometric features with pattern recognition information to define a global cost function.

For attributed PCFGs there are straightforward and efficient algorithms for maximizing the likelihood of the observations given the grammar. So for example, if the conditional margins of the lines is assumed to be Gaussian, then the mean and the variance of this Gaussian distribution can be computed simply from the training data. In contrast, training of non-generative models, because of their complex features, is somewhat more complex.

Parameter learning can be made tractable if the cost function is restricted to a linear combination of features:

$$l(p, R_0, R_1, R_2, \text{DOC}) = \sum_i \lambda_{p,i} f_i(R_0, R_1, R_2, \text{DOC}).$$

where p is a production from the grammar. While the features themselves can be arbitrarily complex and statistically dependent, learning need only estimate the linear parameters $\lambda_{p,i}$.

4 Grammar Learning

There are a number of alternative algorithms for estimating the cost function parameters, including conditional random fields [14] and support vector machines [21]. We propose a much simpler algorithm that is easy both to implement and to understand. The key insight is that learning to parse is much like learning to classify. Our goal is to estimate a set of parameters which assigns low costs to the correct grammatical interpretation/grouping and high cost to all others. This process can be viewed as a learning problem where the examples are all possible nodes from a parse tree: $n =< p, R_0, R_1, R_2, \text{DOC} >$, where p is a production from the grammar, and $R_{\{0,1,2\}}$ are regions such that $R_1 \cap R_2 = \emptyset$ and $R_1 \cup R_2 = R_0$.

```
0) Initialize weights to zero for all productions
1) Parse a set of training examples using
     current parameters
2) For each production in the grammar
2a)    Collect all examples from all charts.
          Examples from the true parse are TRUE.
          All others are FALSE.
2b)    Train a classifier on these examples.
2c)    Update production weights.
          New weights are the cumulative sum.
3) Repeat Step 1.
```

Figure 2. Pseudo-code for the training algorithm.

Conversion into a learning problem is straightforward. First the set of features, f_i, is used to transform a node n_j into a vector of feature values $F_i^j = f_i(n_j)$. The score of a particular node can then be computed $\lambda_p \cdot F^j$, a dot product. The weights for a given production, λ_p, are adjusted so that the cost for TRUE examples is minimized and the cost for FALSE examples is maximized (note that the typical signs are reversed since the goal is assign the correct parse a low cost). Given the linear relationship between the parameters and cost, almost any simple learning algorithm can be used. We have used both perceptron weight updates and AdaBoost of decision stumps [4, 7].

The scheme used in this paper proceeds in rounds (see Figure 2). Beginning with an agnostic grammar, whose parameters are all zero, a labeled set of expressions is parsed. Not surprisingly, it would be exceedingly rare to encounter the correct parse. Rather than train using all possible nodes, learning is restricted to the set of nodes stored in the chart during parsing *plus* the nodes from the correct parse. The nodes from the correct parse are labeled TRUE, while all other nodes are labeled FALSE. A classifier is then trained to assign a negative score to all TRUE nodes and a positive score to all FALSE nodes.

The scoring function trained after one round of parsing is then used to parse the next round. Entries from the new chart are used to train the next classifier. The scores assigned by the classifiers learned in subsequent rounds are summed to yield a single final score.

The basic learning process above can be improved in a number of ways. The set of examples in the chart, while large, may not be large enough to train the classifier to achieve optimal performance. One scheme for generating more examples, is to find the K best parses. The algorithm for K best parsing is closely related to simple chart parsing. The chart is expanded to represent the K best explanations: $C(\mathtt{A}, R, K)$, while computation time increases by a factor of K^2. The resulting chart contains K times as many examples for learning.

It is also important to note that the set of examples observed from early rounds of parsing are not the same as those encountered in later rounds. As the grammar parameters are improved, the parser begins to return parses which are much more likely to be correct. The examples used from early rounds do not accurately represent this later distribution. It is important that the weights learned from early rounds not "overfit" these unusual examples. There are many mechanisms designed to prevent overfitting by controlling the complexity of the classifier. AdaBoost provides a very simple scheme for controlling complexity; simply stop training after just a few rounds.

5 Applications

In order to demonstrate the flexibility and effectiveness of this framework, solutions for two document analysis tasks are described: document layout analysis and printed equation interpretation. Many related applications are suitable as well including, segmentation and recognition of ink drawings, document table extraction, and web page structure extraction. The key differences between applications is: (1) the grammar used to describe the documents; (2) the set of features used to compute the cost functions; (3) the geometric constraints used to prune the set of admissible regions. Once these decisions are made, training data is used to set all parameters of the model.

5.1 Document Layout Analysis

One goal of document layout analysis is to determine the information necessary to convert a scanned document into a fully editable input file for a document preparation program, such as LaTeX or Word. While the text in a scanned file can be easily extracted using OCR, this information is not sufficient to produce an easily editable file. Additional information such as paragraph boundaries, columns, justification, and most importantly reading flow are necessary as well. This document structure information is often missing from PDF and Postscript files as well. Whether for scans, PDF, or Postscript, the addition of document structure information yields a living document that can be repaginated, reformatted, and edited.

```
(Page → SecList)
(SecList → Sec SecList)
(SecList → Sec)
(Sec → ColList)
(ColList → Col ColList)
(ColList → Col)
(Col → ParList)
(ParList → Par ParList)
(ParList → Par)
(Par → LineList)
(LineList → Line LineList)
(LineList → Line)
```

Table 2. Document layout grammar.

Document preparation programs frequently divide the printed page into sections. Each section has some number of columns and each column has some number of paragraphs. This recursive structure is expressed as a grammar in Table 2. Knowledge of this structure is sufficient to accurately produce an editable file from a scanned document.

Experiments are performed using the UWIII document image database [19]. The database contains scanned documents with ground truth for lines, paragraph, regions, and reading order (see Figure 3). For most documents the ground truth labels are easily converted to grammar above. Training and evaluation is performed using a set of 60 documents which include pages from research papers, books, and magazines.

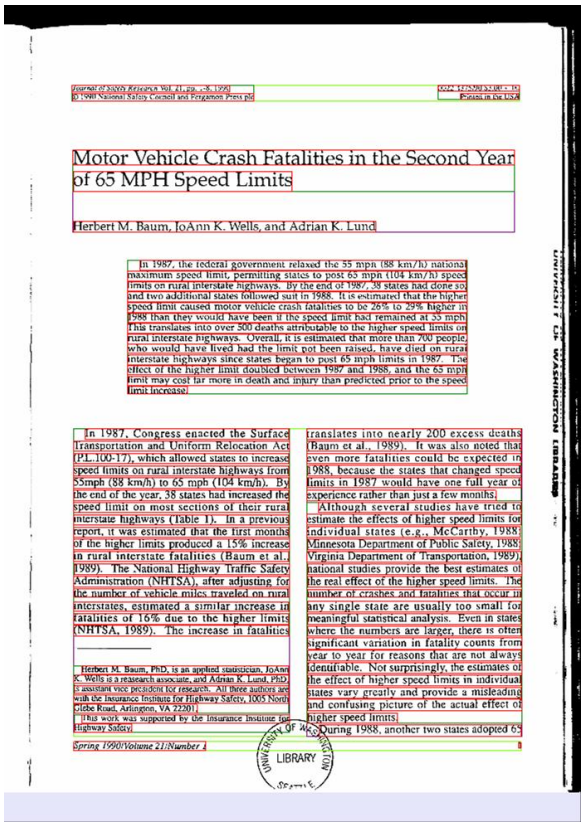
5.2 Printed Mathematics Interpretation

In the academic research community almost all new papers are made available either in PDF or Postscript. While convenient for printing, these formats do not support easy reuse or reformatting. One clear example are the included equations, which cannot be extracted, edited, or searched easily. Other examples include tables and bibliographies.

The de facto standard for scientific publication is LaTeX, in part because it provides powerful and high-quality mathematics layout. Neither PDF nor Postscript documents provide the information required to reconstruct the LaTeX equations used to generate the original.

Given a set of training LaTeX documents, a set of LaTeX macros can be used to "instrument" the document rendering process. The result is a set of instrumented DVI files which can be processed to extract the bounding boxes of characters on the page and the corresponding LaTeX expression. These macros have been applied to a set of LaTeX files made available from the ArXiv pre-print server (see Figure 4).

After post-processing the training data is a collection expressions, each a well-formed syntactic trees of terminals. These trees provide us with the opportunity to directly induce the grammar, since productions of the grammar are di-



Motor Vehicle Crash Fatalities in the Second Year of 65 MPH Speed Limits

Herbert M. Baum, JoAnn K. Wells, and Adrian K. Lund

Figure 3. An example page from the UWIII database. The input to the parsing algorithm is the bounding boxes of the lines. The output is the hierarchical decomposition into sections/columns/paragraphs.

rectly observable from the input trees[3]. The induced grammar is shown in Table 1. Note that the terminals of the grammar are not included and are referred to by the non-terminal `RawItem`. The set of `RawItems` are the characters, digits, and symbols used to build up mathematical expressions. The terminals of the grammar are the primitive connected components of black ink.

Unlike other work on mathematical parsing we do not assume that the terminals have been segmented and recognized before interpretation begins. Recognition of the terminals is an integral part of the parsing process. Every symbol type has an associated grammatical rule that describes the production of the terminals. For example `(RawItem` $\rightarrow$ `EQUALS)` and `(EQUALS` $\rightarrow$ `CC1 CC2)`, which states that the "equals sign" is made up of a pair of connected components. The cost function associated with the `EQUALS` production must learn to assign low cost to

[3]Such a grammar is often called a tree-bank grammar.

$$\frac{\beta^2}{8} T_r\left(U'(\Phi)\right)^2$$

Figure 4. An example equation used to train the mathematical expression recognizer. The input to the algorithm are the set of connected components of black ink and the output is the LaTeX necessary to regenerate the expression.

pairs of connected components that look like '='.

Overall setting up this problem is mechanically simple. The grammar is created from the example LaTeX files and the features are selected automatically from a larger set of generically valuable features which are defined below.

6 Features

The features used to learn the production scoring functions are generally applicable and useful for a wide range of tasks.

A set of geometric bounding box features have proven valuable for measuring the alignment of components. The first type are related to the bounding boxes of the sets R_0, R_1, and R_2. They measure the position of the corners, X_i, Y_i and size, W, H of the box in page coordinates. There are a total of 360 measurement features which we will refer to as $\{m_j(R)\}$. A second set of features is combinatorial and relates all pairs of box measurement features: $g(m_j(R_a), m_{j'}(R_b))$, where the a and b are $\{0, 1, 2\}$ and the function g can be addition, subtraction, multiplication, division, minimum, or maximum. A third set of features measure properties of the bounding boxes of the terminals included in the regions. This includes measuring the minimum, maximum, average, standard deviation, and median of some measurement feature evaluated across all region terminals.

Additionally, there are a large set of pattern recognition features which are designed to discriminate regions based on visual appearance. These features operate on the rendered images of the terminals in the regions (see Figure 6). Visual features are necessary when the terminals themselves must be recognized based on appearance. We adopt the rectangle features proposed by Viola and Jones [23]. They are computationally efficient and have proven effective for a wide range of tasks. Each input image is represented 121 single rectangle features sampled uniformly in location and scale. A much larger set has been used for more difficult image recognition tasks, but these have proven sufficient for these tasks.

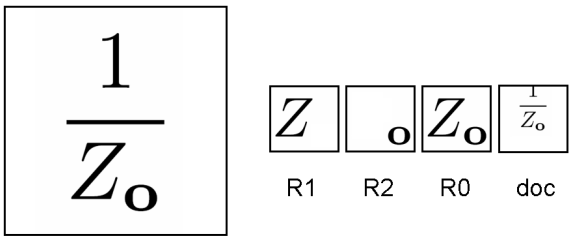


Figure 5. Left: An input mathematical expression. During parsing the expression Z_o is encountered and must be interpreted. Right: The four rendered images used as input to the production scoring process.

Geometric normalization is a critical question when constructing image classification functions. In this case we choose a reference frame which normalizes the size and location of R_0. The target is for R_0 to fill 80% of the visual image. The terminals of R_1 and R_2 are rendered in this coordinate frame. This provide the image features with an input image containing information about the relative positions of R_1 and R_2. So for example, if R_2 is a subscript, the position of its rendered components will be toward the bottom of the reference frame. Finally the terminals from the entire document are rendered in the reference frame of R_0 but with at a much smaller scale. This image encodes document "context" and can be used to perform certain types of local disambiguation.

During parsing every potential region and subregion is encoded as a set of images. When there are many regions the image encoding process, which involves image re-scaling, would naively result in great deal of computation. To avoid this computation, the integral image representation introduced by Viola and Jones is used to compute the rectangle filters at any scale with no additional cost.

7 Experiments

Two sets of experiments were performed using the features described above. The overall process for learning the grammar parameters is described in Figure 2. In each round of learning AdaBoost on decision stumps is used. AdaBoost is used for two reasons. It provides a very simple mechanism for complexity control (early stopping). It also provides a mechanism for feature selection, since each round of boosting selects a single stump which is in turn associated with a single feature.

Since the early rounds of training are likely to encounter examples which are not representative of the final distribution, AdaBoost is run on schedule of increasing complexity. The first round of boosting selects 2 weak classifiers. The second and third rounds select 4 and 8 classifier respectively. Thereafter 8 classifiers (and hence 8 features) are selected in each round of parsing.

Evaluation of parsing results is something of an art. Since no system is perfect it is valuable to define a measure that quantifies the quality of a parse that is mostly correct. One scheme is to measure the recall and precision for each type of production. The ground truth contains many examples of each production. The percentage of times each production is correctly identified is *recall*. The learned grammar yields a parse for each input example. The percentage of times these productions correspond to the correct parse is the *precision*.

The UWIII document database includes 57 files split 80-20 in three rounds of cross-validation (see Table 7). While performance on the training set is near perfect, the performance on the test set is good but far from perfect. In ongoing work (to be reported in this paper at ICCV) we are developing a larger training set. We are also investigating changes in the feature representation that may improve generalization.

For both the document and mathematical equation domains, a typical input with 80 terminals takes approximately 30 seconds to parse on a 1.7GHz Pentium 4 with 1GB of RAM.

	F1	Precision	Recall
Train:			
Average	0.96	0.97	0.96
Weighted	0.95	0.95	0.95
Test:			
Average	0.85	0.86	0.84
Weighted	0.89	0.89	0.88

Table 3. Results on the UWIII document structure extraction task. Average denotes the average performance across all productions. Weighted average assigns weight in the average based on the number of examples encountered.

The equation database includes 180 expressions and a grammar with 51 different mathematical symbols such as λ and δ. The results are reported in Table 7).

8 Conclusion

We have presented an analysis framework that can learn to simultaneously segment and recognize components of printed documents. The framework is quite general, in that

	F1	Precision	Recall
Train:			
Average	1	1	1
Test:			
Average	0.94	0.95	0.94

Table 4. Results on the mathematical expression recognition task.

all parameters of the parsing process are set using a database of training examples. We have demonstrated the effectiveness and generality of the framework by presenting two applications: page layout structure extraction and mathematical expression recognition. In the first case the input to the algorithm is a collection of lines on the page and the output is the section, column, and paragraph structure. In the second case the input is a collection of connected components on the page and the output is a set of recognized mathematical symbols and the LaTeX code necessary to reproduce the input. While the final systems are quite different, very few modifications to the learning and parsing process are necessary to produce an accurate recognition system.

References

[1] D. Blostein, J. R. Cordy, and R. Zanibbi. Applying compiler techniques to diagram recognition. In *Proceedings of the Sixteenth International Conference on Pattern Recognition*, volume 3, pages 123–136, 2002.

[2] E. Charniak, S. Goldwater, and M. Johnson. Edge-based best-first chart parsing. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 127–133, 1998.

[3] P. Chou. Recognition of equations using a two-dimensional stochastic context-free grammar. In *SPIE Conference on Visual Communications and Image Processing*, Philadelphia, PA, 1989.

[4] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *EMNLP*, 2002.

[5] A. Conway. Page grammars and page parsing. a syntactic approach to document layout recognition. In *Proceedings of the Second International Conference on Document Analysis and Recognition*, pages 761–764, Tsukuba Science City, Japan, 1993.

[6] W. for Anonymity. Efficient geometric algorithms for parsing in two dimensions. In *Submitted to ICDAR*, 2005.

[7] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory: Eurocolt '95*, pages 23–37. Springer-Verlag, 1995.

[8] J. F. Hull. Recognition of mathematics using a two-dimensional trainable context-free grammar. Master's thesis, MIT, June 1996.

[9] T. Kanungo and S. Mao. Stochastic language model for style-directed physical layout analysis of documents. In *IEEE Transactions on Image Processing*, volume 5, 2003.

[10] M. Kay. Algorithm schemata and data structures in syntactic processing. pages 35–70, 1986.

[11] J. Kim, D. Le, and G. Thoma. Automated labeling in document images. In *Document Recognition and Retrieval VIII*, volume 4307, January 2001.

[12] D. Klein and C. D. Manning. A* parsing: Fast exact viterbi parse selection. Technical Report dbpubs/2002-16, Stanford University, 2001.

[13] M. Krishnamoorthy, G. Nagy, S. Seth, and M. Viswanathan. Syntactic segmentation and labeling of digitized pages from technical journals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:737–747, 1993.

[14] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[15] S. Mao, A. Rosenfeld, and T. Kanungo. Document structure analysis algorithms: A literature survey. In *Proc. SPIE Electronic Imaging*, volume 5010, pages 197–207, January 2003.

[16] N. Matsakis. Recognition of handwritten mathematical expressions. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, May 1999.

[17] E. G. Miller and P. A. Viola. Ambiguity and constraint in mathematical expression recognition. In *Proceedings of the National Conference of Artificial Intelligence*. American Association of Artificial Intelligence, 1998.

[18] D. Niyogi and S. Srihari. Knowledge-based derivation of document logical structure. In *Third International Conference on Document Analysis and Recognition*, Montreal, Canada, 1995.

[19] I. Philips, S. Chen, and R. Haralick. Cd-rom document database standard. In *Proceedings of 2nd International Conference on Document Analysis and Recognition*, 1993.

[20] L. Rabiner. A tutorial on hidden markov models. In *IEEE*, volume 77, pages 257–286, 1989.

[21] B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. Max-margin parsing. In *Empirical Methods in Natural Language Processing (EMNLP04)*, 2004.

[22] T. Tokuyasu and P. A. Chou. Turbo recognition: a statistical approach to layout analysis. In *Proceedings of the SPIE*, volume 4307, pages 123–129, San Jose, CA, 2001.

[23] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2001.

[24] M. Viswanathan, E. Green, and M. Krishnamoorthy. Document recognition: an attribute grammar approach. In *Proc. SPIE Vol. 2660, p. 101-111, Document Recognition III, Luc M. Vincent; Jonathan J. Hull; Eds.*, pages 101–111, Mar. 1996.